Humboldt-Universität zu Berlin
Institut für Informatik
Lehrstuhl für Komplexität und Kryptographie

Entwicklung und Analyse eines biometrischen Authentifikations-Protokolls

Diplomarbeit

Benjamin Bäßler

Mat.Nr. 176054

Berlin, den 11. September 2003
Betreuer: Matthias Schwan

Danksagung

Ich danke:

- Prof. Johannes Köbler für die Betreuung dieser Arbeit als Gutachter.

- Dipl.-Inf. Matthias Schwan für eine interessante Aufgabenstellung und die Betreuung dieser Arbeit als Zweitgutachter.

- Dr. Carsten Rudolph von der Fraunhofer Gesellschaft für Sichere Telekooperation für seine geduldige und engagierte Unterstützung bei der Spezifikation des Protokolls und der Verwendung des Simple-Homomorphism-Verification-Tool.

- Will Marrero von der DePaul University Chicago für die Bereitstellung seines Model-checking-Tools.

- Mandy König, Lena Strüben und Ulrike Thumm für das Korrekturlesen.

- Heidi Neugebauer für ihre Geduld und viele Informationen zu allerlei formellen Fragen rund um diese Arbeit.

- Meinen Eltern für langjährige Geduld und finanzielle Unterstützung.

- Mady Drews für die moralische Unterstützung während der Entstehungszeit dieser Arbeit.

Einverständniserklärung

Hiermit erkläre ich mich mit der Aufstellung meiner Diplomarbeit 'Entwicklung und Analyse eines biometrischen Authentifikations-Protokolls' in der Bibliothek des Fachbereichs Informatik der Humboldt-Universität zu Berlin einverstanden.

Berlin, den 11. September 2003. Benjamin Bäßler

Selbstständigkeitserklärung

Ich versichere, die vorliegende Diplomarbeit mit dem Titel 'Entwicklung und Analyse eines biometrischen Authentifikations-Protokolls' selbstständig und unter Nutzung keiner anderen als der aufgeführten Literatur verfasst zu haben. Ich habe die Arbeit weder in dieser noch in einer ähnlichen Form einem anderen Prüfungsamt vorgelegt.

Berlin, den 11. September 2003 Benjamin Bäßler

Inhaltsverzeichnis

Einleitung

Immer öfter geistert der Begriff 'Biometrie' durch die Medien; vor allem seit den Ereignissen der letzten Jahre.
Dadurch wurde der Blick verstärkt auf biometrische Systeme gerichtet. Das Ziel eines biometrischen System ist es z.B. den Zugang zu einem gesicherten Gebäudebereich zu regeln.
Dazu müssen aber erst einmal folgende Fragen geklärt werden:

- Was versteht man unter einem biometrischen System?
- Welche biometrischen Systeme gibt es?
- Wie sicher sind biometrische Systeme?

Desweiteren ist zu beachten, dass ein einfacher Austausch z.B. der Eingabetastatur am Geldautomaten durch einen Fingerprint-Scanner nicht möglich ist, da biometrische Daten und dadurch auch die biometrischen Systeme ganz andere Eigenschaften haben, die es zu beachten gilt.

Deshalb wird in Kapitel 1 zuerst der Begriff der Biometrie geklärt und das notwendige Wissen über biometrische Systeme vermittelt.
Anschließend wird in Kapitel 2, basierend auf den 'generellen Bedrohungen für biometrische Systeme' aus den *Common Criteria* [1], untersucht, welche möglichen Angriffspunkte biometrische Template-On-Card-Systeme liefern. Ausserdem wird analysiert welche Gegenmassnahmen denkbar sind.
In Kapitel 3 werden dann, auf Basis dieser Überlegungen, die Sicherheitsziele, die ein biometrisches Template-On-Card-Protokoll zur Authentifikation erfüllen soll aufgestellt. Weiterhin werden die Massnahmen aufgezeigt, die man ergreifen kann, um die Sicherheitsziele zu erreichen. Ebenso werden zwei Template-On-Card-Protokolle für die Benutzerauthentifikation entworfen.
Daran schließt sich in Kapitel 4 eine informelle Analyse der Sicherheit eines der aufgestellten Protokolle an.
Kapitel 5 gibt dann einen Überblick über die gängigsten formalen Methoden und ihre Klassifizierung in 4 Methodentypen nach *Meadows* [26].
Den Abschluß der Arbeit bildet in Kapitel 6 die genauere Betrachtung zweier Ansätze zur Sicherheitsanalyse und die Umsetzung eines der entworfenen Protokolle in die Eingabe-Spezifikation für die jeweils zugehörigen Tools.

Kapitel 1

Biometrische Identifikationssysteme

Ziel dieses Kapitels ist es die notwendigen Begriffe im Zusammenhang mit biometrischen System einzuführen und zu erklären was ein biometrisches System ist und welche Art von Systemen man dabei unterscheiden kann.

1.1 Biometrie

In diesem Abschnitt werden der Begriff 'Biometrie' sowie die wichtigsten Begriffe im Zusammenhang mit biometrischen Systemen erklärt.

1.1.1 Was bedeutet 'Biometrie'?

Die ursprüngliche Bedeutung von Biometrie ist laut dem Online-Lexikon *wissen.de*

> *die Lehre von den Mess- und Zahlenverhältnissen der Lebewesen und ihrer Einzelteile sowie der Lebensvorgänge.*

Überträgt man diesen Begriff auf die Welt der Computer, so steht Biometrie für

> *den Identitätsnachweis von Personen unter Verwendung ihrer individuellen körperlichen Merkmale* [2].

Als 'Identitäts-Merkmale' können nur die Merkmale dienen, die bei jedem Menschen einzigartig sind, d.h. die einer einzigen Person eindeutig zugeordnet werden können. Sie müssen folgende Eigenschaften haben:

- Universalität (jeder Mensch hat dieses Merkmal)
- Einzigartigkeit (bei jedem Menschen ist es verschieden)

- Beständigkeit (es verändert sich nicht mit dem Altern)
- Erfassbarkeit (das technische System kann es quantitativ messen)

Das bekannteste und am häuftigsten verwendete Merkmal ist dabei der Fingerabdruck. Aber auch die Hand- und Venengeometrie, die Iris, die Retina, die Stimme oder das Tippverhalten haben sich als verwendbar erwiesen.

Biometrische Verfahren lassen sich in zwei Gruppen einteilen:

Statische Verfahren:	basierend auf physiologischen Merkmalen, die unveränderlich sind; z.B. der Fingerabdruck
Dynamische Verfahren:	basierend auf verhaltenstypischen Merkmalen, die veränderlich sein können; z.B. die Stimme

Denkbar sind auch Kombinationen aus beiden Gruppen, sogenannte 'Hybridverfahren'.

Die Vor- und Nachteile der jeweiligen Verfahren sind in [3] und [4] genauer dargestellt.
Der Vorteil der statistischen Verfahren liegt darin, dass die physiologischen Merkmale sich über einen langen Zeitraum kaum verändern und damit charakteristisch sind. Deshalb haben statische Verfahren eine hohe Genauigkeit und Zuverlässigkeit und sind darüber hinaus in der Anwendung bequem für den Benutzer. Der Nachteil der statischen Verfahren liegt darin, dass sie 'offen' sind, d.h für jeden sichtbar und zugänglich (mit Ausnahme der Retina). Damit lassen sich physiologische Merkmale unbemerkt durch einen Angreifer erfassen; einen Fingerabdruck z.B. von einem Glas oder die Iris durch einen Optiker. Durch ihre Beständigkeit bringen sie auch das Problem mit, dass sie nicht einfach abgeändert werden können, wenn sie einmal kompromittiert sind; im Gegensatz zu einer PIN. Die Möglichkeit diese Merkmale trotz Widerstand zu erfassen, kann im Bezug auf die Strafverfolgung als Vorteil, im Bezug auf kriminellen Missbrauch jedoch als Nachteil angesehen werden.
Der Vorteil der dynamischen Verfahren besteht darin, das ein verhaltenstypisches Merkmal jederzeit abgeändert werden und somit *nicht dauerhaft 'kompromittiert'* [3] werden kann. Der Umstand, dass die verhaltenstypischen Merkmale uncharakteristisch und unbeständig sind, führt zu *Einschränkungen bei Genauigkeit und Zuverlässigkeit* [3]. Die dynamischen Verfahren sind deshalb störungsanfällig, leicht zu überwinden und benötigen ein aufwendiges Enrollment. Ein weiterer Nachteil besteht darin, dass es sich auf Grund der genannten Eigenschaften empfiehlt sie mit einer Willenserklärung zu verbinden, *welche die Verbindlichkeit und Nichtabstreitbarkeit des jeweiligen Authentifikationsvorgangs erhöht* [3].
Der Vorteil von Hybridverfahren ist, dass sie, indem sie die Vorteile der jeweiligen Verfahren vereinigen, fehlerfreier und benutzerfreundlicher gestaltet werden können. Der Nachteil liegt, wie bei den dynamischen Verfahren, im aufwendigen

Enrollment das notwendig ist. In [3] wird auch darauf hingewiesen, dass durch Hybridverfahren beim Benutzer der *Eindruck einer Totalerfassung* entstehen kann und somit Vorbehalte gegen solche Verfahren provoziert werden können.

1.1.2 Authentifikation

Systeme die die Identität eines Benutzers prüfen nennt man Authentifikations-Systeme. Dabei gibt es drei Möglichkeiten eine Authentifikation durchzuführen:

- durch etwas das der Benutzer weiß (z.B. Passwort oder PIN).
- durch etwas das der Benutzer besitzt (z.B. Smartcard).
- durch etwas das der Benutzer ist (z.B. biometrische Eigenschaft wie Fingerabdruck).

Die heutzutage verbreitetste Art ist die 'wissensbasierte' Authentifikation, z.B. Log-In am PC durch Passwort.
Die Probleme von rein 'wissensbasierten' Authentifikationen sind:

- zu einfache Passwörter, bzw. Geburtsdaten als PIN usw.
- dadurch Empfänglichkeit für 'Hacker-Attacken'.
- Aufwand zur Rücksetzung des Passwörts bzw. der PIN, wenn man diese(s) vergisst.

Durch Kombination von wissensbasierter Authentifikation mit dem Besitz einer Smartcard kann die Sicherheit erhöht werden.

Identifikation vs. Verifikation

Ein biometrisches Authentifikations-System kann einen Benutzer auf zwei Arten authentifizieren:

- Identifikation, d.h *Feststellung der Identität* [6] oder
- Verifikation, d.h. *Bestätigung der Identität* [6]

Eine Personenidentifikation legt fest um welche Person es sich handelt und führt dazu einen 1:n-Vergleich aus. Dabei werden die aktuellen biometrischen Daten eines Benutzers mit allen in einer Datenbank vorhandenen Referenztemplates verglichen, um den Benutzer zu identifizieren. Der Benutzer wird als die diejenige Person identifiziert, deren biometrischer Referenzdatensatz in der Datenbank mit dem aktuellen biometrischen Datensatz der Person innerhalb einer festgelegten Toleranzgrenze übereinstimmt.

Bei einer Verifikation wird ein 1:1-Vergleich ausgeführt. Die aktuellen biometrischen Daten eines Benutzers werden mit dem in der Datenbank gespeicherten Referenztemplate der Person verglichen, die der Benutzer vorgibt zu sein. Stimmen diese beiden Datensätze innerhalb der festgelegten Toleranzgrenze überein, so wird die Identität der Person bestätigt.
Für beide Authentifikations-Modi ist es notwendig vor der Verwendung durch den Benutzer diesen, durch ein sogenanntes Enrollment (siehe 1.1.3.1), im System zu erfassen. Durch diese Erfassung erhält man den biometrischen Referenzdatensatz, der für den Vergleich benötigt wird.
Die in dieser Arbeit betrachteten biometrischen Authentifikations-Systeme verwenden eine (persönliche) Smartcard als Datenbank. Sie führen somit eine Verifikation aus, sind demnach Authentifikations-Systeme im Verifikationsmodus. Diese werden in dieser Arbeit von nun an als biometrische Verifikations-Systeme bezeichnet.

1.1.3 Biometrische Verifikations-Systeme

Der Vorteil der biometrischen Verfahren gegenüber den wissensbasierten ist klar: Man kann die 'Identifikationsdaten' nicht mehr vergessen, da man sie ständig bei sich trägt, außerdem kann man sie nicht verlieren. Da die biometrischen Daten nicht übertragbar sind, steigt die Sicherheit, dass eine Person, die versucht sich zu authentifizieren, im Akzeptanzfall, auch wirklich die ist, die sie vorgibt zu sein. Biometrische Daten eines Menschen sind datenschutzrechtlich eindeutig und deshalb hochsensible Daten (siehe [2] und [5]). Aus diesem Grund ist auf die Speicherung der biometrischen Daten des Benutzers in einer sicheren Datenbank sehr großen Wert zu legen.
Die biometrischen Daten in der Form, wie sie in der Datenbank gespeichert sind, werden im Folgenden als 'Referenztemplate' bezeichnet. Der Begriff 'Template' wird später noch genauer erklärt.

Ein biometrisches Verifikations-System besteht aus:

- dem Enrollment und
- der Verifikation, wobei sich diese unterteilen läßt in:
  - Aufnahme der biometrischen Daten (A)
  - Merkmalsextraktion (Template-Erstellung) (B)
  - Matching (C)
  - Speicherung des Referenztemplates (D)
  - Ausgabe an die Umgebung

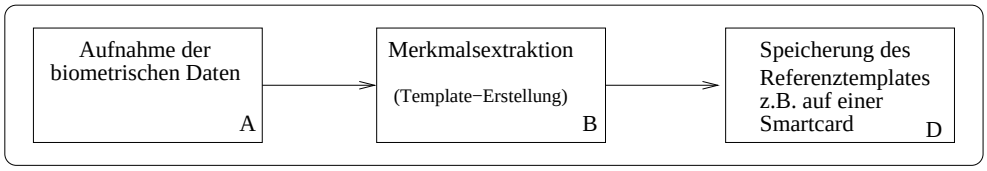

Abbildung 1.1: Ablauf des Enrollments eines neuen Benutzers

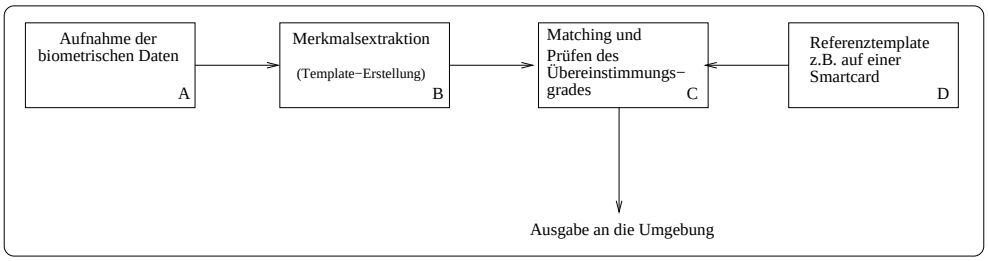

Abbildung 1.2: Ablauf einer biometrischen Verifikation

1.1.3.1 Enrollment und Verifikation

Unter dem Enrollment versteht man die Erfassung des Benutzers im System. Diese steht immer am Beginn der Verwendung des biometrischen Systems durch den Benutzer.
Dabei werden die biometrischen Daten des Benutzers erfasst, das Merkmal extrahiert (somit das Template erstellt) und anschließend als Referenztemplate gespeichert.
Abb.1.1 zeigt den Ablauf eines Enrollments. Dieses unterscheidet sich aus Sicht des Benutzers kaum von dem der Verifikation. Dazu wird das Enrollment an einem System durchgeführt, das den selben Aufbau hat, wie das spätere Verifikations-System. Der Ablauf einer Verifikation ist in Abb.1.2 dargestellt.
Der Unterschied zwischen Verifikation und Enrollment besteht lediglich darin, dass das, durch die Merkmalsextraktion in Einheit(B) erstellte Template bei der Verifikation an die Matching-Einheit (C) geschickt wird, während beim Enrollment das Template als Referenztemplate in einer Datenbank gespeichert wird.

1.1.3.2 Referenztemplates

Unter deinem Template versteht man allgemein das Produkt der Merkmalsextraktion aus den biometrischen Daten (siehe Abb.1.1 und Abb.1.2). Man unterscheidet in einem biometrischen System zwischen dem Template, welches aus den aktuellen biometrischen Daten erstellt wird und dem Referenztemplate, welches beim Enrollment erstellt und in einer Datenbank gespeichert wird.

Das Referenztemplate dient bei jedem Matchingvorgang als Vergleichsgrundlage. Deshalb empfiehlt es sich sehr großen Wert auf die Qualität des Referenztemplates beim Enrollment zu legen.

1.1.3.3 Matching

Beim Matching wird das aktuelle Template mit dem auf der Smartcard gespeicherten Referenztemplate verglichen und der Übereinstimmungsgrad bestimmt. Übersteigt dieser den geforderten Schwellwert wird die erfolgreiche Verifikation des Benutzers an die Umgebung gemeldet. Ist der Übereinstimmungsgrad zu gering, so wird dies ebenfalls dem übergeordneten System mitgeteilt.

1.1.3.4 False Rejection Rate und False Acception Rate

Zwei biometrische Datensätze vom gleichen Benutzer können nie zu 100% übereinstimmen, da zum einen selbst physiologische Merkmale sich mit dem Alterungsprozeß geringfügig ändern, zum anderen durch den Sensor oder die 'Auflageposition' des Merkmals Messabweichungen entstehen. Deshalb muss für das Matching ein Schwellwert festgelegt werden der bestimmt zu welchem Grad die aktuellen Daten mit dem Referenztemplate übereinstimmen müssen, damit der Benutzer als authentifiziert gilt. Leider läßt sich die Güte des Matchingalgorithmus nur empirisch und nicht theoretisch ermitteln.
Zur Gütebestimmung ermittelt man experimentell die False Acception Rate (FAR) und die False Rejection Rate (FRR) des Systems in Abhängigkeit eines Schwellwertes.
Wie man dabei vorgeht beschreibt [6]:

Bestimmung der FRR

Zuerst erfasst man von einer Testperson die Referenzdaten und speichert diese ab. Anschließend erstellt man mehrere neue Datensätze von derselben Testperson und ermittelt von jedem einzelnen dieser Datensätze den Übereinstimmungsgrad mit den Referenzdaten. Ein Diagramm, in dem man die Häufigkeit der Übereinstimmungen gegen den Grad der Übereinstimmungen aufträgt, ermöglicht es abzuschätzen wie groß die Wahrscheinlichkeit ist, dass die Testperson bei einem bestimmten Schwellwert abgelehnt wird. Dies ist der Fall wenn der Übereinstimmungsgrad von Referenztemplate und aktuellem Datensatz unterhalb des geforderten Übereinstimmungsgrads, also des Schwellwerts, liegt. Der prozentuale Anteil fälschlich zurückgewiesener Berechtigter bei einem bestimmten Schwellwert ist die sogenannte FRR. Die linke der beiden Abbildungen in Abb.1.3 zeigt die zu erwartende Fehlerrate FRR in Abhängigkeit vom Schwellwert. Dabei bedeutet ein Schwellwert von '1' identische Übereinstimmung und ein Schwellwert von '0' keine Übereinstimmung. Die Abbildung zeigt, dass *je größer der Schwellwert und damit der geforderte Übereinstimmungsgrad eines Datensatzes mit dem*

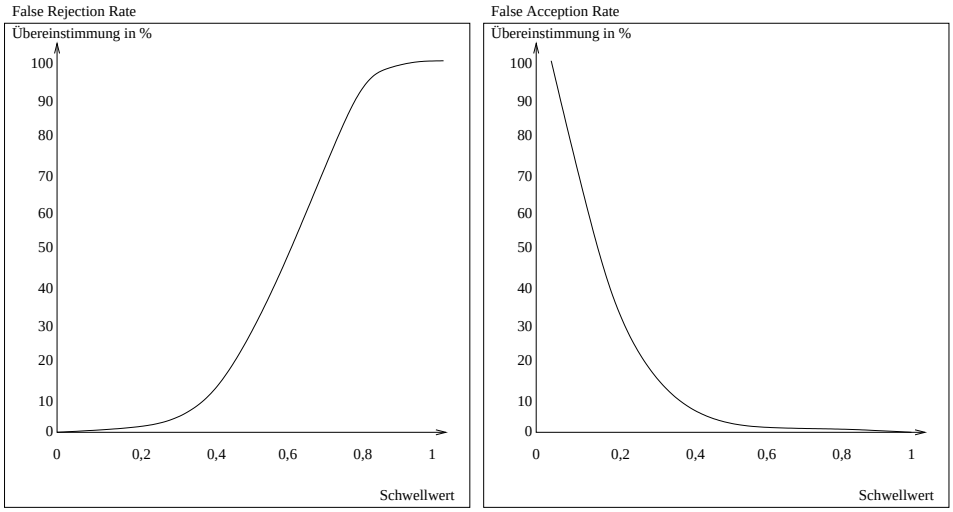

Abbildung 1.3: Verteilung des Anteils der zu Unrecht Zurückgewiesenen (FRR)/Zugelassenen (FAR) in Abhängigkeit vom Schwellwert (Quelle [6])

Referenzdatensatz gewählt wird, desto größer wird die Zahl der unrechtmäßigen Zurückweisungen [6].

Bestimmung der FAR

Hierbei prüft man die neuen Datensätze möglichst vieler verschiedener Testpersonen auf ihre Überstimmung mit den Referenzdaten der bei der Bestimmung der FRR erfassten Testperson. Auch hier ermittelt man *die Häufigkeit der Übereinstimmungen in Abhängigkeit vom Übereinstimmungsgrad* [6]. Aus dem sich daraus ergebenden Diagramm kann man nun wiederum die Wahrscheinlichkeit abschätzen, dass eine nichterfasste Testperson zugelassen wird. Die FAR ist dann der prozentuale Anteil der fälschlich zugelassenen Unberechtigten. Die rechte der beiden Abbildungen in Abb.1.3 zeigt die zu erwartende Fehlerrate FAR in Abhängigkeit vom Schwellwert. Die Abbildung zeigt, dass *je kleiner der Schwellwert und damit der geforderte Übereinstimmungsgrad eines Datensatzes mit dem Referenzdatensatz gewählt wird, desto größer wird die Zahl der Falsch-Akzeptanzen* [6].

Bestimmung der EER

Bestimmt man den Schnittpunkt der beiden sich ergebenden Kurven, so ergibt sich daraus die Equal Error Rate (ERR). Abb.1.4 zeigt den idealisierten Verlauf biometrischer Fehlerkurven. Im Schaubild sind die Fehlerraten FAR und FRR (y-Achse) in Abhängigkeit vom Schwellwert (x-Achse) dargestellt.
Der Schnittpunkt von FAR und FRR liefert die EER, d.h. die Fehlerrate an der

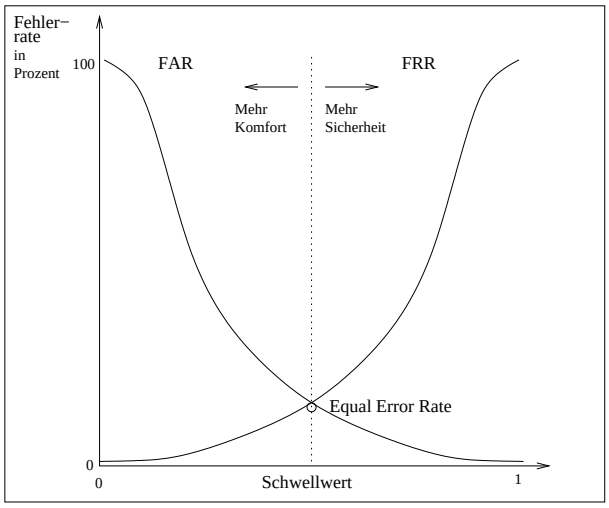

Abbildung 1.4: Typische Fehlerkurve bei biometrischen Verfahren (Quelle [6])

FAR und FRR gleich sind. Somit ist die EER *'ein Maß für die Trennfähigkeit des Systems zwischen erfassten und nichterfassten Personen'* [6].
Durch Verschieben des Schwellwertes entlang der x-Achse läßt sich, je nach Bedarf des Systems, ein definierter Sicherheitslevel erreichen.
Generell steht ein höherer Schwellwert (niedrigere FAR, höhere FRR) für Sicherheit, d.h. weniger nichterfasste Benutzer werden vom System akzeptiert. Ein niedrigerer Schwellwert (höhere FAR, niedrige FRR) steht für Komfort, d.h. wenig bis keine erfassten Benutzer werden vom System zurückgewiesen.

1.1.3.5 Adaptive Systeme

Unter einem 'adaptiven' System versteht man ein System, das bei jedem erfolgreichen Verifikationsvorgang das Referenztemplate durch die aktuellen Daten anpasst. Hintergrund dieser Systeme ist, dass sich selbst Merkmale statischer Verfahren mit dem Alter geringfügig ändern und sich durch solch ein 'adaptives' System das Referenztemplate mit ändert.
Allerdings sind 'adaptive' Systeme für Smartcards aus Sicherheitsüberlegungen nicht sinnvoll, da Smartcards gewöhnlich so konfiguriert werden, dass nur einmal, nämlich beim Enrollment, ein Referenztemplate geschrieben werden kann.

1.2 Biometrische Systeme

Auf Grund der möglichen Rollen der Smartcard in einem biometrischen System ergibt sich eine Unterteilung in drei mögliche Systeme:

1. Template-On-Card-Systeme (TOC)

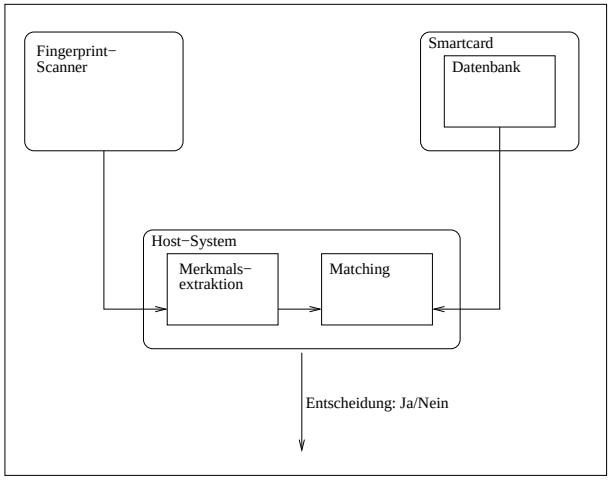

Abbildung 1.5: Aufbau eines TOC-Systems

2. Match-On-Card-Systeme (MOC)

3. System-On-Card-Systeme (SOC)

Dabei sind Template-On-Card-Systeme und Match-On-Card-Systeme (nur für Fingerprint) bereits technisch umgesetzt, während System-On-Card-Systeme noch in der Entwicklung und Forschung stecken.
Die Systeme unterscheiden sich nicht nur in ihren Sicherheitseigenschaften, sondern auch in ihrer Anwendung.
So *ist es wichtig zu beachtig, dass der Matching-Prozeß immer von der Einheit ausgeführt werden sollte, die die Identität des Benutzers feststellen muss und z.B. Zugriff auf Sicherheitsfunktionen oder Zutritt zu bestimmten Räumen gewähren muss* [7].
Im Folgenden werden nun die drei möglichen Systeme genauer dargestellt.

1.2.1 Template-On-Card-Systeme

Bei den in Abb.1.5 dargestellten TOC-Systemen finden die Merkmalsextraktion und das Matching auf dem Host statt. Der Smartcard-Reader und der Fingerprint-Scanner sind extern und über Leitungen mit dem Host verbunden. TOC-Systeme kommen zur Anwendung, wenn das Host-System wissen muss, wer die anwesende Person ist. Denn solch ein System entscheidet z.B. über den Zutritt zu einem Raum. Deshalb wird die Verifikation auf dem Host ausgeführt.
Anwendungsgebiete können sein:

- Zutritt zu einem Gebäude
- Zugang zu einem Rechner

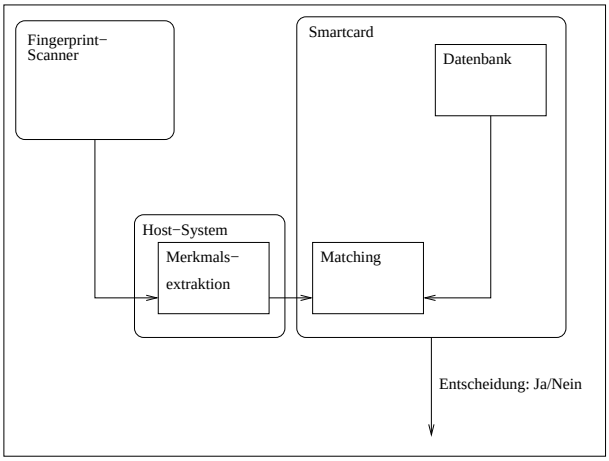

Abbildung 1.6: Aufbau eines MOC-Systems

Die Vorteile, die sich aus der Speicherung des Referenztemplates auf einer Smartcard ergeben, sind offensichtlich. Eine Smartcard ist portabel und es muss nur eine Verifikation, d.h. ein 1:1-Vergleich, ausgeführt werden. Dies erhöht die Sicherheit.
Man benötigt keine Datenbank, die sowohl gewartet werden muss als auch 'interessant' für Angriffe und aus Sicht des Datenschutzes bedenklich ist (siehe [2] und [5]).
Smartcards andererseits sind billig in der Herstellung, schützen die darauf befindlichen Daten gut gegen Angriffe von aussen und können so konfiguriert werden, dass sie ihre Daten nur nach voriger erfolgreicher gegenseitiger Authentifikation beider Teilnehmer (Smartcard und anfragende Einheit) ausgeben (siehe [9] und [10]).

1.2.2 Matching-On-Card-Systeme

Bei einem in Abb.1.6 dargestellten MOC-System wird nicht nur das Referenztemplate auf der Smartcard gespeichert, es wird auch das Matching direkt auf der Smartcard ausgeführt und das Ergebnis direkt an die Umgebung weitergegeben. Damit verläßt das Referenztemplate nie die Smartcard. Deshalb erhöht MOC die Sicherheit eines biometrischen Systems.
Ein MOC-System wird verwendet, wenn die Smartcard sowohl Smartcard-Funktionen wie die Matching-Funktion als auch, wie beim TOC-System, die auf ihr gespeicherte Daten gegen unberechtigten Zugriff schützen muss.
Zu schützende Funktionen können z.B. die Erstellung einer elektronischen Signatur sein oder die Daten-Ver-/Entschlüsselung. Zu schützende Daten sind z.B. medizinische Daten.
Ein Beispiel für ein MOC-System ist eine Signatur-Karte, bei der die biometri-

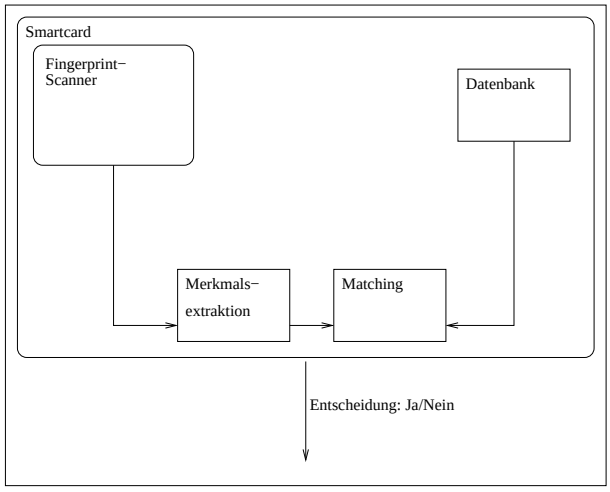

Abbildung 1.7: Aufbau eines SOC-Systems

sche Benutzer-Verifikation dazu dient, den privaten Schlüssel zur Erzeugung einer elektronischen Signatur freizuschalten.
Der Vorteil, die Verifikation der biometrischen Daten auf der Smartcard auszuführen ist z.B. dass das Referenztemplate die Smartcard nie verläßt und somit immer in der sicheren und persönlichen Umgebung verbleibt.

Eine ausführliche Beschreibung eines MOC-Systems und eine konkrete Umsetzung findet sich bei *Utimaco* [8].

1.2.3 System-On-Card-Systeme

Abb.1.7 zeigt den Aufbau eines SOC-Systems. Bei SOC-Systemen sind außer der Speicherung des Referenztemplates noch weitere komplexe Techniken wie ein biometrischer Sensor für die Datenaufnahme, sowie die Merkmalsextraktion und das Matching in die Smartcard integriert.
Bis jetzt wurden Fingerprintsensoren nur mit mäßigem Erfolg in Smartcards integriert. Dies liegt zum einen an der geringen Dicke, die die Smartcard eines SOC haben muss, zum anderen an deren noch zu hohen Herstellungskosten.
Eine andere Idee ist kryptographische Funktionen mit einem Sensor, einer Recheneinheit und einem Kartenleser in eine einzelne Einheit zu integrieren. Dies nennt man ein System-On-Token-System (SOT).
SONY und *Rainbow Technologies* haben diese Idee bereits verwirklicht [7].

Tabelle 1.1 gibt einen Überblick über die Eigenschaften der genannten biometrischen Systeme. Darin wird festgehalten:

- wie komplex die Technologie der Systeme ist,

	TOC	**MOC**	**SOC**	**SOT**
Komplexität	niedrig	mittel	sehr hoch	hoch
Matching-Einheit	Host-System	Smartcard	Smartcard	Smartcard-Chip im Token
Kosten	niedrig	niedrig	hoch	hoch
Anwendungs-Beispiele	Zugangskontrolle	Signatur	Signatur	Signatur
Verfügbarkeit	gut	gut	2-4 Jahre	selten

Tabelle 1.1: Eigenschaften der biometrischen Systeme (Quelle: TB1-Bionorm [7])

- wer das Matching ausführt,
- wie hoch die Produktionskosten der Systeme ist,
- was mögliche Einsatzgebiete der Systeme sind und
- inwieweit sie verfügbar sind bzw. wann mit ihrer Verfügbarkeit zu rechnen ist.

1.2.4 Biometrische Systeme als Allheilmittel?

Nachdem bis hierhin geklärt wurde, was man unter einem biometrischen System versteht und welche Arten von Systeme es gibt, soll nun kurz auf die Probleme biometrischer Systeme bzw. ihres Einsatzes eingegangen werden.

1.2.4.1 Ersetzen von PIN-basierten Systemen durch biometrische Systeme

Aus der Erklärung des Aufbaus von biometrischen System in 1.1.3 ergibt sich, dass der Matching-Algorithmus bei biometrischen Systemen eine andere Rolle spielt als in PIN-basierten Systemen. Während bei den PIN-basierten Systemen der Übereinstimmungsgrad zwischen dem aktuellen Template (der eingegeben Zahl) und dem Referenztemplate (der auf der Karte gespeicherten Zahl) für eine erfolgreiche Verifikation immer 100% sein kann und auch muss, ist es notwendig für ein biometrisches System einen Schwellwert festzulegen.
Deshalb und wegen den in Kapitel 2 beschriebenen sich ergebenden neuen Angriffsmöglichkeiten muss, wenn man ein PIN-basiertes System durch ein biometrisches System ersetzt, das gesamte Verifikations-Protokoll erneuert bzw. umgeschrieben werden.

1.2.4.2 Diskriminierung durch biometrische Systeme

Dieser Aspekt biometrischer Protokolle wird in [6] *in Abschnitt 8.5.* betrachtet. Die Autoren weisen darin darauf hin, dass *es kein körperliches Merkmal gibt, das bei allen Menschen überhaupt oder in gleich starker Ausprägung vorkommt.* Daraus ergeben sich zwei Punkte:

- Biometrische Systeme müssen mit einer Testgruppe getestet werden, die die spätere Nutzergruppe repräsentiert.
  So macht z.B. der Test eines Gesichtserkennungs-Systems für den asiatischen Markt mit einer Testgruppe aus Europäern keinen Sinn.

- Man muss betrachten, inwieweit und wodurch ein System bestimmte Nutzer diskriminiert und wie man dies beheben kann.

Folgende drei Aspekte führt [6] dabei an:

1. *Ausgrenzung durch das verwendete Merkmal*

   Die Ausprägung von Merkmalen kann durch genetische Einflüsse oder Abnutzung beeinträchtigt sein. Dies kann dazu führen, dass solch ein Nutzer entweder gar nicht erst im System erfasst werden kann (*failure-to-enroll*) oder die Falschzurückweisungsrate (FRR) stark ansteigt und eine Diskrimierung des Nutzers bewirkt.
   Demnach müssen Maßnahmen getroffen werden, um solchen Nutzern eine gleichwertige Nutzung des Systems zu ermöglichen.

2. *Ausgrenzung aufgrund personenbezogener Besonderheiten*

   Auch Erkrankungen, körperliche Besonderheiten oder Behinderungen wie z.B. Taubheit, Stummheit, Blindheit oder Contagan-Schäden können dazu führen, dass eine Person ein Verfahren nicht anwenden kann.
   Auch hierfür sind geeignete Maßnahmen zu treffen.

3. *Notwendigkeit von Ersatzverfahren*

   Wegen 1. und 2. ist es notwendig, dass **alle biometrische Systeme** ein Ersatzverfahren anbieten. Dieses Ersatzverfahren ist auch deswegen notwendig, weil kein Nutzer zur Verwendung biometrischer Verfahren gezwungen werden darf und ihm dadurch keine Nachteile entstehen dürfen. Als Ersatzverfahren bietet sich somit immer die Beibehaltung des PIN-Verfahrens an, wobei prinzipiell auch die Verwendung eines weiteren biometrischen Verfahrens, das ein anderes biometrisches Merkmal nutzt, denkbar wäre.

Folgerung

Dieses Kapitel hat eine Einführung in Biometrie, in Authentifikation und biometrische Systeme gegeben. Es hat die notwendigen Begriffe eingeführt und außerdem die verschiedenen biometrischen Authentifikations-Systeme, die mit einer Smartcard arbeiten vorgestellt. Für Systeme im Verifikationsmodus wurde die Bezeichnung 'Verifikations-Systeme' festgelegt.
Das folgende Kapitel beschäftigt sich nun mit den Bedrohungen, die sich voralem für biometrische Verifikations-Systeme ergeben und zeigt mögliche Gegenmassnahmen auf.

Kapitel 2

Bedrohungen für Authentifikations-Systeme

In diesem Abschnitt werden Authentifikationssysteme auf ihre möglichen Bedrohungen untersucht.
Dabei wird zwischen:

- Bedrohungen die alle Authentifikations-Systeme betreffen,
- Bedrohungen die sich aus der Verwendung von Biometrie ergeben und
- Bedrohungen die speziell biometrische TOC-Systeme betreffen

unterschieden.

2.1 Allgemeine Bedrohungen

Unter allgemeinen Bedrohungen sind Angriffsmöglichkeiten gemeint, die jedes Authentifizierungs-System betreffen.
Dies sind z.B.:

- Wiederholungsangriffe, wie z.B. Ausprobieren aller möglichen Zahlenkombinationen einer PIN.
- alternative Authentifizierungsmöglichkeiten.
- Denial-Of-Service (DOS).

Alle diese Bedrohungen sind bekannt und mögliche Gegenmassnahmen bei den heutigen Authentifizierungs-Systemen verwirklicht. Um Wiederholungsangriffe zu verhindern wurden Fehlbedienungszähler eingeführt. Alternative Authentifizierungsmöglichkeiten müssen mindestens dieselbe Sicherheit gewährleisten wie

der Haupt-Authentifikationsvorgang. Für DOS-Angriffe wurden sichere Ausfallzustände definiert. Allerdings lassen sich solche Angriffe auch nicht durch ein sicheres Authentifizierungs-Protokoll verhindern.
Interessanter sind deshalb die neuen und speziellen Bedrohungen, die sich aus der Verwendung von Biometrie ergeben.

2.2 Bedrohungen für biometrische Systeme im Verifikationsmodus

Das Interesse dieser Arbeit richtet sich auf TOC-Systeme, die mit dem Fingerabdruck als biometrisches Merkmal arbeiten und damit immer Verifikations-Systeme sind.
Durch die Verwendung von Biometrie entstehen neue Angriffsmöglichkeiten.
Diese ergeben sich zum einen durch die 'Öffentlichkeit' mancher biometrischer Merkmale, wie z.B. dem Fingerabdruck, zum anderen aus dem Matching-Verfahren, bei dem nie eine 100%ige Übereinstimmung erzielt werden kann (siehe FAR/FRR).
Dabei kann man grundsätzlich zwischen *physikalischen Angriffen*, die sich direkt gegen die Einheiten des Systems richten (z.B. Aufbruch und Veränderung des Smartcard-Readers) und *Angriffen der Verbindungen bzw. der Datenübertragung* (z.B. Abhören oder Einspielen von Daten) unterscheiden.
Eine genauere Aufstellung der möglichen Angriffe und der denkbaren Gegenmassnahmen wird in 2.4 dargestellt.

2.3 Bedrohungen für biometrische Systeme im Identifikationsmodus

Da sich biometrische Identifikations-Systeme nur durch ein anderes Vorgehen beim Matching und die Speicherung der Referenztemplates aller Benutzer in einer Datenbank von den Verifikations-Systemen unterscheiden, wird hier auf eine eigene Darstellung der Gefahren verzichtet.
Generell ist zu sagen, dass die Speicherung des Referenztemplates in einer Datenbank einen sehr guten Schreib- und Leseschutz der Datenbank voraussetzt.
Zum anderen erhöht der 1:n-Vergleich bei der Identifikation die Erfolgswahrscheinlichkeit der Angriffe mit einem fremden oder gefälschten biometrischen Merkmal.

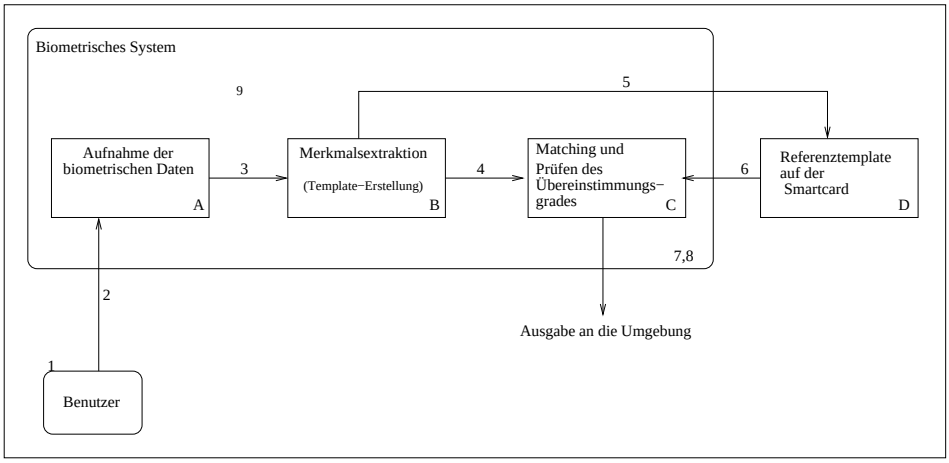

Abbildung 2.1: Schema für die Bedrohungsanalyse

2.4 Bedrohungen und Gegenmassnahmen

Bei der gesamten Bedrohungsanalyse soll davon ausgegangen werden, dass der Administrator nicht in betrügerischer Absicht oder unabsichtlich 'falsch' handelt, da die dadurch auftretenden Angriffsmöglichkeiten auch durch ein geeignetes Protokoll nicht zu verhindern sind. Dazu gehören z.B. das (unabsichtliche) Verändern des Matching-Schwellwerts, des Referenztemplates oder des Fehlbedienungszählers. Um diese Bedrohungen zu verhindern wäre es vorstellbar, eine Art '4-Augen-Prinzip' für sensible Änderungen am System einzuführen.
Desweiteren nehmen wir eine Smartcard als eine vollkommen sichere Datenbank an. Mögliche Angriffe und deren Verhinderung werden in [9] und [10] beschrieben. Zudem wird für die Übertragung der Daten eine symmetrische Verschlüsselung zu Grunde gelegt.

Ausgangspunkt für die Analyse ist das aus den in 1.1.3 gezeigten Schemen zusammengesetzte Schema zum Systemaufbau. Dieses ist in Abb.2.1 dargestellt.
Die unten aufgeführten Bedrohungen basieren auf den *Common Criteria - Biometric Evaluation Methodology Supplement (BEM)* [1]. Die Zahlen im Bild und in der Analyse stehen für die Angriffspunkte.

Eine andere Darstellung möglicher Angriffe findet sich in [11], wo auch einige mögliche Gegenmassnahmen dargestellt werden.

Im Folgenden werden nun die möglichen Angriffe auf solche Systeme dargestellt. Diese werden nach den möglichen Angriffsstellen geordnet. In Abb.2.1 steht jede Zahl für eine mögliche Angriffsstelle. Die möglichen Gegenmassnahmen folgen jeweils im Anschluß.

1. Bedrohungen durch den Benutzer:

Ein berechtigter Benutzer bietet unbewußt, unfreiwillig (Zwang) oder absichtlich (Zusammenarbeit) sein biometrisches Muster dem Angreifer an.

1.1 *Der Angreifer gelangt in Besitz des biometrischen Musters eines berechtigten Benutzers, z.B. durch Abnahme des Fingerabdrucks von einem Glas.*

1.2 *Der Angreifer stiehlt ein biometrisches Muster eines berechtigten Benutzers z.B. durch Abschneiden des erfassten Fingers oder Installieren eines falschen biometrischen Lesegerätes, um das biometrische Muster aufzunehmen.*

1.3 *Der berechtigte Benutzer stellt sein biometrisches Muster dem Angreifer zur Verfügung (Zusammenarbeit).*

1.4 *Der berechtigte Benutzer verändert sein biometrisches Muster um einen Angriff durch den Angreifer zu unterstützen.*

Gegenmassnahmen für: 1. Bedrohungen durch den Benutzer:
Die Angriffe 1.1-1.3 gehen alle davon aus, dass es möglich ist aus einem vorhandenen Fingerabdruck einen künstlichen Finger zu bauen und dessen Fingerabdruck dem Fingerprintleser zu präsentieren, so daß dieser der Fingerabdruck als echt akzeptiert. Verschiedene Tests haben ergeben, dass dies bei den vorhandenen Systemen leider auch tatsächlich der Fall ist [12], da die Fingerprint-Scanner dieser Systeme über keine ausreichende 'Lebenderkennung' verfügen.
Die Firma Optel hat jedoch einen Fingerprint-Scanner entwickelt der mit Ultraschall arbeitet und zusätzlich zum Fingerabdruck auch dessen Blutfluss untersucht und das 'Material' des Fingers bestimmt. Somit können sowohl abgeschnittene als auch künstliche Finger als solche erkannt und zurückgewiesen werden [13].

Um die Möglichkeit einer Installation eines falschen biometrischen Lesegerätes an einem TOC-System, das wie eine ATM (Automatic Teller Machine) aufgebaut ist, zu verhindern, schlagen *Hachez/Koeune/Quisquater* [11] die Nutzung der Smartcard und eines vertraulichen (persönlichen) Gerätes mit graphischer Benutzeroberfläche vor und ein Protokoll durch das sich das Lesegerät gegenüber dem Benutzer vor Präsentation der Benutzerdaten authentifiziert.
In diesem Zusammenhang ist auch interessant, dass die Firma Optel auf Basis ihres Ultraschall-Fingerprint-Scanners ein Kartenterminal entwickelt hat das Tastatur, Display, Kartenleser und Identitätsprüfer in einem Gerät vereint [14].

Bedrohung 1.4 macht nur Sinn bei einem adaptiven System oder wenn schon beim Erstellen des Referenztemplates für die Template-Datenbank ein falscher Finger präsentiert wird.
Da die im betrachteten System verwendeten Smartcards, um Fälschungen vorzubeugen, schreibgeschützt werden müssen ist ein adaptives TOC-System nicht

möglich.
Die Verwendung eines künstlichen Fingers bei der Erstellung des Referenztemplates wird bei Bedrohung 5.1 betrachtet.

2. Bedrohungen im Bereich der Aufnahmeeinheit:

2.1 *Der Angreifer verwendet sein eigenes biometrisches Muster in einem 'zero-effort' Täuschungsversuch um die Identität* ***(a)*** *eines zufällig gewählten berechtigten Benutzers,* ***(b)*** *eines ausgesuchten schwachen biometrischen Templates oder* ***(c)*** *eines berechtigten Benutzers mit einem dem biometrischen Muster des Angreifers ähnlichen biometrischen Muster (z.B. eines Zwillings) anzunehmen.*

2.2 *Der Angreifer verändert sein eigenes Verhalten (z.B. Stimme, Unterschrift) oder Aussehen (z.B. Gesicht, Hand) in einem Täuschungsversuch um die Identität* ***(a)*** *eines ausgesuchten berechtigten Benutzers oder* ***(b)*** *eines ausgesuchten schwachen biometrischen Templates anzunehmen.*

2.3 *Der Angreifer verwendet ein künstliches biometrisches Muster (z.B. gefälschter Fingerabdruck, Stimmaufnahme) bei einem Versuch die Identität* ***(a)*** *eines ausgesuchten berechtigten Benutzer oder* ***(b)*** *eines ausgesuchten schwachen biometrischen Templates anzunehmen.*

2.4 *Der Angreifer verwendet ein qualitativ schlechtes oder nichtbiometrisches Muster beim Versuch ein schwaches oder qualitativ normales biometrisches Template zu matchen.*

2.5 *Der Angreifer verwendet ein übriggebliebenes biometrisches Bild, das auf dem biometrischen System zurückgelassen wurde (typischerweise einen latenten Fingerabdruck) beim Versuch die Identität des letzten berechtigten Benutzers anzunehmen.*

2.6 *Der Angreifer verwendet sein eigenes biometrisches Muster, nachdem das biometrische Muster des Angreifers* ***(a)*** *auf einem gefälschten persönlichen Datenträger gespeichert wurde z.B Smartcard oder* ***(b)*** *auf illegalem Weg direkt in das Vergleichssystem eingegeben wurde.*

Gegenmassnahmen für : 2. Bedrohungen im Bereich der Aufnahmeeinheit

Da in dieser Arbeit speziell die Bedrohungen für die Authentifikation an einem Template-On-Card-System (TOC-Systeme) untersucht werden verringern sich die Bedrohungen 2.1-2.3 weitreichend. Dadurch ist das Szenario von Bedrohung 2.1.(a) folgendes:
Der Angreifer hat eine gültige Smartcard gefunden und versucht sich mit seinem biometrischen Muster Zugang zu verschaffen.
Ob solch ein Versuch erfolgreich ist hängt von der *FAR* ab. Somit ist die FAR geeignet zu wählen, um die Erfolgswahrscheinlichkeit eines solchen Angriffs möglichst gering zu halten. Was dabei zu beachten ist wurde in *Abschnitt 1.1.3* erläutert. Desweiteren kann man durch die Einführung eines Fehlbedienungszählers die Erfolgswahrscheinlichkeit dieses Angriffs weiter verringern.

Da Smartcards so konstruiert werden können, dass sie lese- und schreibgeschützt sind kann der Angriff aus Bedrohung 2.1.(b) nur in Zusammenarbeit mit einem berechtigten Benutzer erfolgen, welcher beim Enrollment bewusst ein schlechtes Template erzeugt (siehe Bedrohung 5.1.) und dann seine Smartcard an den Angreifer weitergibt. Demnach gilt es Bedrohung 5.1. auszuschliessen.
Dem sehr speziellen Fall der Bedrohung 2.1.(c) kann, da selbst eineiige Zwillinge nicht vollkommen identische Fingerabdrücke haben, wiederum nur durch eine geeignete Wahl der FAR entgegengetreten werden.

Da unser TOC-System Fingerabdrücke nutzt, ist die Bedrohung 2.2 eher gering, da sie lediglich ein Spezialfall von Bedrohung 2.1 ist.

Bei Bedrohung 2.3 stellt sich das Problem in den Besitz einer Smartcard zu kommen und/oder zu wissen, dass sie ein schwaches Template enthält.
Zudem erkennt der, schon bei Bedrohung 1.1-1.3 angeführte, Ultraschall-Fingerabdruckleser von Optel [13] einen künstlichen Finger. Somit kann Bedrohung 2.3 als behebbar angesehen werden. Auch Fingerprint-Scanner, die zusätzlich zum Fingerabdruck noch den Blutfluß messen, können viele künstliche Finger als solche erkennen.

Generell gilt: Die Bedrohungen 2.1-2.3 können durch Speicherung des Referenztemplates auf einer Smartcard stark verringert werden.

Um Bedrohung 2.4 entgegenzuwirken muss der Fingerprint-Scanner, sowohl beim Enrollment als auch bei der Verifikation, schlechte Fingerabdrücke erkennen können und diese dann zurückweisen. Offen bleibt dabei, wie ein 'schlechter Fingerabdruck' zu definieren ist und wie ein Fingerprint-Scanner die Qualität eines Fingerabdrucks erkennt.

Bedrohung 2.5 dürfte im Moment die wohl technisch grösste Bedrohung darstellen. Überdeckt man solch einen latenten Fingerabdruck mit Folie und legt einen 'fremden' Finger auf, so können selbst Fingerprint-Scanner die sowohl Fingerabdruck als auch Blutfluß messen, damit teilweise überwunden werden.
Der bei den Bedrohungen 1.1-1.3 beschriebene Ultraschall-Fingerprint-Scanner jedoch erkennt in solch einem Fall die Folie auf Grund ihres Materials und kann damit auch diesen Angriff verhindern.

Bedrohung 2.6.(a) kann man durch gegenseitige Authentifikation von Smartcard und Lesegerät verhindern. Wie eine gegenseitige Authentifikation ausgeführt werden kann ist in *ISO/IEC JTC 1/SC 27 N2019* [17] beschrieben.
Die gegenseitige Authentifikation der Komponenten und die Verschlüsselung der Daten ermöglichen es auch Bedrohung 2.6.(b) zu verhindern. Dazu muss der Fingerprint-Scanner alle seine Daten verschlüsseln, bevor er sie an das Vergleichssystem schickt. Das Vergleichssystem prüft dann die Daten auf Authentizität, bevor ein Vergleich erfolgt.
Leider sind die heute erhältlichen Fingerprint-Scanner nicht in der Lage Daten zu verschlüsseln. Technisch ist es jedoch möglich Fingerabdruck-Scanner mit einer

Verschlüsselungsfunktion zu bauen.
Geht man davon aus, dass das Referenztemplate auf der Smartcard nicht gefälscht werden kann **und** die Authentifikation zwischen Smartcard und Host ungestört läuft, dann ist Angriff 2.6 nicht durchführbar.

3. Bedrohungen bei der Übertragung zwischen Aufnahme- und Extraktionseinheit:

3.1 *Der Angreifer hört ein biometrisches Muster während der Übertragung zwischen Aufnahme- und Extraktionseinheit ab.*

3.2 *Der Angreifer spielt ein berechtigtes biometrisches Muster direkt in die Extraktionseinheit ein, z.B. durch einen Replay-Angriff, der die Aufnahmeeinheit umgeht.*

Gegenmassnahmen für: 3. Bedrohungen bei der Übertragung zwischen Aufnahme- und Extraktionseinheit

Da es heutzutage noch nicht möglich ist aus abgehörten biometrischen Daten einen 'echten' Fingerabdruck nachzubauen ist Bedrohung 3.1 im Moment gering. Um aber dem Fortschritt der Technik vorzubeugen und somit auch diese Bedrohung zu verhindern, bietet es sich an, nach einer gegenseitigen Authentifikation der Einheiten, die Daten verschlüsselt zu übertragen.
Jedoch beinhalten die meisten heute käuflichen Fingerprint-Scanner keine solche Funktionalität, weshalb dies bei der Entwicklung eines sicheren Protokolls zu beachten ist.

Den in Bedrohung 3.2. dargestellten 'Replay-Angriff' kann man verhindern, indem man den gesendeten Daten einen Zeitstempel hinzufügt. Dabei ist die 'Frische' immer von der empfangenden Einheit zu prüfen.
Ein Umgehen der Aufnahmeeinheit und direktes Einspielen wird durch die gegenseitige Authentifikation der Einheiten vor dem Datenaustausch verhindert.

Die einfachste Methode alle in 3. genannten Bedrohungen zu verhindern wäre es sowohl Aufnahmeeinheit als auch Smartcard-Reader in eine einbruchssichere Box zu integrieren. Generell gilt für die genannten Bedrohungen, dass sie hauptsächlich für Identifikations-Systeme relevant sind. Denn bei TOC/MOC-Systemen benötigt man zusätzlich zum abgefangenen biometrischen Muster noch die Smartcard, die das Referenztemplate liefert.

4. Bedrohungen bei der Übertragung zwischen Extraktions- und Vergleichseinheit:

4.1 *Der Angreifer hört ein extrahiertes biometrisches Merkmal während der Übertragung zwischen Extraktions- und Vergleichseinheit ab.*

4.2 *Der Angreifer spielt extrahiertes biometrisches Merkmal direkt ins Vergleichssystem ein.*

Gegenmassnahmen für: 4. Bedrohungen bei der Übertragung zwischen Extraktions- und Vergleichseinheit
Auch für Bedrohung 4.1 reicht, wie für 3.1, die gegenseitige Authentifikation und das Verschlüsseln der Daten um einem sinnvollen Abhören der Daten entgegenzuwirken.

Bedrohung 4.2 entspricht der Bedrohung 3.2 und kann auf dieselbe Art verhindert werden.

In dem TOC-System, das in dieser Arbeit betrachtet wird, befinden sich Extraktions- und Vergleichseinheit (d.h. das Matching) beide im Host und somit per Definition in einer einbruchssicheren Umgebung. Dadurch spielen die Bedrohungen aus 4. für das betrachtete System keine Rolle.
Wie bei 3. gilt auch bei den Bedrohungen von 4., dass sie für TOC/MOC-Systeme nicht sehr bedeutsam sind, sofern der Angreifer nicht im Besitz des Referenztemplates, d.h. der Smartcard, ist.

5. Bedrohungen bei der Erzeugung bzw. Speicherung des Referenztemplates:

5.1 *Der Benutzer erzeugt ein qualitativ schlechtes, stark voneinander abweichendes oder nicht-biometrisches Muster, modifiziert sein eigenes Verhalten oder verwendet ein künstliches Muster mit der Absicht ein schwaches biometrisches Template zu erzeugen.*

5.2 *Ein nicht-berechtigter Benutzer wird aufgenommen, durch:* ***(a)*** *Administrator-Fehler, z.B. Legitimation nicht ausreichend geprüft oder* ***(b)*** *Abfangen des Templates eines berechtigten Benutzers und Ersetzen durch das Template des Angreifers während der Aufnahme.*

5.3 *Das biometrische Template des Angreifers ist auf einem gefälschten persönlichen Datenträger gespeichert (z.B. Smartcard).*

5.4 *Der Angreifer stiehlt das biometrische Referenztemplate eines berechtigten Benutzers aus der Template-Datenbank oder aus einem anderen biometrischen System.*

5.5 *Der Angreifer modifiziert oder löscht biometrische Templates aus der Template-Datenbank.*

5.6 *Der Angreifer hört ein berechtigtes biometrisches Referenztemplate während der Übertragung zwischen Extraktionseinheit und Template-Datenbank ab.*

Gegenmassnahmen für: 5. Bedrohungen bei der Erzeugung des Referenztemplates
Diese Bedrohungen betrachten die Möglichkeiten schon bei der Erzeugung des Referenztemplates für die Template-Datenbank, d.h. dem Enrollment, einen Fingerabdruck zu erzeugen, der einen Angriff ermöglicht bzw. dessen Chancen verbessert.
Da das Enrollment einen sensiblen und wichtigen Bereich darstellt sollte hier-

auf große Sorgfalt gelegt werden. D.h. auch die Benutzer sollte vorher gut über die Funktionsweise informiert und beim Enrollment selber unterstützt werden. Ebenso sollte das Enrollment an einem 'stand-alone-Rechner' stattfinden, der denselben Aufbau wie das spätere Authentifikations-System hat.
Auch sollte bei TOC-Systemen verhindert werden, dass ein berechtigter Benutzer die Möglichkeit hat sein eigenes Referenztemplate zu lesen oder zu verändern.

Um Bedrohung 5.1 zu vermeiden sollte die Extraktionseinheit 'schlechte' Fingerabdrücke zurückweisen und eine Neu-Aufnahme fordern. Außerdem muss der Fingerprint-Scanner künstliche Finger als solche erkennen und Alarm geben.

Wenn auch beim Enrollment die Daten vom Leser zur Template-Datenbank verschlüsselt übertragen werden, dann verhindert dies ein sinnvolles Abfangen und Ersetzen, wie in Bedrohung 5.2 beschrieben.
Dasselbe ist erfüllt, wenn man die Umgebung in der das Enrollment stattfindet als 'tamper-proof', also einbruchssicher, annimmt.
Adminstrator-Fehler, wie eine unzureichende Legitimations-Prüfung sind technisch nicht zu vermeiden.

Soll eine gefälschte Smartcard, wie in Bedrohung 5.3. beschrieben, vom System als gültig akzeptiert werden, so müsste sie über das Schlüsselpaar des Root verfügen, um eine erfolgreiche gegenseitige Authentifikation mit dem Host durchzuführen. (Für eine genauere Erklärung der gegenseitigen Authentifikation siehe Abschnitt 3.2.1)
Nimmt man den geheimen Schlüssel des Root als nicht brechbar an, so verhindert die gegenseitige Authentifikation diese Bedrohung.
Zudem ist darauf zu achten, dass die Smartcard ihr Referenztemplate erst nach erfolgreicher Authentifikation freigibt.

Die *SecCommerce Technologies AG* beschreibt in [9] warum man eine Smartcard als eine sichere Datenbank ansehen kann. Damit wird das Stehlen des Referenztemplates, wie in Bedohung 5.4. beschrieben, aus der Smartcard als nicht möglich angenommen.
Für die Authentifizierung an anderen biometrischen Systemen gilt, wie für alle Authentifikations-Systeme, dass möglichst kein Merkmal (Passwort) zweimal verwendet werden sollte; d.h. möglichst an jedem biometrischen System sollte ein anderer Finger genutzt werden. Hierbei können die beschränkten Ressourcen an biometrischen Merkmalen zu einem Problem werden. Einen möglichen Ausweg könnte dabei die 'virtuelle PIN' von *Cifro* bieten [15].
Zudem wäre ein Sicherheitsstandard für biometrische Systeme hilfreich, damit das Brechen jedes biometrischen Systems gleich schwer ist.

Die in [9] beschriebenen Eigenschaften der Smartcard verhindern auch das in Bedrohung 5.5. beschriebene Löschen oder Verändern des Referenztemplates.

Bedrohung 5.6. läßt sich auf dieselbe Weise wie die Bedrohungen 4.1. und 3.1. verhindern.

6. Bedrohungen bei der Übertragung zwischen Template-Datenbank und Vergleichseinheit:

6.1 *Der Angreifer hört ein berechtigtes biometrisches Referenztemplate während der Übertragung zwischen Template-Datenbank und Vergleichseinheit ab.*

6.2 *Der Angreifer spielt sein eigenes Template direkt ins Vergleichssystem ein.*

Gegenmassnahmen für: 6. Bedrohungen bei der Übertragung zwischen Template-Datenbank und Vergleichseinheit

Eine Smartcard kann Daten vor dem Verschicken verschlüsseln. Damit wird Bedrohung 6.1 verhindert.

Wird vor der Übertragung eine Authentifikation zwischen Smartcard und Vergleichseinheit durchgeführt und dabei ein Sessionkey vereinbart, dann wird dadurch das direkte Einspielen in Bedrohung 6.2 verhindert.

7. Bedrohungen im Bereich der Hardware-Komponenten:

z.B. Biometrie-Sensor, Portal-Hardware, eingebaute Schaltkreise, I/O-Hardware, Computer, etc.

7.1 *Der Angreifer verändert, modifiziert, überbrückt oder deaktiviert eine oder mehrere Hardware-Komponenten.*

7.2 *Der Angreifer hört ab/spielt ein berechtigtes biometrisches Template aus/in eine oder mehrere(n) Hardware-Komponenten.*

8. Bedrohungen im Bereich der Software-Komponenten:

8.1 *Der Angreifer verändert, modifiziert, überbrückt oder deaktiviert eine oder mehrere Software-Ausführungen.*

8.2 *Der Angreifer hört ab/spielt ein berechtigtes biometrisches Template aus/in eine oder mehrere(n) Software-Komponten.*

9. Bedrohungen im Bereich der Verbindungen:

9.1 *Der Angreifer verändert, modifiziert, überbrückt oder deaktiviert eine oder mehrere Verbindungen zwischen Komponenten.*

9.2 *Der Angreifer hört ab/spielt ein berechtigtes biometrisches Muster oder Template wenn es zwischen den Einheiten oder Komponenten übertragen wird.*

Gegenmassnahmen für: 7./8./9. Bedrohungen im Bereich der Hardware/Software-Komponenten und der Verbindungen

Um Bedrohung 7./8./9.1 zu umgehen sollten alle Hardware/Software-Komponenten und Verbindungen 'tamper-proof' sein, d.h. sie sollten Einbrüche oder Veränderungen erkennen und ggf. melden.
Desweiteren verhindern die gegenseitige Authentifikation und die Verschlüsselung der Daten mit einem Sessionkey, dass ein Überbrücken oder Deaktivieren einer

oder mehrerer Hardware/Software-Komponten oder Verbindungen nicht bemerkt wird.

Wie bereits bei 7./8./9.1 dargestellt, gilt auch bei Bedrohung 7./8./9.2, dass eine gegenseitige Authentifikation ein Einspielen eines berechtigten Templates verhindert und dessen Verschlüsselung ein Abhören für den Angreifer uninteressant macht.

Folgerung

Wie dieses Kapitel zeigt, liefern biometrische Systeme eine Vielzahl von Angriffsmöglichkeiten, die für PIN-basierte System nicht existieren. Auch deshalb können PIN-basierte Systeme nicht einfach durch biometrische Systeme ersetzt werden. Es ist notwendig für biometrische Systeme neue Protokolle zu entwerfen, um die Sicherheit des Authentifikations-Vorganges zu gewährleiten.
Im nächsten Kapitel werden nun Sicherheitsziele definiert und der Versuch unternommen für zwei Einsatzmöglichkeiten biometrischer Verifikations-Systeme, Protokolle zu entwickeln, die die Sicherheitsziele erreichen.

Kapitel 3

Entwicklung eines Protokolls für TOC-Systeme

Nachdem im vorangegangenen Kapitel die Vielzahl an möglichen Bedrohungen für TOC-Systeme dargestellt wurde, wird in diesem Kapitel ein Protokoll für TOC-Systeme entwickelt.
Dazu werden zuerst die Sicherheitsziele, die ein TOC-System erreichen soll, aufgeführt. Anschließend werden zwei verschiedene TOC-Systeme dargestellt und die notwendigen Massnahmen beschrieben, um die genannten Sicherheitsziele zu erreichen.

3.1 Sicherheitsziele

Aus den im vorigen Kapitel dargestellten Bedrohungen für TOC-System, lassen sich folgende Sicherheitsziele für solche Systeme ableiten:

1. **Keine unberechtigten Benutzer**
   Das System soll nur berechtigten Benutzer akzeptieren, d.h über eine hohe Mechanismenstärke verfügen.
   (Mit einer gültigen Smartcard soll sich nur der rechtmäßige Besitzer der Smartcard authentifizieren können; die FAR sollte also im Idealfall 0 sein.)

2. **Lebenderkennung**
   Der Fingerprint-Scanner soll nur echte Finger akzeptieren.
   (Er muss über eine ausreichende Lebenderkennung verfügen, um die Verwendung von gefälschten Fingern oder latenten Fingerabdrücken zu erkennen.)

3. **Gesichertes Referenztemplate**
   Das Referenztemplate soll nicht gelöscht oder verändert werden können.

4. **Kein Austausch von Einheiten**
   Es soll nicht möglich eine einzelne Einheiten des Systems unbemerkt auszutauschen.
   (Zum Beispiel soll die Installation eines falschen Fingerprint-Scanners nicht möglich sein.)

5. **Keine direkte Einspielung von Daten**
   Es sollen keine Daten direkt ins System eingespielt werden können.

   (a) *Ein abgehörter Fingerabdruck soll nicht am Fingerprint-Scanner vorbei direkt ins System eingespielt werden können.*

   (b) *Ein abgehörtes Referenztemplate soll nicht am Smartcard-Reader vorbei direkt in den Host eingespielt werden können.*

6. **Sichere Datenübertragung**
   Ein Abhören der übertragenen Daten soll unmöglich bzw. uninteressant sein.
   (Aus den übertragenen Daten soll sich für den Angreifer kein Template rekonstruieren lassen.)

7. **Nur zugelassene Smartcards**
   Das System soll gefälschte Smartcards erkennen und zurückweisen.

8. **Ausgabe nur an zugelassene Hosts**
   Die Smartcard soll ihr Referenztemplate erst nach Authentifikation des Hosts ausgeben.

9. **Datenlöschung nach Vergleich**
   Nach dem Vergleich sollen die Templates in der Vergleichseinheit gelöscht bzw. überschrieben werden bevor das Ergebnis des Matching-Algorithmus ausgegeben wird.

Die Sicherheitsziele lassen sich in drei Gruppen einteilen:

- Sicherheitsziele, die man nur durch einen geeigneten Matching-Algorithmus bzw. eine geeignete Einstellung der FAR/FRR erreichen kann (1.).

- Sicherheitsziele, die man nur durch bauliche Massnahmen erreichen kann (2.+3.).

- Sicherheitsziele, die man durch ein 'sicheres' Protokoll erreichen kann (4.-9.).

3.2 Massnahmen

Bevor man sich nun Gedanken macht, mit welchen Massnahmen man die oben genannten Sicherheitsziele erfüllen kann, muss man sich überlegen, wie das System 'physikalisch' aufgebaut sein soll. Denn einige der genannten Ziele kann man schon durch das Integrieren mehrerer Einheiten in eine 'einbruchssichere' Box erreichen.
Sicherheitsziel 1: (Keine unberechtigten Benutzer) hängt vom jeweiligen Algorithmus ab bzw. speziell von der FAR.
Die Common Criteria definieren in [1] eine *strength of function* in Abhängigkeit von der FAR folgendermassen:

Strength of Function Level	Maximum FAR
SOF-Basic	0.01 (1 von 100)
SOF-Medium	0.0001 (1 von 10000)
SOF-High	0.000001 (1 von 1000000)

Wie jedoch in 1.1.3 bereits erläutert wurde hängen die Einstellungen von FAR und FRR voneinander ab und müssen deshalb in Abhängigkeit vom jeweiligen Einsatzgebiet des Systems, d.h. vom Anspruch an Komfort und Sicherheit, gewählt werden.
Da dies ein Problem des Algorithmus und nicht des Protokolls ist wird **Sicherheitsziel 1: (Keine unberechtigten Benutzer)** im Folgenden nicht weiter betrachtet.

Der physikalische Aufbau hängt stark damit zusammen wo das System eingesetzt werden soll. Zwei Szenarien sind dabei interessant:

- PC-Login (Zugang)
- Türschloss (Zutritt)

Im Folgenden werden diese beiden konkreten Systeme, die nach der Definition aus [7] TOC-Systeme sind (siehe auch 1.2.1), dargestellt und die jeweiligen Authentifikations-Protokolle erstellt.
Die dabei verwendeten Abkürzungen sind in Tabelle 3.1 zusammengefasst.

3.2.1 PC-Login-System

Bei einem PC-Login-System gehen wir von einem 'offenen' System aus, wie es z.B. in einem öffentlichen Rechenzentrum steht. Wie Abb.3.1 zeigt handelt es sich dabei um ein System bei dem Smartcard-Reader, Fingerprint-Scanner und Host einzelne Bauteile sind. Smartcard-Reader und Fingerprint-Scanner sind jeweils über eine Leitung mit dem Host verbunden. Der Host beinhaltet die Matching- und die Extraktions-Einheit.

Bezeichnung	Bedeutung
H	Host
S, SMC	Smartcard
F, FPS	Fingerprintscanner
RND^X, RND	Zufallszahl
SF	Scramble Function
MD	aktueller Fingerabdruck
SD	Scrambled Data
X_p	öffentlicher Schlüssel des Teilnehmer X
X_s	geheimer Schlüssel des Teilnehmer X
Z^X	Zertifikat des Teilnehmer X
RT	Referenztemplate
$\|\|$	Konkatenation
SF(N,M)	Anwendung von SF mit Zufallszahl N auf Nachricht M
S(M)	Nachricht M verschlüsselt mit Schlüssel S
TTP	Vertrauenswürdige Instanz (Trusted Third Party)
ATM	Automatic Teller Machine z.B. ein Geldautomat
Token	Nachricht, in der mehrere Teilnachrichten zusammengefasst sind

Tabelle 3.1: Verwendete Bezeichnungen

Dieses System ist durch die Zugänglichkeit aller Komponenten und Leitungen für viele Bedrohungen offen. Deswegen sind mehrere Massnahmen notwendig um die genannten Sicherheitsziele zu erreichen.

Will man **Sicherheitsziel 2 (Lebenderkennung)** in einem solchen System erreichen, muss man einen Fingerprint-Scanner wählen, der mehr als nur den Fingerabdruck erfasst. Verschiedene technische Möglichkeiten für Fingerprint-Scanner sind in [16] dargestellt.
Ein Fingerprint-Scanner der im Moment gegen alle Angriffsversuche per Fälschung sicher zu sein scheint ist der holographische Fingerprint-Scanner von Optel [13].

Sicherheitsziel 3 (Gesichertes Referenztemplate) erreicht man durch die Speicherung des Referenztemplates auf einer Smartcard, welche man so konfiguriert, dass nur durch den Administrator das Template aus dem Enrollment verändert oder gelöscht werden kann. Dies kann man z.B. durch Schutz des Templates mit einer Administrator-PIN oder ähnlichem erreichen.
Die Sicherheitseigenschaften von Smartcards werden in [9] und [10] genauer untersucht.

Der Fingerprint-Scanner ist die einzige Einheit, deren Austausch sinnvoll ist. Das Austauschen des Smartcard-Readers macht keinen Sinn, da die Smartcard ihre Daten erst nach gegenseitiger Authentifikation mit dem Host und dann in ver-

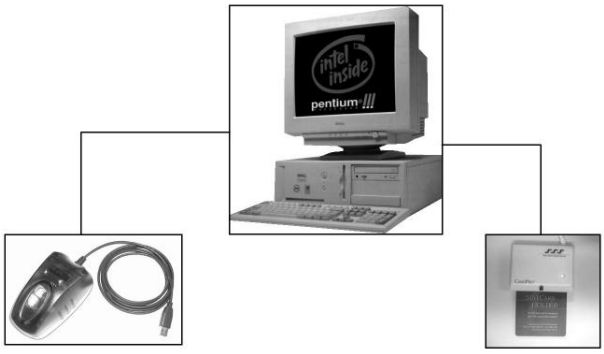

Abbildung 3.1: Beispiel eines PC-Login-Systems

schlüsselter Form ausgibt. Ein Aufzeichnen dieser Daten durch den Smartcard-Reader ist damit gleichbedeutend mit dem Abhören der Kommunikation. Also muss das Protokoll **Sicherheitsziel 6 (Sichere Datenübertragung)** erfüllen. Um **Sicherheitsziel 4 (Kein Austausch von Einheiten)** zu erreichen muss man verhindern, dass der Fingerprint-Scanner unbemerkt ausgetauscht werden kann. Leider können die heutigen Fingerprint-Scanner keine Daten verschlüsseln, wodurch keine gegenseitige Authentifikation zwischen Host und Fingerprint-Scanner ausgeführt werden kann. Somit gibt es keine Möglichkeit den Austausch über das Protokoll zu bemerken, weshalb man **Sicherheitsziel 4 (Kein Austausch von Einheiten)** nur erreichen kann, indem man das System z.B. durch eine Kamera 'überwacht'. Eine weitere Möglichkeit wäre eine Liste bekannter MAC-Adressen im Host und das Abfragen und Prüfen der MAC-Adresse des Fingerprint-Scanners durch den Host vor der Kommunikation.

Will man **Sicherheitsziel 5(a) (Kein direktes Einspielen eines Fingerabdrucks)** erreichen und somit das Einspielen abgehörter Daten am Fingerprint-Scanner vorbei verhindern, so muss man den Übertragungsmodus des Fingerprint-Scanners betrachten.
Es ist davon auszugehen, dass der Fingerabdruck im Klartext vom Fingerprint-Scanner zum Host gesendet wird, da der Fingerprint-Scanner nicht verschlüsseln kann. Damit ist der Fingerabdruck leicht abzuhören und kann später wieder unbemerkt eingespielt werden.
Scheuermann/Schwiderski-Grosche/Struif bieten in [16] neben Verschlüsselung und Klartext einen weiteren Übertragungsmodus an. Diesen bezeichnen sie als *Schutz der Daten durch eine 'Scramble Function'*. Ziel ist es das spätere Einspielen des abgehörten Fingerabdrucks zu verhindern. Abb.3.2 zeigt den Ablauf des in Tabelle 3.2 beschriebenen Protokolls.

Um Replay-Angriffe zu verhindern verwendet der Host jedesmal eine andere Zufallszahl *RND*, wodurch sich jedesmal ein anderes *SD* ergibt.

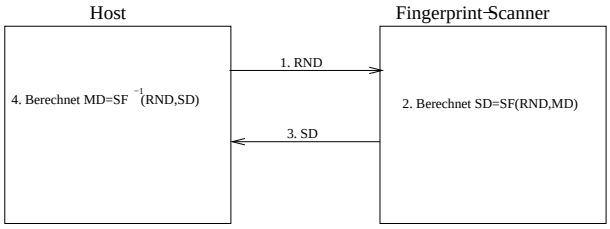

Abbildung 3.2: Berechnung und Übertragung von 'scrambled data' [16]

Scrambled-Data-Protokoll
1. Der Host *H* schickt eine Zufallszahl *RND* an den Fingerprint-Scanner *F*. 2. Der Fingerprint-Scanner *F* berechnet anhand der 'Scramble Function' *SF* aus der Zufallszahl *RND* und dem Fingerabdruck *MD* die gewürfelten Daten *SD*. 3. Der Fingerprint-Scanner *F* schickt *SD* an den Host *H*. 4. Der Host *H* rechnet aus *SD* mit der 'Scramble Function' *SF* den Fingerabdruck *MD* zurück.

Tabelle 3.2: Protokoll zu Abb.3.2

Voraussetzung dafür ist jedoch, dass die 'Scramble Function' geheim bleibt, da ein Angreifer sonst folgende Angriffsmöglichkeit hat:

1. Angreifer hört eine 'korrekte' Kommunikation zwischen Host und Fingerprint-Scanner ab und erhält so *RND* und *SD*.

2. Angreifer berechnet mit der 'Scramble Function' *SF* aus SD wieder MD.

3. Angreifer beschafft sich die zugehörige gültige Smartcard.

4. Angreifer beginnt Anmeldung am System und fängt die aktuelle Zufallszahl RND, die der Host an den Fingerprint-Scanner schickt, ab. Dann berechnet er mit dem abgehörten MD, dem aktuellen RND und der 'Scramble Funktion' *SF* das aktuelle SD und schickt es an den Host zurück.

5. Angreifer wird als berechtigter Benutzer authentifiziert.

Es ist jedoch anzumerken, dass die heutigen Fingerprint-Scanner weder über eine 'Scramble Function' noch über irgendeine Art von Verschlüsselungsfunktion verfügen. Ein Bekannt-werden des aktuellen Fingerabdrucks wird als unkritisch gesehen, da dieser auch im Alltag vielfältig zugänglich ist.

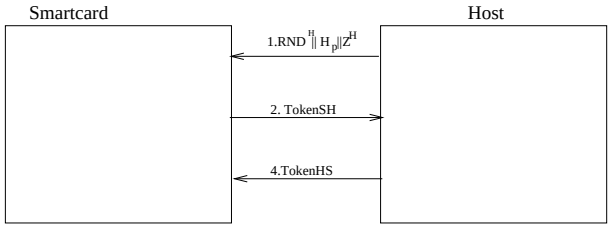

Abbildung 3.3: Three pass authentication

Voraussetzung:
1. Als zugrunde liegende Verschlüsselung wird eine asymmetrische Verschlüsslung angenommen. 2. Sei (R_p, R_s) das Schlüsselpaar der TTP. 3. Der Host verfügt über ein Schlüsselpaar (H_p, H_s), R_p und ein $Z^H = R_s(H_p)$. 4. Die Smartcard verfügt über ein Schlüsselpaar (S_p, S_s), R_p und ein $Z^S = R_s(S_p)$.

Tabelle 3.3: Voraussetzungen für das *Authentifkations-Protokoll*

Sicherheitsziel 5(b) (Kein direktes Einspielen eines Referenztemplates) ist einfacher zu erreichen, da die Smartcard Daten verschlüsseln kann.
Will man das direkte Einspielen von abgehörten (verschlüsselten) Referenztemplates verhindern, so führt man eine gegenseitige Authentifikation zwischen Host und Smartcard ein, vor deren Erfolg weder der Host Daten akzeptiert noch die Smartcard Daten versendet.
Abb.3.3 zeigt eine *Mutual three pass authentication*, die sich an der der ISO/IEC in [17] orientiert. Das zugehörige Protokoll realisiert zum einen die gegenseitige Authentifikation zwischen Smartcard und Host, zum anderen vereinbaren die Teilnehmer dabei einen gemeinsamen symmetrischen Sessionkey.
Dem zugehörigen Protokoll wird der Name *Authentifikations-Protokoll* gegeben. Die für dieses Protokoll notwendigen Voraussetzungen sind in Tabelle 3.3 zusammengefasst, das Protokoll selbst steht in Tabelle 3.4.

Aus dem Protokoll in Tabelle 3.4 ergibt sich, dass sich die Smartcard nach Schritt 3 gegenüber dem Host authentifiziert hat und der Host gegenüber der Smartcard nach Schritt 5. Außerdem wurde RND^S als Sessionkey vereinbart.

Mit den bei **Sicherheitsziel 5 (Keine direkte Einspielung von Daten)** dargestellten Massnahmen erfüllt man auch **Sicherheitsziel 6 (Sichere Datenübertragung)**. Denn weder aus den Daten, die der Fingerprint-Scanner ver-

Authentifikations-Protokoll

1. Der Host H erzeugt eine Zufallszahl RND^H und schickt diese, zusammen mit Z^H und H_p an die Smartcard S.
2. Die Smartcard S prüft den übertragenen öffentlichen Schlüssel H_p mit Hilfe von Z^H durch
$$H_p = R_p(Z^H),$$
erzeugt eine Zufallszahl RND^S, erzeugt einen Token
$$TokenSH = (Z^S \| S_p \| RN^S),$$
wobei
$$RN^S = S_s(H_p(RND^S) \| RND^H)$$
und schickt diesen an den Host H.
3. Der Host H prüft den im Token $TokenSH$ enthaltenen öffentlichen Schlüssel der Smartcard S durch
$$S_p = R_p(Z^S),$$
entschlüsselt damit RN^S und prüft er der Adressat ist und ob RND^H die von ihm erzeugt Zufallsszahl ist.
4. Der Host H erzeugt $TokenHS = (RN^H)$, wobei
$$RN^H = H_s(S_p(RND^S) \| RND^H)$$
ist und schickt diesen an die Smartcard S.
5. Die Smartcard S entschlüsselt mit H_p die Nachricht RN^H
$$H_p(RN^H) = (S_p(RND^S) \| RND^H)$$
und prüft, ob sie der Adressat ist, ob RND^H die von ihr in Schritt 1 erhaltene Zufallszahl ist und ob RND^S die in Schritt 2 von ihr erzeugte Zufallszahl ist.

Tabelle 3.4: Protokoll zu Abb.3.3

sendet, noch aus den Daten, die die Smartcard versendet, läßt sich ein Fingerabdruck bzw. das Referenztemplate gewinnen. Damit sind diese Daten uninteressant für einen Angreifer.

Desweiteren erfüllt die gegenseitige Authentifikation zwischen Smartcard und Host auch **Sicherheitsziel 7 (Keine gefälschten Smartcards)**, da eine ge-

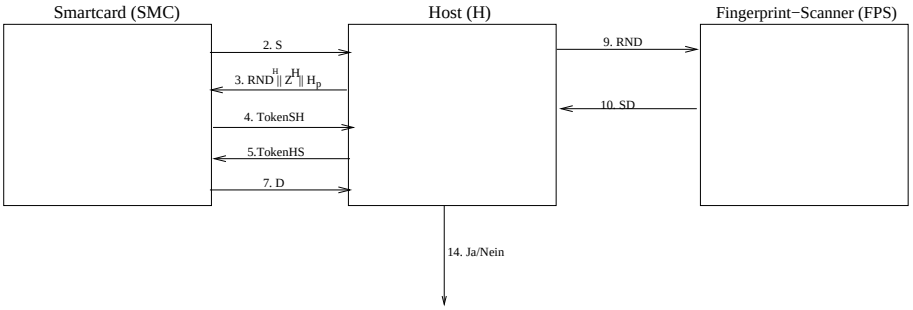

Abbildung 3.4: Benutzer-Verifikation an einem PC-Login-System

fälschte Smartcard nicht das, für die Authentifikation notwendige Z^S bzw. R_s kennt und so gegenüber dem Host seine Identität nicht nachweisen kann. Damit wird sie als gefälscht erkannt und zurückgewiesen.

Sicherheitsziel 8 (Ausgabezeitpunkt des Referenztemplates) betrifft nur die Reihenfolge der Schritte und muss in der Abfolge der Protokollschritte beachtet werden. **Sicherheitsziel 9 (Datenlöschung nach Vergleich)** stellt lediglich einen weiteren wichtigen Protokollschritt dar, der das spätere Auslesen des Referenztemplates aus dem Host verhindern soll.
Aus den bei den einzelnen Sicherheitszielen genannten Massnahmen ergibt sich für das **PC-Login-System** das folgende *Benutzer-Verifikation-Protokoll*. Dies ist in Abb.3.4 dargestellt und wird in Tabelle 3.5 erläutert. Wie das *Benutzer-Verifikations-Protokoll* als reines Kommunikations-Protokoll geschrieben aussieht zeigt Tabelle 3.6.

Das Kommunikations-Protokoll betrachtet nur die Kommunikation zwischen den Teilnehmern. Die fehlenden Schritte sind interne Schritte der jeweiligen Teilnehmer. Diese werden durch solch ein Protokoll nicht erfasst, müssen aber bei der Analyse sehr wohl betrachtet werden.

3.2.2 Raum-Zutritts-System

Im Gegensatz zum 'offenen' PC-Login-System kann man das in Abb.3.5 dargestellte Zutritts-System als 'geschlossen' ansehen, da sich alle Einheiten in einer einbruchssicheren Box befinden. Somit hat ein Angreifer keinen Zugang zu den Leitungen zwischen den Einheiten und kann nur versuchen, das System über Eingaben am Fingerprint-Scanner bzw. Smartcard-Reader zu überlisten.
Damit erreicht dieses System einige der angestrebten Sicherheitsziele schon durch seinen physikalischen Aufbau.
Sicherheitsziele, die durch den physikalischen Aufbau, d.h. das Zusammenfassen der Einheiten in einer einbruchssicheren Box, des Zutritts-Systems erreicht werden sind:

Benutzer-Verifikations-Protokoll für ein PC-Login-System
1. Der Benutzer steckt seine Smartcard in den Smartcard-Reader.
2. Die Smartcard schickt dem Host ein Startsignal, z.B. ihre Identität S.
3. Das *Authentifikations-Protokoll* wird ausgeführt.
4. Bei Erfolg der gegenseitigen Authentifikation verwendet die Smartcard RND^S als symmetrischen Schlüssel und verschlüsselt das Referenztemplate RT zu $D = RND^S(RT)$ und schickt dies an den Host.
5. Der Host entschlüsselt D zu RT.
6. Das *Scrambled-Data-Protokoll* wird ausgeführt.
7. Der Host vergleicht MD mit RT und berechnet den Übereinstimmungsgrad.
8. Anschließend überschreibt der Host MD und RT im Speicher.
9. Der Host überprüft, ob der Übereinstimmungsgrad oberhalb des Schwellwertes liegt und entscheidet damit, ob die Verifikation erfolgreich war und teilt dem Benutzer über den Bildschirm mit, dass er verifiziert bzw. abgelehnt wurde.

Tabelle 3.5: Protokoll zu Abb.3.4

	Kommunikations-Protokoll	
2.	$SMC \longrightarrow Host:$	S
3.	$SMC \longleftarrow Host:$	$RND^H \| R_s(H_p) \| H_p$
4.	$SMC \longrightarrow Host:$	$R_s(S_p) \| S_p \| S_s(H_p[RND^S] \| RND^H)$
5.	$SMC \longleftarrow Host:$	$H_s(S_p[RND^S] \| RND^H)$
7.	$SMC \longrightarrow Host:$	$RND^S(RT)$
8.	$Host \longrightarrow FPS:$	RND
9.	$Host \longleftarrow FPS:$	$SF(RND, MD)$
14.	$Host \longrightarrow Ausgabe:$	$Ja/Nein$

Tabelle 3.6: *Benutzer-Verifikations-Protokoll* aus Tabelle 3.5 als reines Kommunikations-Protokoll

- **Sicherheitsziel 5 (Keine direkte Einspielung von Daten)**:
  Ein Einspielen am Fingerprint-Scanner bzw. Smartcard-Reader vorbei ist nicht möglich.

Abbildung 3.5: Beispiel eines biometrischen Systems als Türschloss

- **Sicherheitsziel 6 (Sichere Datenübertragung)**:
  Die Datenübertragung kann nicht mehr abgehört werden.

Damit ist die verschlüsselte Übertragung der Daten zwischen den Einheiten nicht mehr nötig. Dadurch vereinfacht sich das Benutzer-Verifikations-Protokoll etwas.

Auf den ersten Blick erscheint es, als würde auch **Sicherheitsziel 4 (Kein Austausch von Einheiten)** durch diese bauliche Massnahme erreicht, denn ein unbemerkter Austausch einzelner Einheiten des Systems ist nicht mehr möglich.
Hachez/Koeune/Quisquater weisen in [11] jedoch darauf hin, dass nun ein sogenannter 'false ATM'-Angriff möglich ist. Bei diesem wird die ganze Box ausgetauscht und darauf gewartet, dass ein Benutzer sich authentifizieren will. Die falsche Box nimmt den Fingerabdruck auf, speichert diesen und geht dann in einen Fehlerzustand.
Um solch einen Angriff zu verhindern kann man wiederum die Smartcard und die in 3.2.1 bei **Sicherheitsziel 5(b) (Kein direktes Einspielen eines Referenztemplates)** angeführte *mutual three pass authentication* verwenden. Diese führt dazu, dass sich der Benutzer, bevor er seine biometrischen Daten präsentiert, davon überzeugen kann, dass das gesamte System funktioniert.
Jedoch ergibt sich daraus ein weiteres Problem: *Woher weiß der Benutzer das sich der Host gegenüber der Smartcard erfolgreich authentifiziert hat?*
Hachez/Koeune/Quisquater schlagen deshalb eine weitere kleine Einheit mit ei-

ner LED (Light Emitting Diode) als graphische Ausgabe vor, die der Benutzer bei sich trägt und der er, genauso wie seiner Smartcard, vertrauen kann. In diese Einheit schiebt der Benutzer seine Smartcard und verbindet sie mit dem System. Auf diese Weise kann die Smartcard dem Benutzer anzeigen, ob die Authentifikation zwischen Smartcard und Host erfolgreich war.
Diese Vorgehensweise ist laut [11] bereits Gegenstand von FINREAD [18].

Jedoch stellt sich nun die Frage: *Wie groß ist der Schaden, wenn der Benutzer auf einen solchen Angriff hereinfällt?*
Alles worüber sich ein Angreifer damit Besitz verschafft ist der aktuelle Fingerabdruck des Benutzers, da die Smartcard wie beschrieben das Referenztemplate erst nach erfolgreicher gegenseitiger Authentifikation ausgibt.
Beim heutigen Stand der Technik ist es weder möglich aus einem 'bloßen' Fingerabdruck ein 'korrektes' Template zu erzeugen, um dieses direkt in den Host einzuspielen, noch kann man daraus einen falschen Finger bauen, der einen passenden Fingerabdruck liefert um den Matching-Algorithmus zu überlisten.
Und auch wenn die Technik irgendwann soweit sein sollte, dann braucht man keine falsche ATM um in den Besitz des Fingerabdrücks des gewünschten Benutzers zu kommen, sondern nur ein Glas oder ähnliches.

Um

- **Sicherheitsziel 2 (Lebenderkennung)**
- **Sicherheitsziel 3 (Gesichertes Referenztemplate)**
- **Sicherheitsziel 7 (Nur zugelassene Smartcards)**
- **Sicherheitsziel 8 (Ausgabe nur an zugelassene Hosts)**
- **Sicherheitsziel 9 (Datenlöschung nach Vergleich)**

bei diesem Zutritts-System zu erfüllen, können dieselben Massnahmen ergriffen werden wie beim PC-Login-System in 3.2.1.

Damit ergibt sich für das Zutritts-System das in Tabelle 3.7 dargestellte *Benutzer-Verifikations-Protokoll.*

Tabelle 3.8 zeigt das zugehörige Kommunikations-Protokoll.

	Benutzer-Verifikations-Protokoll für das Raum-Zutritts-System
1.	Der Benutzer steckt seine Smartcard in den Smartcard-Reader.
2.	Die Smartcard schickt ein Startsignal, z.B. ihre Identität S, an den Host, also den Türöffner.
3.	Das *Authentifikations-Protokoll* wird ausgeführt.
4.	Bei Erfolg der gegenseitigen Authentifikation verwendet die Smartcard RND^S als symmetrischen Schlüssel und verschlüsselt das Referenztemplate RT zu $D = RND^S(RT)$ und schickt dies an den Host.
5.	Der Host entschlüsselt D zu RT.
6.	Der Host fordert vom Fingerprint-Scanner den aktuellen Fingerabdruck.
7.	Der Fingerprint-Scanner sendet den Fingerabdruck MD an den Host.
8.	Der Host vergleicht MD mit RT und berechnet den Übereinstimmungsgrad.
9.	Anschließend überschreibt der Host MD und RT im Speicher.
10.	Der Host überprüft, ob der Übereinstimmungsgrad oberhalb des Schwellwertes liegt und entscheidet damit, ob die Verifikation erfolgreich war und öffnet im positiven Fall die Tür.

Tabelle 3.7: Benutzer-Verifikations-Protokoll für das Raum-Zutritts-System

		Kommunikations-Protokoll
2.	$SMC \longrightarrow Host:$	S
3.	$SMC \longleftarrow Host:$	$RND^H \| R_s(H_p) \| H_p$
4.	$SMC \longrightarrow Host:$	$R_s(S_p) \| S_p \| S_s(H_p[RND^S] \| RND^H)$
5.	$SMC \longleftarrow Host:$	$H_s(S_p[RND^S] \| RND^H)$
7.	$SMC \longrightarrow Host:$	$RND^S(RT)$
9.	$Host \longleftarrow FPS:$	MD
13.	$Host \longrightarrow Ausgabe:$	$Ja/Nein$

Tabelle 3.8: *Benutzer-Verifikations-Protokoll* aus Tabelle 3.7 als reines Kommunikations-Protokoll

Folgerung

In diesem Kapitel wurden Sicherheitsziele für TOC-Systeme definiert. Anschließend wurden die Massnahmen angegeben, durch welche die Sicherheitsziele erreicht werden und für zwei spezielle TOC-Systeme jeweils ein Authentifikations-Protokoll erstellt.
Was zu analysieren bleibt ist, ob die Protokolle die Sicherheitsziele wirklich erreichen. Dazu wird im nächsten Kapitel eine informelle Analyse des PC-Login-Protokolls durchgeführt.

Kapitel 4

Informeller Nachweis der Sicherheitseigenschaften

Bevor im nächsten Kapitel formale Methoden für die Analyse der Korrektheit und Sicherheit kryptographischer Protokolle eingeführt werden, wird in diesem Abschnitt ein informeller Nachweis dieser Eigenschaften geliefert.

4.1 Vorüberlegung

1. Das PC-Login-Protokoll
   Wie aus der Darstellung hervorgeht ist das PC-Login-Protokoll im Grunde aus zwei Teilprotokollen zusammengesetzt.
   Das Protokoll für die gegenseitige Authentifikation zwischen Smartcard und Host ist stark an das entsprechende Protokoll der *ISO/IEC* [17] angelehnt, welches als sicher und korrekt angesehen werden kann.
   Die 'Scramble Function' als Übertragungs-Modus für den aktuellen Fingerabdruck ist sicher solange die Funktion geheim ist. Die Korrektheit des dazugehörigen Protokolls ist leicht ersichtlich, da es nur aus zwei Berechnungen und zwei Übertragungen besteht.
   Zu zeigen bleibt, dass weder die Übergänge zwischen den Teilprotokollen noch das Ende des Protokolls die Sicherheit und Korrektheit des Protokolls verändern.

   (a) Zwischen den beiden Teilprotokollen passiert folgendes:

   - In Schritt 4 des *Benutzer-Verifikations-Protokolls* verschlüsselt die Smartcard ihr Referenztemplate RT mit dem vereinbarten symmetrischen Schlüssel RND^S zu D und schickt dieses an den Host.
   - in Schritt 5 desselben Protokolls entschlüsselt der Host D zu RT.

   Die Sicherheit dieser Aktionen hängt von der Qualität des gewählten symmetrischen Verschlüsselungsverfahrens ab und von der Qualiät der

von der Smartcard erzeugten und als Schlüssel verwendeten Zufallszahl RND^S.
Das verwendete symmetrische Verschlüsselungsverfahren muss sich zudem auf einer Smartcard realisieren lassen, genauso wie die Generierung der Zufallszahl. Sichere symmetrische Verschlüsselungsverfahren sind z.B. Triple-DES oder AES [19]. Dass die Generierung von Zufallszahlen auf Smartcards möglich ist zeigt [20].

(b) Am Ende des *Benutzer-Verifikations-Protokolls* befindet sich:

- In Schritt 7 der Vergleich von Referenz- und aktuellem Template.
- In Schritt 8 das Überschreiben der Daten.
- In Schritt 9 die Ausgabe des Ergebnisses des Matching-Algorithmus.

Der Matching-Algorithmus hat keine Auswirkung auf die Sicherheit des Protokolls.
Durch das Überschreiben der Daten **vor** der Ausgabe des Ergebnisses des Matching-Algorithmus wird verhindert, dass ein potenzieller Angreifer das System nach Ausgabe des Ergebnisses, aber vor Löschung der Vergleichsdaten zum Absturz bringt und so in den Besitz eines passenden (Template,Referenztemplate)-Paares kommt.
Die Ausgabe des Ergebnisses am Bildschirm dient lediglich dazu den Benutzer zu informieren und hat somit ebenfalls auf die Sicherheit des Protokolls keinen Einfluß.

2. Das Zutritts-System
Da in diesem Protokoll die Datenübertragung in einer sicheren Umgebung stattfindet ist keine Verschlüsselung nötig. Damit entfallen hier die Sicherheitsüberlegungen aus 1(a).
Die unter 1(b) betrachteten Schritte sind jedoch auch in diesem Protokoll vorhanden, sodass sich die Sicherheitsüberlegungen übertragen lassen.
Wie das PC-Login-Protokoll, so enthält auch das Protokoll des Zutritts-Systems das Teilprotokoll zur gegenseitigen Authentifikation zwischen Smartcard und Host-System. Jedoch fehlt die 'Scramble Function', weil, wie oben erwähnt, die Verbindungen der Einheiten und damit die Datenübertragungen durch bauliche Massnahmen geschützt sind.
Da bei den Schritten, die in diesem Protokoll anstelle der 'Scramble Function' stehen lediglich Daten im Klartext innerhalb einer sicheren Umgebung übertragen werden, muss die Protokoll-Sicherheit nicht weiter betrachtet werden.

Trotz dieser Überlegungen soll das PC-Login-Protokoll im Folgenden noch einmal, an Hand der Vorschläge von *Abadi und Needham* für den Entwurf korrekter kryptographischer Protokolle [21], analysiert werden.

4.2 Ansatz von Abadi und Needham

Als Grundlage wird die Arbeit *Prudent Engineering Practice for Cryptographic Protocols* von *Abadi und Needham* [21] verwendet. Darin geben die Autoren Hinweise für die 'fehlerfreie Entwicklung kryptographischer Protokolle' und zeigen bekannte Fehler aus existierenden Protokollen auf.
Für den Nachweis der Sicherheit und Korrektheit wird gezeigt, dass die in [21] angegebenen Prinzipien von dem aufgestellten Protokoll erfüllt werden.
Dazu werden nun die in der Arbeit aufgestellten Prinzipien genannt und anschließend argumentiert, warum das erstellte Protokoll diese beachtet.

4.2.1 Grundlagen

Abadi/Needham betrachten in [21] zuerst zwei Prinzipien als elementaren Voraussetzungen, die ein Protokoll erfüllen muss um verständlich zu sein, dadurch Fehler zu vermeiden und damit Sicherheit zu gewährleisten.

Prinzip 1 (Explizite Kommunikation)

> *Jede Nachricht sollte sagen, was sie bedeutet: die Interpretation einer Nachricht sollte nur von ihrem Inhalt abhängen. Es sollte möglich sein einen klaren deutschen Satz aufzuschreiben, der den Inhalt beschreibt - auch wenn es einen passenden Formalismus gibt.*

Die Erstellung der Protokolle in 3.2 erfolgte sprachlich und wurde anschließend aus der sprachlichen Darstellung abgeleitet.

Prinzip 2 (Passende Aktion)

> *Die Bedingungen für eine Nachricht auf Grund derer eine Handlung ausgeführt wird, sollten klar dargelegt werden. Dadurch kann jeder, der den Entwurf auf Verwendbarkeit prüft sehen. ob diese akzeptabel sind oder nicht.*

Bei der Erstellung des Protokolls wurden folgende Voraussetzungen festgelegt:

- physikalischer Aufbau des Systems.
- Art der Verschlüsselung und Sicherheit der Schlüssel.
- Eignung des Matching-Algorithmus.
- Fähigkeiten der einzelnen Einheiten bzgl. Verschlüsselung.

Bezogen auf die Nachrichten wurde in den Protokoll-Vorraussetzungen (siehe Tabelle 3.3) explizit festgelegt über welche Schlüssel die einzelnen Teilnehmer verfügen. Damit wird auch festgelegt unter welchen Bedingungen die Teilnehmer die jeweiligen Nachrichten erstellen können.

Für das *Authentifikations-Protokoll* gilt:

- In Schritt 1 schickt der Host $(RND^H \| Z^H \| H_p)$ an die Smartcard. Er kann diese Nachricht nur erstellen sofern er Kenntnis über die in der Nachricht enthaltenen Teilnachrichten hat. Diese Bedingung wird durch die Voraussetzungen in Tabelle 3.3 erfüllt bzw. durch die Fähigkeit des Hosts eine Zufallszahl zu generieren.

- In Schritt 2 schickt die Smartcard $(Z^S \| S_p \| RN^S)$ an den Host, wobei $RN^S = S_s(H_p(RND^S) \| RND^H \| H)$. Die ersten beiden Teilnachrichten besitzt die Smartcard durch die Voraussetzung. RN^S kann sie generieren, sofern sie:

  - S_s kennt. (Gilt durch die Voraussetzung.)
  - RND^S erstellen kann. (Wurde in 4.1.1.(a) bzw. [20] gezeigt.)
  - H_p kennt. (Dies setzt wiederum voraus, dass sie R_p kennt, um Z^H zu entschlüsseln und mit H_p aus der vorigen Nachricht zu vergleichen. Nach Voraussetzung kennt die Smartcard R_p.)
  - RND^H kennt. (Dieses wurde der Smartcard im vorigen Schritt geschickt.)

  Damit sind alle Bedingungen für die Nachricht in Schritt 2 gegeben.

- In Schritt 4 schickt der Host RN^H an die Smartcard, wobei $RN^H = H_s(S_p(RND^S) \| RND^H \| S)$.

  - Der Host kennt H_p nach Voraussetzung.
  - S_p erhält der Host auf diesselbe Weise wie die Smartcard H_p im vorigen Schritt.
  - RND^S erhält der Host durch $S_p(RN^S) = (H_p(RND^S) \| RND^H \| H)$ und anschließend $H_s(H_p(RND^S)) = RND^S$.

  Damit sind auch alle Bedingungen für Schritt 4 gegeben.

Damit sind die Bedingungen für die Nachrichten im *Authentifkations-Protokoll* betrachtet.

Für das *Scrambled-Data-Protokoll* gilt:

- In Schritt 1 schickt der Host die Zufallszahl RND an den Fingerprintscanner. (Da er diese selber erzeugen kann ist die Bedingung für diese Nachricht erfüllt.)
- In Schritt 3 schickt der Fingerprint-Scanner $SD = SF(RND, MD)$ an den Host. Voraussetzung dafür ist, dass er
  - die 'Scramble Function' SF ausführen kann. (Wird vorausgesetzt.)
  - MD kennt. (Erhält er vom Benutzer.)
  - RND kennt. (Erhält er in Schritt 1 vom Host.)

Damit sind alle Bedingungen für das *Scrambled-Data-Protokoll* gegeben.

Zu betrachten bleiben die Schritt 4 und Schritt 10 des *Benutzer-Verifikations-Protokolls*:

- In Schritt 4 schickt die Smartcard $D = RND^S(RT)$ an den Host. Voraussetzung für diese Nachricht ist, dass der Host und die Smartcard RND^S als gemeinsamen Session-Key vereinbart haben. (Dies geschieht während des *Authentifikations-Protokolls*. RT kommt aus dem Speicher der Smartcard.)
- In Schritt 10 gibt der Host an die Umgebung eine $Ja/Nein$-Nachricht aus. Diese generiert er an Hand des Matching-Algorithmus aus dem Vergleich von MD und RT. (Kenntnis von MD erlangt er im *Scrambled-Data-Protokoll* und Kenntnis von RT im obigen Schritt 4.)

Nun sind alle Nachrichten bzgl. ihrer Bedingungen untersucht und erläutert.

4.2.2 Benennung

Das folgende Prinzip beschäftigt sich damit wann bzw. warum die Nennung des Adressaten in einer Nachricht sinnvoll ist.

Prinzip 3

> *Wenn die Identität eines Teilnehmers wesentlich für die Bedeutung einer Nachricht ist, so ist es erforderlich den Namen des Teilnehmers in der Nachricht explizit zu erwähnen.*

In den aufgestellten Protokollen ist der Host der einzige Teilnehmer, der mit mehreren anderen Teilnehmern kommuniziert. Da jedoch weder die Smartcard noch der Fingerprint-Scanner mit Nachrichten für den jeweils anderen 'umgehen' können, ist es nicht notwendig den Adressaten der Nachricht beizufügen, da die Nachrichten eindeutig zugeordnet werden können.

- Die gegenseitige Authentifikation findet nur zwischen Smartcard und Host-System statt.
- Die verschlüsselte Nachricht an das Host-System kommt von der Smartcard.
- Die 'Scramble Function' betrifft nur Host-System und Fingerprint-Scanner.

Deshalb wurde auf die explizite Nennung des Adressaten in den Nachrichten verzichtet.

4.2.3 Verschlüsselung

In diesem Abschnitt werden die Prinzipien betrachtet, die die sinnvolle und durchdachte Verwendung von Verschlüsselung und Signatur betreffen.

Prinzip 4 (Gebrauch von Verschlüsselung)

> *Sei Dir klar darüber, warum verschlüsselt wird. Verschlüsselung ist nicht ganz billig. Im vorhinein sollte klar definiert werden warum verschlüsselt wird, sonst kann es zu Redundanz führen. Verschlüsselung ist kein Synonym für Geheimhaltung und unpassender Gebrauch kann zu Fehlern führen.*

Dabei weisen die Autoren auf die möglichen Gründe für Verschlüsselung hin. Verschlüsselung kann verwendet werden, um

- die Vertraulichkeit einer Nachricht zu bewahren.
  Nur der gewünschte Empfänger der Nachricht kennt den Schlüssel um den Klartext wiederherzustellen.
- die Authentizität einer Nachricht zu garantieren.
  Nur der wirkliche Absender der Nachricht kennt den Schlüssel, der verwendet wurde um den Klartext zu verschlüsseln.
- Teile einer Nachricht zusammenzubinden, genannt: Binding. (Bem: Dann ist Signatur ausreichend.)
  Die Aussage eines Bindings ist Protokoll-abhängig.
- Zufallszahlen zu generieren.

Im betrachteten Protokoll tritt Verschlüsselung auf bei:

1. der gegenseitigen Authentifikation zwischen Smartcard und Host-System.
2. der Übertragung des Referenztemplates von der Smartcard zum Host-System.

Zu 1.:

- In Schritt 4 des *Kommunikations-Protokolls* für das PC-Login-System (siehe Tabelle 3.6) verschlüsselt die Smartcard $RN^S = S_s(H_p(RND^S)\|RND^H)$ und schickt dies zusammen mit $Z^S = R_s(S_p)$ und S_p an das Host-System. Sinn dieser Verschlüsselung ist:
  - Binden von $H_p(RND^S)$ an RND^H.
  - Geheimhaltung von RND^S als späterem Schlüssel durch Verschlüsselung mit H_p.
  - Nachweis, dass die Nachricht von einer bekannten (gültigen) Smartcard stammt, durch Zuhilfenahme von Z^S und S_p (Authentizität).
- Für die Verschlüsselung durch das Host-System in Schritt 4 gilt dasselbe.

zu 2.:

- in Schritt 7 des *Kommunikations-Protokolls* für das PC-Login-System verschlüsselt die Smartcard ihr Referenztemplate RT mit dem symmetrischen Schlüssel RND^S zu $D = RND^S(RT)$.
  Sinn dieser Verschlüsselung ist:
  - Geheimhaltung des Referenztemplates

Damit wurde gezeigt, dass alle Verschlüsselungen sinnvoll und notwendig sind.

Prinzip 5 (Signieren verschlüsselter Daten)

> *Wenn ein Teilnehmer Daten signiert, die bereits verschlüsselt wurden, so kann nicht abgeleitet werden, dass der Teilnehmer den Inhalt der Nachricht kennt. Andererseits kann man sehr wohl ableiten, dass der Teilnehmer, der die Nachricht signiert und dann zur Geheimhaltung verschlüsselt, den Inhalt der Nachricht kennt.*

Wie man oben sieht muss dieses Prinzip beim vorliegenden Protokoll sehr genau betrachtet werden, da genau dies getan wird:

> Die Smartcard verschlüsselt RND^S mit H_p und signiert diese Daten dann mit anderen Daten durch S_s zu RN^S.

Prinzip 5 sagt nun: Der Host kann, nachdem er RN^S entschlüsselt hat, nicht sicher sein, dass die Smartcard RND^S kennt. Was ist jedoch die Funktion von RND^S in diesem Protokoll?

- Schlüssel für die spätere Übertragung des Referenztemplates an den Host.

Wäre RND^S dem Sender der Nachricht nicht bekannt, verfügt er also nicht über das Schlüsselpaar (S_s, S_p), so kann er die Antwort RN^H des Host nicht entschlüsseln. Dadurch ist er dann auch nicht in der Lage D zu erstellen.
Damit spielt es an dieser Stelle für den Host keine Rolle, ob der Sender von RN^S Kenntnis von RND^S hat.

4.2.4 Pünktlichkeit (Zeitstempel, Sequenznummern und andere Nonces)

Bei den folgenden Prinzipien wird darauf hingewiesen, dass es wichtig ist die Rolle und Bedeutung von Nonces gut zu überdenken und zu überprüfen.

Prinzip 6 (Eigenschaften von Nonces)

> *Sei Dir klar darüber, von welchen Eigenschaften von Nonces Du ausgehst. Was sich zur Sicherstellung der zeitlichen Reihenfolge eignet, muss sich nicht zur Sicherstellung der Zusammengehörigkeit eignen - und unter Umständen wird Zusammengehörigkeit besser durch andere Mittel sichergestellt.*

Diesem Prinzip liegt der Gedanke der Autoren zu Grunde, Nonces dazu zu verwenden ein Protokoll gegen Replay-Attacken zu schützen. *Es wird eine Nachricht versendet, welche zu einer Antwort führt, die nur mit der Kenntnis der ersten erstellt werden konnte. Das Ziel ist es, zu garantieren, dass die zweite Nachricht nach der ersten erstellt wurde und manchmal sie zusammen zu binden [21].*
Genau dies passiert im *Authentifikations-Protokoll* bei der gegenseitigen Authentifikation zwischen Smartcard und Host durch die Verwendung von RND^H und RND^S. RN^S kann in Schritt 2 nur mit Kenntnis von RND^H aus Schritt 1 erstellt werden. Und RN^H in Schritt 4 nur mit Kenntnis von RND^S aus Schritt 2.
Ebenso können die Schritte 4 und 5 aus dem *Benutzer-Verifikations-Protokoll* für das PC-Login-System erst nach gegenseitiger Kenntnis von RND^S ausgeführt werden.

Prinzip 7 (Frische)

> *Der Nutzen einer voraussagbaren Größe (wie z.B. dem eines Zählers) kann sich dazu eignen durch ein Challenge-Respond-Verfahren Neuheit zu garantieren. Aber wenn eine voraussagbare Größe eine Wirkung hat, so sollte sichergestellt werden, dass ein Angreifer nicht in der Lage ist, einen Challenge zu simulieren und später einen Response einzuspielen.*

Alle im PC-Login-Protokoll verwendeten Nonces sind Zufallszahlen, welche in jedem Protokoll-Lauf frisch generiert werden und nur in diesem akzeptiert bzw. verwendet.
Damit sind sie nicht voraussagbar.

Prinzip 8 (Zeitstempel)

> *Wenn Zeitstempel verwendet werden um Frische mit Bezug auf die aktuelle Zeit zu garantieren, so muss der Unterschied zwischen den lokalen Uhren auf verschiedenen Maschinen wesentlich geringer sein als das erlaubte Alter einer Nachricht, die als gültig betrachtet wird.*

Im Protokoll werden keinerlei Zeitstempel verwendet.

Prinzip 9 (Frische: Verwendung vs. Generierung)

> *Ein Schlüssel, der vor kurzem verwendet wurde, z.B. um einen Nonce zu verschlüsseln, kann nun schon ziemlich alt sein und möglicherweise kompromittiert. Kürzliche Verwendung macht einen Schlüssel nicht besser als er sonst ist.*

Dies ist eine Betrachtung, die vor allem symmetrische Schlüssel betrifft. Deshalb diskutieren die Autoren dieses Prinzip auch am Beispiel des Needham-Schroeder-Protokolls [22] und der Frage nach der Frische des ausgetauschten symmetrischen Kommunikationsschlüssels K_{AB}.
Im *Benutzer-Verifikations-Protokoll* für das PC-Login-System werden vor allem asymmetrische Schlüssel verwendet die fest sind und deren Frische deshalb nicht zur Diskussion steht. Einziger symmetrischer Schlüssel ist die Zufallszahl RND^S. Nun stellt sich die Frage: Woher weiß das Host-System, dass RND^S ein frischer Schlüssel ist? Oder: Wie könnte ein Angriff aussehen, der eine nicht-frische RND^S ausnutzt?
Die Smartcard generiert die Zufallszahl RND^S während der gegenseitigen Authentifikation mit dem Host-System. Deshalb müsste ein Angreifer, der sein eigenes Referenztemplate ins System einspielen will, die Smartcard dazu bringen eine ihm bekannte Zufallszahl RND^S zu generieren. Nur dann ist er in der Lage sein mit RND^S verschlüsseltes Referenztemplate einzuspielen. Dies kann auf Grund der Sicherheitseigenschaften von Smartcards (siehe [9] und [10]) als nicht möglich angesehen werden. Außerdem wurde in Kapitel 3 gezeigt, dass das System **Sicherheitsziel 5: (Keine direkte Einspielung von Daten)** erfüllt. Dadurch kann der Angreifer solch ein selbst erstelltes D nicht ins System einspielen.
Damit kann gesagt werden: Das Protokoll garantiert zwar nicht die Frische des symmetrischen Schlüssels RND^S, aber die Sicherheit des Protokolls wird dadurch nicht beeinträchtigt. Es wird vorausgesetzt, dass die Smartcard bei jedem Protokolllauf eine frische Zufallszahl RND^S generiert.

4.2.5 Erkennen von Nachrichten und Verschlüsselungen

Prinzip 10

> *Wenn eine Verschlüsselung verwendet wird, um die Bedeutung der Nachricht zu zeigen, dann sollte es möglich sein zu sagen, welche Verschlüsselung verwendet wurde. Im allgemeinen Fall, in dem die Verschlüsselung protokoll-abhängig ist, soll es möglich sein abzuleiten, dass die Nachricht zu diesem Protokoll, ja sogar zu einem bestimmten Lauf dieses Protokolls gehört und Kenntnis über ihre Nummer im Protokoll zu erlangen.*

Auch dieses Prinzip zielt wieder auf Aktionen mit symmetrischem Schlüssel ab. Als Beispiel wählen die Autoren wieder die letzten beiden Schritte des Needham-Schroeder-Protokolls [22], in denen ein Teilnehmer den Challenge N_b sendet und $N_b + 1$ als Response zurückerhält. Das Problem ist wie der Teilnehmer Challenge und Response unterscheidet und damit ausschließen kann, dass ein Angreifer einfach den Challenge abgefangen und zurückgeschickt hat.
Im PC-Login-Protokoll gibt es kein Challenge-Response-Vorgang mit symmetrischem Schlüssel.
RND^S dient lediglich zur Geheimhaltung des Referenztemplates bei der Übertragung. Außerdem lässt sich jede Nachricht im Protokoll eindeutig einem Protokoll-Schritt zuordnen. Deshalb betrifft Prinzip 10 dieses Protokoll nicht.

4.2.6 Glauben

Prinzip 11

> *Der Designer des Protokolls sollte wissen auf welche Glaubensbeziehungen sein Protokoll angewiesen ist und warum die Abhängigkeiten notwendig sind. Die Gründe warum bestimmte Glaubensbeziehungen akzeptabel sind sollten explizit dargelegt werden.*

Folgende Glaubensbeziehungen wurden vorausgesetzt:

- Sowohl die symmetrische (z.B. AES,Triple-DES) als auch die asymmetrische Verschlüsselung (z.B. RSA) ist ausreichend sicher.
  Dies kann auf Grund einer Vielzahl an Untersuchungen angenommen werden.
- Die Smartcard wählt in jedem Durchlauf eine frische Zufallszahl RND^S als symmetrischen Schlüssel.
- Das Protokoll erfüllt die in 3.1 aufgestellten Sicherheitsziele
- Die 'Scramble Function' ist geheim und wird ausgetauscht, sobald sie kompromittiert wurde.

Folgerung

Die Argumente aus 3.2 und die in diesem Abschnitt angeführten Nachweise legen die Vermutung nahe, dass das Protokoll sicher ist. Bewiesen ist dies jedoch nicht. Um die Sicherheit und Korrektheit eines solchen Protokolls besser zu analysieren ist es wünschenswert formale Methoden zu verwenden. Aus diesem Grund werden im nächsten Kapitel formale Methoden eingeführt und im letzten Kapitel dann zwei Methoden, die sich für eine Analyse des Protokolls eignen, genauer dargestellt.

Kapitel 5

Formale Methoden

Das 'Needham-Schroeder-Authentifikations-Protokoll' [22] galt als ein sicheres Authentifikations-Protokoll in verteilten Systemen. Im Zuge dessen wurden weitere Protokolle, wie z.B. Kerberos [23], entwickelt und auch verwendet.
Es dauerte jedoch nicht allzulange bis ein Fehler in diesem Protokoll entdeckt wurde [24], den Needham/Schroeder in einer neuen Version des Protokolls dann behoben [25].

Die Erkenntnis, dass solch ein allgemein als sicher angenommenes Protokoll, einen raffinierten Fehler enthalten kann, warf die Forderung nach formalen Methoden für die Analyse von Authentifikations-Protokollen auf. Bis heute ist der Fehler im 'Needham-Schroeder-Protokoll' [22] das meist gewählte Beispiel, um die Funktionalität einer Analyse-Technik nachzuweisen, indem man zeigt, dass sie diesen Fehler aufdeckt.

5.1 Analyse-Methoden

Meadows definiert in [26] Methodentypen für die Analyse von kryptographischen Protokollen:

- **Typ 1:**

  Modellieren und verifizieren des Protokolls an Hand von Spezifikations-Sprachen und Verifikations-Werkzeugen, welche nicht speziell für die Analyse kryptographischer Protokolle entwickelt wurden.

- **Typ 2:**

  Entwicklung von Experten-Systemen, die ein Protokoll-Designer verwenden kann um verschiedene Szenarien zu entwerfen und untersuchen.

- **Typ 3:**

  Modellieren der Anforderungen einer Protokoll-Familie an Hand von Logiken, die für die Analyse von Wissen&Glauben entwickelt wurden.

- **Typ 4:**

  Entwickeln eines formalen Modells, welches auf den algebraischen Termersetzungseigenschaften von kryptographischen Systemen beruht.

Rubin/Honeyman geben in [27] eine Übersicht über die jeweils gängigsten Methoden aus den verschiedenen Typen. Sie merken auch an, dass *die Methoden des Typ 1 die am wenigsten beliebten sind, wogegen die Methoden des Typ 3 die am meisten verbreiteten sind. In allen Fällen sind die Methoden unabhängig von den zugrunde liegenden kryptographischen Mechanismen.*

In den folgenden Abschnitten werden die vier Methodentypen etwas genauer beschrieben.

5.1.1 Methodentyp 1

Die Methoden des Typ 1 modellieren und verifizieren das Protokoll an Hand von Spezifikations-Sprachen und Verifikations-Werkzeugen, welche nicht speziell für die Analyse kryptographischer Protokolle entwickelt wurden. Man versucht dabei die Korrektheit des kryptographischen Protokolls auf dieselbe Weise zu zeigen, wie die Korrektheit von Programmen. Das Problem dabei liegt darin, dass dadurch die Korrektheit, aber nicht unbedingt die Sicherheit der Protokolle gezeigt wird [28].
Um die Korrektheit eines kryptographischen Protokolls mit diesen Methoden zeigen zu können, muss man das Protokoll zuerst in die jeweilige Spezifikation überführen, um dann die Techniken anwenden zu können. Dazu gibt es verschiedene Möglichkeiten. *Sidhu* schlägt in [28] eine Spezifikations-Technik vor, in der das Protokoll als ein gerichteter Graph angesehen wird. *Varadharajan* verwendet dagegen in [29] LOTOS um Authentifikations-Protokolle zu spezifizieren. *Kemmerer* versucht in [30] maschinen-gestützte Verifikationstechniken anzuwenden. Dazu beschreibt er die Eigenschaften, die das Protokoll erfüllen soll als Zustands-Invarianten. Die Theoreme, die bewiesen werden müssen, um zu zeigen, dass das Protokoll die Invarianten erfüllt, sollen dann automatisch vom Verifikations-System generiert werden.

5.1.2 Methodentyp 2

Die Methoden des Typ 2 entwickeln Experten-Systeme, die ein Protokoll-Designer verwenden kann, um verschiedene Szenarien zu entwerfen und untersuchen. Dabei definieren diese Systeme einen unerwünschten Zustand und überprüfen anschließend, ob dieser Zustand vom Anfangszustand aus erreichbar ist.
Das Problem bei diesem Methodentyp ist, dass damit nur sichergestellt werden kann, dass ein Protokoll verschiedene bekannte Fehler nicht enthält, da diese als unerwünschte Zustände definiert werden können. Unbekannte Fehler findet man mit diesem Methodentyp jedoch nicht.
Longley/Rigby zeigen in [31] auf, was Experten-Systeme bieten:

- eine neue Perspektive auf ein Authentifikations-System,
- eine Technik um Modelle zu bauen, die ständig verfeinert werden können,
- eine Methode für die Interaktion zwischen dem Modell, welches eine bessere Einsicht in die Operationen des Systems bietet,
- ein Modell das auf 'was, wenn'-Fragen antwortet und
- eine Methode um die Effektivität von vorgeschlagenen Systemmodifikationen zu testen.

Rubin/Honeyman [27] stellen fest, dass *Experten-Systeme in Verbindung mit anderen Analyse-Techniken, wie z.B. solchen des Typ 3 und 4, verwendet werden können, diese jedoch nie ersetzen werden können.*

Vertreter des Methodentyp 2 sind u.a.:

- das Longley-Rigby-Tool [31] (von Longley/Rigby)
- der Interrogator [33] (von Millen/Clark/Freedman)

5.1.3 Methodentyp 3

Die Methoden des Typ 3 modellieren die Anforderungen einer Protokoll-Familie an Hand von Logiken, die für die Analyse von Wissen&Glauben entwickelt wurden. Die BAN-Logik [34] ist wohl die bekannteste der Methoden dieses Typs und war der Auslöser für eine ausgiebige Erforschung dieses Methodentyps.
Andere Methoden dieses Typs sind

- die GNY-Logik von *Gong/Needham/Yahalom* [35] als eine Erweiterung der BAN-Logik
- die CKT5-Logik von *Bieber* [36] und deren Erweiterung durch *Snekkens* [37]

- die Axiome bzgl. Vertrauen&Glauben von *Rangan* [38]
- die Logik von *Syverson* [39]
- die Logik von *Kailar et. al* [41] und
- die Logik von *Moser* [42]

Gligor et al. beschreiben in [43] explizit den Horizont dieser Logiken:

> *Das primäre Ziel der Authentifikations-Logiken, nämlich zu verifizieren, ob ein Protokoll gegenseitige Authentifikation zwischen zwei Teilnehmern bietet und frische Schlüssel von bekannter Qualität verteilt, hilft den Bereich dieser Logiken sehr präzise einzuschränken. Jede Verwendung dieser Logiken jenseits dieses Punktes kann gefährlich sein und sollte deshalb sehr genau geprüft werden.*

5.1.4 Methodentyp 4

Die Methoden des Typ 4 entwickeln ein formales Modell, welches auf den algebraischen Termersetzungseigenschaften von kryptographischen Systemen beruht. Diese Methoden schließen die Analyse der Erreichbarkeit bestimmter Systemzustände ein, wodurch sie den Methoden des Typ 2 ähneln. Während die Methoden des Typ 2 von einem unsicheren Zustand ausgehen und zeigen, dass es dorthin keinen Pfad gibt, versuchen die Methoden des Typ 4 jedoch zu zeigen, dass ein unsicherer Zustand nicht erreicht werden kann.
Die ersten, die diese Methode verwendet haben waren *Dolev/Yao* [44]. Anschließend wurde sie u.a. von *Merritt* [45], *Syverson* [39, 32] und *Meadows* [46, 47, 48, 26] weiterverfolgt.
Beispiele für Methoden dieses Typs sind

- das Modell nach *Dolev und Yao* [44],
- der NARROWER Algorithmus zur Protokoll-Analyse [46],
- die KPL-Logik nach *Syverson* [39, 40] und
- der NRL-Protocol-Analyser von *Syverson und Meadows* [32]

Der NRL-Protocol-Analyzer scheint auf den ersten Blick eigentlich zum Typ 2 zu gehören, wird aber, da er auf dem Modell von Dolev&Yao basiert, dem Typ 4 zugeordnet.

Folgerung

Die meisten der genannten Methoden wurden speziell für die Analyse eines bestimmten Protokolls entwickelt. Damit ist die Anwendung dieser Methoden auf 'neue' Protokolle schwierig.
Das nächste Kapitel verwendet deshalb für die Analyse des PC-Login-Protokolls *Model − checking* und argumentiert warum man dies als fünften Methodentyp ansehen kann.

Kapitel 6

Model-checking

In diesem Abschnitt werden zwei formale Methoden auf ihre Verwendbarkeit für den Beweis der Sicherheit des PC-Login-Protokolls untersucht. Beide Ansätze betreiben Model-checking und beinhalten Tools, die die Analyse des Protokolls automatisch ausführen.
Diese Tools arbeiten beide mit einer erschöpfenden Suche auf dem Zustandsraum. Dabei suchen sie nach einer Spur, die in einen unsicheren Zustand führt. Dadurch verfolgen sie eine Vorgehensweise, wie man sie von den Methoden des Typ 4 kennt.
Model-checking beruht jedoch nicht auf algebraischen Termersetzungseigenschaften von kryptographischen Systemen. Außerdem wurden die beiden Methoden und Tools speziell für die Analyse kryptographischer Protokolle entwickelt. Deshalb bietet es sich an Model-checking als einen fünften formalen Methodentyp zur Analyse kryptographischer Protokolle zu definieren.

6.1 Definition: Model-checking

H. Schlingloff definiert Model-checking in [49] folgendermassen:

> *Model-checking ist eine automatisierte Technik um Korrektheitseigenschaften von sicherheitskritischen reaktiven Systemen zu verifizieren.*

Unter einem reaktiven System versteht er dabei ein System, das aus mehreren Komponenten besteht die miteinander und mit der Systemumgebung interagieren können.
Um solch eine Verifikation auszuführen benötigt man:

- eine Modellsprache mit der man das System beschreiben kann,
- einen Spezifikationssprache mit der man die Eigenschaften formulieren kann und

- ein Folgerungs-Kalkül um den Verifikationsprozess auszuführen.

Laut [49] wird das System gewöhnlich als ein Zustands-Übergangs-Graph modelliert und die Eigenschaften mittels temporaler Logik formuliert. Anschließend wird eine effiziente Suche verwendet um zu prüfen, ob der Zustands-Übergangs-Graph die temporalen Formeln erfüllt oder nicht.

6.2 Model-checking (Clarke/Marrero/Jha)

Clarke/Marrero/Jha untersuchen in [50, 51] den Ansatz die Sicherheit von kyptographischen Protokollen an Hand von Model-checking zu untersuchen.

6.2.1 Einführung

Der Model-checker, den dieser Ansatz verwendet, besteht aus zwei orthogonalen Komponenten:

- einer *Zustands-Untersuchungs-Komponente* und
- einer *Nachrichten-Ableitungs-Maschine*

In der *Zustands-Untersuchungs-Komponente* wird jeder vertrauenswürdige Teilnehmer durch eine Folge von Aktionen beschrieben, die er während eines Protokoll-Laufes ausführt. Er kann als eine Maschine mit endlich vielen Zuständen gesehen werden. Da das Protokoll einer asynchronen Komposition der Zustandsautomaten entspricht, stellt die 'Spur' der ausgeführten Aktionen eine mögliche Ausführung des Protokolls durch die Teilnehmer dar. Um zu prüfen, ob verschiedene Sicherheitseigenschaften verletzt werden, kann man nun eine erschöpfende Suche auf dem Zustandsraum der Komposition ausführen.
Die *Nachrichten-Ableitungs-Maschine* modelliert die erlaubten Aktionen des Angreifers. *Sie kann als ein einfacher natürlicher Folgerungs-Theorem-Prüfer zur Erstellung gültiger Nachrichten gesehen werden [51].* Dabei werden die auf Grund von bekannten Nachrichten ausführbaren Aktionen durch eine Menge von Schlußfolgerungs-Regeln beschrieben. Da jede Aktion umkehrbar ist, enthält sie sowohl eine Einführungs- als auch eine Eliminationsregel. Wie man in Tabelle 6.1 sieht reduzieren Elimininationsregeln die Länge einer Nachricht und Einführungsregeln verlängern diese. Die Eigenschaft, dass jede Aktion sowohl Eliminiations- als auch Einführungsregeln enthält, garantiert die Existenz einer 'normalisierten Ableitung'. Bei einer 'normalisierten Ableitung' stehen im Ableitungsbaum alle Eliminationsregeln oberhalb der Einführungsregeln. Es lässt sich zeigen, dass es

Einführungsregel		Elimininationsregel	
$\cdot I$	$\frac{m_1\ m_2}{m_1 \cdot m_2}$	$\cdot E_l$	$\frac{m_1 \cdot m_2}{m_1}$
		$\cdot E_r$	$\frac{m_1 \cdot m_2}{m_2}$
$\{\}_k I$	$\frac{m\ k}{\{m\}_k}$	$\{\}_k E$	$\frac{\{m\}_k\ k^{-1}}{m}$

Tabelle 6.1: Einführungs- und Eliminationsregeln

zu jeder Ableitung eine solche 'normalisierte Ableitung' gibt und diese eine beschränkte Länge hat. (Der interessierte Leser findet den Beweis dieser Aussagen in [51] (Kapitel 7) eine genauere Erläuterung.) Die Existenz einer 'normalisierten Ableitung' ermöglicht einen effizienten Algorithmus um zu entscheiden, ob eine Nachricht gültig ist oder nicht.
Der Angreifer kann eine Nachricht abfangen, umleiten und mittels Verschlüsselung, Entschlüsselung, Konkatenation und Projektion neue Nachrichten erstellen. Außerdem kann er jede gesendete Nachricht abfangen und sie seiner Wissensmenge hinzufügen. Aus dieser Wissensmenge kann er dann versuchen neue Nachrichten abzuleiten. Jede Nachricht, die er an einen vertrauenswürdigen Teilnehmer schickt, muss durch die *Nachrichten-Ableitungs-Maschine* erzeugt worden sein.
Die Trennung der Funktionalität führt zu einem sehr intuitiven Berechungsmodell. Zum einen muss, im Gegensatz zu Termersetzungs-Systemen, keine Menge von Termersetzungs-Regeln erstellt werden, um zu modellieren, wie der Angreifer andere Teilnehmer dazu bringen kann neue Nachrichten zu generieren. Zum anderen müssen, im Gegensatz zu Methoden die lediglich auf der Suche im Zustandsraum basieren, die Fähigkeiten des Angreifers nicht als Zustandsmaschine codiert werden. Die Fähigkeiten des Angreifers werden direkt in den Model-checker selbst integriert.
Durch das Vorhandensein einer *Nachrichten-Ableitungs-Maschine* muss man nicht vorher festlegen welche Nachrichten bzw. welchen Nachrichtentyp das Modell annimmt. Dem Angreifer ist es erlaubt, beim Versuch einen vertrauenswürdigen Teilnehmer zu täuschen, eine beliebige Nachricht zu erzeugen.

6.2.2 Modellansatz

Der Model-checker geht von einer 'perfekten Verschlüsselung' aus. Dies bedeutet laut [51]:

- *Um den Klartext aus einem chiffrierten Text zu erhalten, muss man den Entschlüsselungs-Schlüssel kennen.*

- *Das Kryptosystem enthält genug Redundanz, so dass man einen chiffrierten Text nur durch Verschlüsselung mit dem zugehörigen Schlüssel erzeugen kann. Dies impliziert, dass es keine Verschlüsselungs-Kollisionen gibt. Wenn zwei chiffrierte Texte gleich sind, dann müssen sie aus demselben Klartext durch Verschlüsselung mit demselben Schlüssel entstanden sein.*

Durch diese Annahme kann man die Sicherheitsuntersuchung auf das Protokoll beschränken.
Das Modell ist intuitiv, da es die grundlegende Idee von Nachrichtenerzeugung und Kommunikation einfängt.
Dies geschieht folgendermassen:

- Jede Rolle im Protokoll wird durch eine Folge von einfachen Kommandos wie SEND,RECEIVE und NEWSECRET beschrieben. Diese beschreiben wie die Teilnehmer während eines Protokoll-Laufes interagieren.

- Die asynchrone Komposition der Aktions-Folgen der einzelnen Teilnehmer liefert das Gesamt-Modell des Protokolls.

- Der Angreifer ist ein unspezifizierter Teilnehmer. Dieser fängt jede Nachricht ab und schickt sie dann weiter bzw. leitet sie um. Der Angreifer kann zudem Nachrichten verschicken, um vorzugeben ein vertrauter Teilnehmer zu sein.

- Ein Protokoll-Lauf besteht aus einigen aufeinanderfolgenden Aktionen von einer Menge von Teilnehmern und dem Angreifer.

- Die 'Spur' ist die Folge von einem oder mehreren Protokoll-Läufen. Die 'Spur' wird analysiert um herauszufinden, ob die Sicherheit des Protokolls kompromittiert wurde. D.h. ob der Angreifer irgendwann ein Geheimnis gelernt hat oder ein Teilnehmer A glaubt einen Protokoll-Lauf mit Teilnehmer B beendet zu haben, während Teilnehmer B nicht am Protokoll-Lauf teilgenommen hat.

- Im allgemeinen kann eine Menge von Sicherheits-Anforderungen an Hand einer Logik spezifiziert werden und dann die 'Spur' daraufhin untersucht werden, ob die Anforderungen verletzt werden.

6.2.3 Untersuchte Eigenschaften

Zwei Arten von Eigenschaften sollen untersucht werden:

- Sicherheits-Eigenschaften

- Temporale Eigenschaften (Korrespondenz)

Für die Untersuchung der Sicherheits-Eigenschaften wird dem Model-Checker eine Menge von Termen gegeben, die der Angreifer nicht erfahren darf. Während der Verifikation ist dann zu prüfen, ob der Angreifer nicht an diese Geheimnisse gelangt.
Die temporalen Eigenschaften führen zur Korrespondenz-Relation von *Woo und Lam* [52]. Die Korrespondenz-Relation $X \hookrightarrow Y$ ist erfüllt, wenn auf jede X-Aktion eine Y-Aktion folgt und es eine 1:1-Abbildung von X-Aktionen auf Y-Aktionen gibt.
Um diese Eigenschaft zu prüfen wird der globale Zustand um einen Zähler erweitert. Jede Korrespondenz-Eigenschaft $X \hookrightarrow Y$ bekommt einen eigenen Zähler, welcher die Differenz der X- und Y-Aktionen festhält. Wird dieser negativ (mehr X- als Y-Aktionen), dann ist die Korrespondenz-Eigenschaft an dieser Stelle verletzt. Wird er nie negativ, so gibt es immer eine 1:1-Abbildung von X- auf Y-Aktionen.

6.2.4 Erstellung von Nachrichten

Die Nachrichten eines Protokoll-Laufs erhält man durch Konkatenation und Verschlüsselung kleiner Teilnachrichten.
Die kleinsten Teilnachrichten nennt man 'atomare Nachrichten'. Davon gibt es vier Arten:

- *Schlüssel* werden dazu verwendet Nachrichten zu verschlüsseln.
- *Teilnehmer-Namen* werden verwendet um auf die Teilnehmer eines Protokolls zu verweisen.
- *Nonces* sind Zufallszahlen. Da sie zufällig erzeugt werden geht man davon aus, dass eine Nachricht, die eine Nonce enthält, erstellt wurde, nachdem die Nonce erstellt wurde.
- *Daten*, welche keine Rolle dabei spielen wie das Protokoll funktioniert, aber zwischen Teilnehmern übertragen werden sollen.

Die Autoren definieren:
Sei $\mathcal{A}$ der Raum der atomaren Nachrichten. Dann ist die Menge aller Nachrichten $\mathcal{M}$ über eine Menge von atomaren Nachrichten $\mathcal{A}$ folgendermassen induktiv definiert:

- Sei $a \in \mathcal{A}$, dann ist $a \in \mathcal{M}$
- Sei $m_1 \in \mathcal{M}$ und $m_2 \in \mathcal{M}$, dann gilt $m_1 \cdot m_2 \in \mathcal{M}$
- Sei $m \in \mathcal{M}$ und Schlüssel $k \in \mathcal{A}$, dann ist $\{m\}_k \in \mathcal{M}$

Die folgenden Regeln geben an, wie eine neue Nachricht aus bereits bekannten Nachrichten erzeugt werden kann. Dazu wird definiert wie eine Nachricht aus einer initialen Menge von Information $\mathcal{I}$ abgeleitet werden kann.

1. Sei $m \in \mathcal{I}$, dann gilt $\mathcal{I} \vdash m$
2. Sei $\mathcal{I} \vdash m_1$ und $\mathcal{I} \vdash m_2$, dann gilt $\mathcal{I} \vdash m_1 \cdot m_2$ (Konkatenation)
3. Sei $\mathcal{I} \vdash m_1 \cdot m_2$, dann gilt $\mathcal{I} \vdash m_1$ und $\mathcal{I} \vdash m_2$ (Projektion)
4. Sei $\mathcal{I} \vdash m$ und $\mathcal{I} \vdash k$ für einen Schlüssel k, dann gilt $\mathcal{I} \vdash \{m\}_k$ (Verschlüsselung)
5. Sei $\mathcal{I} \vdash \{m\}_k$ und $\mathcal{I} \vdash k^{-1}$, dann gilt $\mathcal{I} \vdash m$ (Entschlüsselung)

Dabei definiert $\vdash$ die Ableitungsfunktion. Ob eine gegebene Nachricht abgeleitet werden kann ist entscheidbar.
(Die Argumentation hierfür läßt sich wieder in [51] Kapitel 7 nachlesen.)

6.2.5 Definition des Modells

Um das Modell nun formal zu definieren muss man beschreiben wie der globale Gesamt-Zustand und die lokalen Zustände der einzelnen Teilnehmer definiert sind und wie die Aktionen die Zustände verändern.

- Das Modell besteht aus asynchroner Komposition einer Menge von benannten, kommunizerenden Prozessen, welche die vertrauenswürdigen Teilnehmer und den Angreifer modellieren.
- Der Zustand eines vertrauenswürdigen Teilnehmers wird durch die Belegung seiner lokalen Variablen und durch seinen Zähler definiert. Im Ausgangszustand ist die Belegung der Variablen leer.
- Jeder Protokoll-Lauf eines vertrauenswürdigen Teilnehmers wird als einer dieser Prozesse modelliert und durch die Folge von Aktionen, die er ausführt, beschrieben.
- Der Zustand des Angreifers wird durch die Menge, der ihm bekannten Nachrichten festgelegt. Deshalb wird der Angreifer durch die Spur der empfangenen Nachrichten und die Verwendung der *Nachrichten-Ableitungs-Maschine* modelliert. Die *Nachrichten-Ableitungs-Maschine* beschreibt wie der Angreifer neue Nachrichten erzeugen kann.

Formal wird jeder Teilnehmer als ein Triple $< N, p, B >$ definiert, mit:

- $N \in names$ ist ein Teilnehmername.

- p ist ein Prozess, der als Sequenz von auszuführenden Aktionen gegeben ist.
- $B : vars(N) \rightarrow \mathcal{M}$ ist eine Menge von Belegungen für $vars(N)$.
  $vars(N)$ ist dabei die Menge der Variablen, die Teilnehmer N verwendet.

Der Angreifer dagegen wird durch ein Paar $< Z, I >$ modelliert, mit:

- $Z \in names$ ist der Name des Angreifers.
- $I \subseteq \mathcal{M}$ ist die Menge der Nachrichten, die dem Angreifer aus dem Grundzustand oder per Abhören bekannt sind.

Der globale Zustand ist ein Triple $< \Pi, C, S >$, mit:

- Π ist das Produkt der verschiedenen Teilnehmer- und des Angreifer-Prozesses.
- C ist eine Menge von Zählern, einer für jede Korrespondenz-Relation. Für jede Relation $X_i \hookrightarrow Y_i$ gibt es einen Zähler $C_i \in IN$. Dessen Wert ist die Differenz zwischen der Anzahl Y_i- und der Anzahl X_i-Prozesse, die bisher aufgetreten sind.
- $S \subseteq \mathcal{M}$ ist die Menge von Nachrichten, die als geheim angesehen werden. Diese Worte darf der Angreifer nicht erfahren. Dazu gehören z.B. die privaten Schlüssel der anderen Teilnehmer.

Jeder Teilnehmer kann zwei Arten von Aktionen ausführen:

- *interne Aktionen* und
- *Kommunikations-Aktionen*

Interne Aktionen werden asynchron ausgeführt und jeder Teilnehmer darf interne Aktionen ausführen. Man definiert eine Übergangs-Relation $\rightarrow$ zwischen Teilnehmern:

$$A \rightarrow B \quad \Leftrightarrow \quad \text{Teilnehmer } A \text{ kann eine Aktion ausführen und zu einem Teilnehmer werden, der sich wie } B \text{ verhält.}$$

Kommunikations-Aktionen bestehen aus Sende- und Empfangs-Aktionen. Sie können als synchrone Aktion zwischen dem Angreifer und dem Teilnehmer, der sie ausführt, gesehen werden. Eine Empfangs-Aktion kann als Nachricht vom Angreifer an den Teilnehmer gesehen werden. Sie kann die Belegungen des vertrauenswürdigen Teilnehmers vergrößern. Eine Sende-Aktion kann als Nachricht vom Teilnehmer an den Angreifer gesehen werden und kann so das Wissen des Angreifers vergrößern.
Damit passiert bei einer Sende-Aktion folgendes:

- der Angreifer speichert die neue Nachricht in seinem lokalen Speicher.
- Der Teilnehmer führt folgenden Übergang aus:

$$< A, SEND(s - msg).q', B_A > \rightarrow < A, q', B_A >$$

Soll nun eine Empfangs-Aktion ausgeführt werden, muss die Nachricht des Angreifers mit der empfangenen Nachricht übereinstimmen. Eine Nachricht $s - msg \in \mathcal{I}$ eines Angreifers stimmt mit einer Nachricht $r - msg$ eines Teilnehmers $B =< B, q, B_B >$ überein, wenn es eine Ersetzungsfunktion $\sigma_B : vars(B) \rightarrow \mathcal{M}$ gibt, die B_B ($B_B \subseteq \sigma_B$) erweitert, so dass $s - msg = \sigma_B(r - msg)$. Stimmen die Nachrichten überein, so kann folgender Übergang ausgeführt werden:

$$< B, RECEIVE(r - msg).q', B_B > \rightarrow < B, q', \sigma'_B >$$

Dabei ist σ'_B die kleinste Ersetzungsfunktion, die die obigen Bedingungen erfüllt. *Interne Aktionen* werden meist verwendet um neue Informationen zu erzeugen. So erzeugt $NEWSECRET$ eine Nonce oder einen Sitzungs-Schlüssel und belegt mit ihm eine Variable. Diese neue Belegung wird zur Menge der Belegungen des Teilnehmers hinzugefügt:

$$< A, NEWSECRET(var).p', B > \rightarrow < A, p', B' >$$

Dabei gilt $B' = B[var \leftarrow val]$, falls val ein Wert ist der bei der Aktion neu generiert wird.
Desweiteren gibt es die Aktionen $BEGINIT$, $ENDINIT$, $BEGRESPOND$ und $ENDRESPOND$. Diese verändern den Zustand eines Teilnehmers nicht und markieren nur den Beginn bzw. das Ende seiner Teilnahme am Protokoll.
Sie dienen dazu die Korrespondenz-Relationen, wie z.B. $A.ENDINIT(B) \hookrightarrow B.BEGRESPOND(A)$, zu überprüfen. Ist die Relation erfüllt, so weiß man, dass wenn A das Protokoll beendet, B am Protokoll mit A teilgenommen hat.
Die Aktionen, die ein Teilnehmer ausführen kann sind durch die Folge von Aktionen p beschränkt, die seine Rolle im Protokoll repräsentieren. Der Angreifer dagegen ist unbeschränkt und kann jede Aktion zu jeder Zeit ausführen, wobei er nur solche Nachrichten verschicken kann, die er aus seinem Wissen ableiten kann. Um den Zustandraum endlich zu halten, legt man o.B.d.A. fest, dass der Angreifer nur endlich oft $NEWSECRET$ ausführen darf. Die Autoren von [51] argumentieren, dass dies keine Beschränkung ist, da in allen untersuchten Protokolle keiner der Teilnehmer einen Nonce oder Schlüssel auf Frische prüft und deshalb der Angreifer nie eine $NEWSECRET$-Aktion ausführt.

6.2.6 Such-Algorithmus

Interessant sind alle in einem Protokoll möglichen Spuren. Alles im Modell ist endlich, bis auf die Menge der vom Angreifer erzeugbaren Nachrichten. Wenn

man zeigen kann, dass der Angreifer nie eine unendliche Menge von Nachrichten erzeugt, dann ist das gesamte Modell endlich. Damit kann man eine Tiefensuche ausführen um alle möglichen Spuren des Modells und damit die Verletzung der Sicherheits-Eigenschaften zu prüfen.
Die Menge der vom Angreifer erzeugten Nachrichten ist endlich, da die Menge der Nachrichten nur eine Rolle spielt, wenn geprüft werden muss, ob sich aus dem Wissen des Angreifers:

- ein Geheimnis oder
- eine Nachricht auf die ein vertrauenswürdiger Teilnehmer wartet

ableiten läßt.
Im ersten Fall hat man eine endliche Menge von Nachrichten zu prüfen. Dies ist entscheidbar (siehe [51] Kapitel 7).
Der zweite Fall schließt das Prüfen der Übereinstimmung mit einem Muster ein. Im Normalfall ist das Muster restriktiv genug, um die Ableitung auf eine endliche Anzahl von Übereinstimmungen zu beschränken (da es nur eine endliche Anzahl atomarer Nachrichten gibt). Dies muss nicht immer der Fall sein. Daher beschränken die Autoren von [51] die Länge der Ableitungen, die verwendet werden können um Nachrichten zu finden, welche mit einer Variable, während eines Matching-Prozesses übereinstimmen. Die Autoren weisen weiter darauf hin, dass sie in der Praxis bisher mit einer Ableitungslänge von 1 ausgekommen sind.
Somit ist die Menge der Nachrichten endlich und damit auch der Zustandsraum und man kann eine Tiefensuche verwenden, um eine erschöpfende Suche auszuführen.

6.2.7 Informations-Algorithmen

Da es einen effizienten Algorithmus für $\mathcal{I} \vdash m$ gibt kann man zwei Algorithmen angeben, die folgende Fragen beantworten:

1. Wie wird die Menge der bekannten Informationen des Angreifers erweitert, wenn dieser eine neue Nachricht lernt?
2. Wie wird nach einer Ableitung für m aus $\mathcal{I}^*$ gesucht, die nur Einführungs-Regeln verwendet? ($\mathcal{I}^*$ ist der Abschluß der Anfangsmenge der Annahmen $\mathcal{I}$ unter allen Eliminationsregeln)

Folgender Algorithmus beantwortet die 1.Frage:

funct $add(\mathcal{I}, m)$

foreach $i \in \mathcal{I}$ **do**

if $i = \{x\}_y \wedge m = y^{-1}$

then $\mathcal{I} := add(\mathcal{I}, x)$
if $y \in \mathcal{I}$ **then** $\mathcal{I} := \mathcal{I} - i$ **fi**
fi
od

if $m \in \mathcal{A}$
then return $\mathcal{I} \cup \{m\}$
elseif $m = x \cdot y$
then return $add(add(\mathcal{I}, x), y)$
elseif $m = \{x\}_y \wedge y^{-1} \in \mathcal{I}$
then if $y \in \mathcal{I}$
then return $add(\mathcal{I}, x)$
else return $add(\mathcal{I} \cup \{m\}, x)$
fi
else
return $\mathcal{I} \cup \{m\}$
fi

Jede neue Nachricht die der Angreifer erhält, wird zu der Menge der Nachrichten die er bereits hat hinzugefügt. Die Menge wird anschliessend unter den Eliminationsregeln abgeschlossen und 'redundante' Nachrichten werden entfernt.

Die Suche nach der Ableitung von m aus $\mathcal{I}^*$ schließt die Prüfung $m \in \mathcal{I}^*$ ein. Wenn diese fehlschlägt, dann wird rekursiv nach den Komponenten von m gesucht.
Der nächste Algorithmus realisiert dies:

funct $in(\mathcal{I}, m)$

if $m \in \mathcal{I}$
then return $true$
elseif $m = x \cdot y$
then return $in(\mathcal{I}, x) \wedge in(\mathcal{I}, y)$
elseif $m = \{x\}_y$
then return $in(\mathcal{I}, x) \wedge in(\mathcal{I}, y)$
else
return $false$
fi

Damit ist nun der Model-Checker ausreichend beschrieben und der Blick kann wieder auf das zu untersuchende Protokoll gerichtet werden.

6.2.8 Umsetzung des PC-Login-Protokolls (Marrero)

Nun soll das in Tabelle 3.5 dargestellte Benutzer-Verifikations-Protokoll für den PC-Login für den Model-checker spezifiziert werden.
Das Kommunikations-Protokoll ist (siehe auch Tabelle 3.6):

1. $SMC \longrightarrow Host:$ S
2. $SMC \longleftarrow Host:$ $RND^H \| R_s(H_p) \| H_p$
3. $SMC \longrightarrow Host:$ $R_s(S_p) \| S_p \| S_s(H_p[RND^S] \| RND^H)$
4. $SMC \longleftarrow Host:$ $H_s(S_p[RND^S] \| RND^H)$
5. $SMC \longrightarrow Host:$ $RND^S(RT)$
6. $Host \longrightarrow FPS:$ RND
7. $Host \longleftarrow FPS:$ $SF(RND, MD)$
8. $Host \longrightarrow Ausgabe:$ $Ja/Nein$

In der folgenden Spezifikation werden die Teilnehmer folgendermassen abgekürzt:

- Host = H
- Smartcard (SMC) = S
- Fingerprintscanner (FPS) = F

Das Protokoll kann in zwei Teile zerlegt werden.
Der erste Teil geht von Schritt 1.-5. Dabei ist die Smartcard der Initiator und der Host der Antwortende. Der zweite Teil geht von Schritt 6.-8. und darin ist der Host der Initiator und der Fingerprintscanner der Antwortende.

Spezifikation des Protokolls (Marrero)

Das Protokoll wird nacheinander für jeden Teilnehmer als eine Abfolge von versendeten und empfangenen Nachrichten spezifiziert.

Zur Erklärung der Umsetzung des Protokolls in die Spezifikationssprache soll das Vorgehen für den Teilnehmer *Smartcard* erläutert werden. Diese nimmt am obigen Kommunikations-Protokoll von Schritt 1 bis Schritt 5 teil. Kommentare werden jeweils zwischen /* und */ angeben und nach der Spezifikation der einzelnen Protokollschritte angegeben.
Eine Spezifikation des gesamten Kommunikations-Protokolls (ohne Kommentare) findet sich in Anhang A.

Smartcard

- [
  (beginit (*var* H))
  /* 'beginit' steht immer für den Beginn einer Kommunikation. '*var* H' steht für einen Teilnehmer, den die Smartcard für H hält. */

- (send (*var* H)
  (S))

  /* Protokollschritt 1: Die Smartcard S schickt an den Teilnehmer H die Nachricht S. */

- (receive (*var* H)
  (concat (*var* RND^H)
  (encrypt (privkey R)(pubkey (*var* H)))
  (pubkey (*var* H))))

  /* Protokollschritt 2: Die Smartcard S erhält von dem Teilnehmer H eine Nachricht bestehend aus:

  1. einer Zufallszahl RND^H.
  2. dem öffentlichen Schlüssel von Teilnehmer H verschlüsselt mit dem geheimen Schlüssel des Root R.
  3. dem öffentlichen Schlüssel von Teilnehmer H.

  'concat' steht für die Konkatentation von Nachrichten.
  Schlüssel werden folgendermassen aufgebaut:

  (Art_des_Schlüssels Besitzer_des_Schlüssels)

  'encrypt' steht für die Verschlüsselung. Dabei steht der Schlüssel an erster Stelle, die zu verschlüsselnde Nachricht an zweiter. */

- (newsecret (*var* RND^S))

  /* Interner Schritt, in dem die Smartcard S die Zufallszahl RND^S erzeugt. Da diese geheim bleiben soll, wird sie als 'newsecret' spezifiziert. */

- (send (*var* H)
  (concat (encrypt (privkey R)(pubkey S))
  (pubkey S)
  (encrypt (privkey S)(concat (encrypt (pubkey(*var* H))(*var* RND^S)
  (*var* RND^H)))))

  /* Protokollschritt 3: Die Smartcard S schickt dem Teilnehmer H eine Nachricht bestehend aus:

  1. dem öffentlichen Schlüssel von Smartcard S verschlüsselt mit dem geheimen Schlüssel des Root R.
  2. dem öffentlichen Schlüssel von Smartcard S.
  3. – Der Zufallszahl RND^S verschlüsselt mit dem öffentlichen Schlüssel des Teilnehmer H und
     – der Zufallszahl RND^H.

     Und diese Daten gemeinsam verschlüsselt mit dem geheimen Schlüssel der Smartcard S.

  */

- (receive (*var* H)
  (encrypt (privkey (*var* H))(concat (encrypt (pubkey S)(*var* RND^S))
  (*var* RND^H)))))

  /* Protokollschritt 4: Die Smartcard S empfängt vom Teilnehmer H eine Nachricht bestehend aus:

  - Der Zufallszahl RND^S verschlüsselt mit dem öffentlichen Schlüssel der Smartcard S und
  - der Zufallszahl RND^H.

  Und diese Daten werden gemeinsam mit dem geheimen Schlüssel des Teilnehmer H verschlüsselt. */

- (newsecret (*var* RT))

  /* Interner Schritt: Die Smartcard S liest den Fingerabdruck RT aus ihrem Speicher. Dieser soll geheim bleiben. */

- (send (*var* H)
  (encrypt (*secret* RND^S)(*var* RT)))

  /* Protokollschritt 5: Die Smartcard S verschlüsselt den Fingerabdruck RT mit RND^S als Schlüssel und schickt diese Nachricht an den Teilnehmer H. */

- (endinit (*var* H))

  Die Smartcard beendet die Kommunikation mit Teilnehmer H durch 'endinit'.

-]

Auf dieselbe Art wie bei der Smartcard gezeigt, lassen sich auch der Host und der Fingerprintscanner anhand ihrer Kommunikation spezifizieren.
Die konkrete Umsetzung findet sich in Anhang A.

Der Angreifer wird auf andere Weise spezifiziert und wird deshalb nun kurz erläutert.

Lokaler Speicher des Angreifer zu Beginn

(S H F *Angreifer* (pubkey S) (pubkey H) (pubkey *Angreifer*) (privkey *Angreifer*))

Der Angreifer wird durch sein Anfangswissen spezifiziert. Deshalb definiert man nur seinen lokalen Speicher zu Beginn des Protokolls.
Nach Definition kennt der Angreifer zu Beginn alle Namen der Teilnehmer, deren öffentliche Schlüssel und sein eigenes Schlüsselpaar.
Im Laufe des Protokolls wird der lokale Speicher des Angreifers dann durch all das Wissen aufgefüllt, das er aus den abgefangenen Nachrichten ableiten kann.

6.3 Asynchrone Produktautomaten (Gürgens/Ochsenschläger/Rudolph)

In *kryptographische Protokolle als ein System von Protokollagenten mittels asynchroner Produktautomaten (APA)* [53] unterteilen die Autoren den Zustand eines Teilnehmers (Agenten) in mehrere Komponenten. Diese Komponenten beschreiben jeweils welche Schlüssel der Teilnehmer kennt, wie seine Sicht auf das Protokoll ist und welche Ziele durch das Protokoll erreicht werden sollen. Die Kommunikation wird durch das Hineinlegen und Wegnehmen von Nachrichten in bzw. aus einer für alle Teilnehmer zugänglichen Komponente *Network* modelliert.
Bei ihrem Ansatz wird der Angreifer als ein weiterer Protokoll-Teilnehmer modelliert, der Zugriff auf die Komponente *Network* hat. Dadurch kann er alle verschickten Nachrichten lesen. Aufgrund der Informationen, die sich aus diesen Nachrichten generieren lassen und aufgrund seines Anfangswissen kann er dann neue Nachrichten erzeugen und in *Network* ablegen.

Für die Analyse und Verifikation verwenden die Autoren als Werkzeug das Simple-Homomorphism-Verification-Tool (SHVT) [55], welches die Verletzung von Sicherheitszielen automatisch mittels Erreichbarkeitsanalyse findet.

6.3.1 Formale Definition eines APA

Man kann einen APA als Menge elementarer Automaten betrachten. Die einzelnen Zustandskomponenten definieren den Zustand des APA. Das Kreuzprodukt der zu den einzelnen Komponenten gehörenden Zustandsmengen liefert die Zustandsmenge des APA. Damit sind die Zustandsmengen der elementaren Automaten eine Teilmenge des Kreuzprodukts einiger Zustandskomponenten des APA. Die elementaren Automaten haben jeweils Zugriff auf einige der Zustandskomponenten und kommunizieren darüber, dass sie den Inhalt ihrer gemeinsamen Komponenten verändern.

Sei $\mathcal{S}$ die Indexmenge für die Zustandsmengen, $\mathcal{E}$ die Indexmenge für die elementaren Automaten und $\mathcal{P}(\mathcal{S})$ die Potenzmenge von $\mathcal{S}$.

Für jeden elementaren Automaten $EA_e = (\phi_e, \tau_e, \Delta_e)$ mit $e \in \mathcal{E}$, sei

- ϕ_e sein Alphabet und
- $\Delta_e \subseteq \tau_e \times \phi_e \times \tau_e$ seine Zustands-Überführungs-Funktion, wobei
    - $\tau_e \subseteq \otimes_{\mathcal{S} \in N(e)}(Z_{\mathcal{S}})$ seine Zustandsmenge ist.

Die Zustandsmenge τ_e eines elementaren Automaten ist die Teilmenge des Kreuzprodukts aller benachbarten Zustandsmengen $Z_{\mathcal{S}}, \mathcal{S} \in N(e)$. Deshalb sind die

elementaren Automaten über die Zustandskomponenten $Z_{\mathcal{S}}$ mit der Nachbarrelation verbunden.
Durch

- $\forall_{e \in \mathcal{E}} : N(e) \neq \emptyset$
- $\mathcal{S} = \cup_{e \in \mathcal{E}} N(e)$

wird festgelegt, dass jeder elementare Automat mindestens eine benachbarte Zustandskomponente hat und jede Zustandskomponente mit mindestens einem elementaren Automaten verbunden sein muss.

Dann ist ein APA A formal geschrieben

$$A = [(Z_s)_{s \in \mathcal{S}}, (EA_e)_{e \in \mathcal{E}}, N, z_0]$$

mit

- $N : \mathcal{E} \rightarrow \mathcal{P}(\mathcal{S})$
- $z_0 \in \otimes_{s \in \mathcal{S}}(Z_s)$

d.h. er besteht aus

- einer Familie von Zustandsmengen (Z_s)
- einer Familie elementarer Automaten (EA_e)
- einer Nachbarschaftsrelation N und
- einem Startzustand z_0.

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

6.3.2 Anwendung des APA auf das PC-Login-Protokoll

Die Struktur des zugehörigen APA sieht man in Abb.6.1.
Die Kreise stellen die Zustandskomponenten dar und die Rechtecke die elementaren Automaten. Die Kanten geben an, auf welche Zustandskomponenten der jeweilige elementare Automat Zugriff hat.
Der dargestellte APA hat die drei elementaren Automaten Smartcard (S), Host (H) und Fingerprint-Scanner (F). Die drei elementaren Automaten sind über die Zustands-Komponente $Network$ miteinander verbunden. In dieser werden die Nachrichten abgelegt und abgeholt. Außerdem hat jeder elementare Automat noch 4 private Zustandskomponenten $State_i, Symkey_i, Asymkey_i$ und $Goal_i$ ($i \in \{S, H, F\}$) auf die niemand ausser dem jeweiligen elementaren Automaten

selbst Zugriff hat.
$State_i$ fungiert als Speicher und enthält zu Beginn des Protokolls die Namen und jeweiligen Rollen der anderen Teilnehmer, sowie Informationen mit wem eine Kommunikation gestartet bzw. auf welche Aufforderungen reagiert werden soll. $Symkey_i$ enthält die symmetrischen, $Asymkey_i$ die asymetrischen Schlüssel des Teilnehmer i. In $Goal_i$ werden im Laufe des Protokolls die (Sicherheits-)Ziele, die das Protokoll erreichen soll, gespeichert. Diese werden als Formeln die erfüllt sein müssen angegeben.

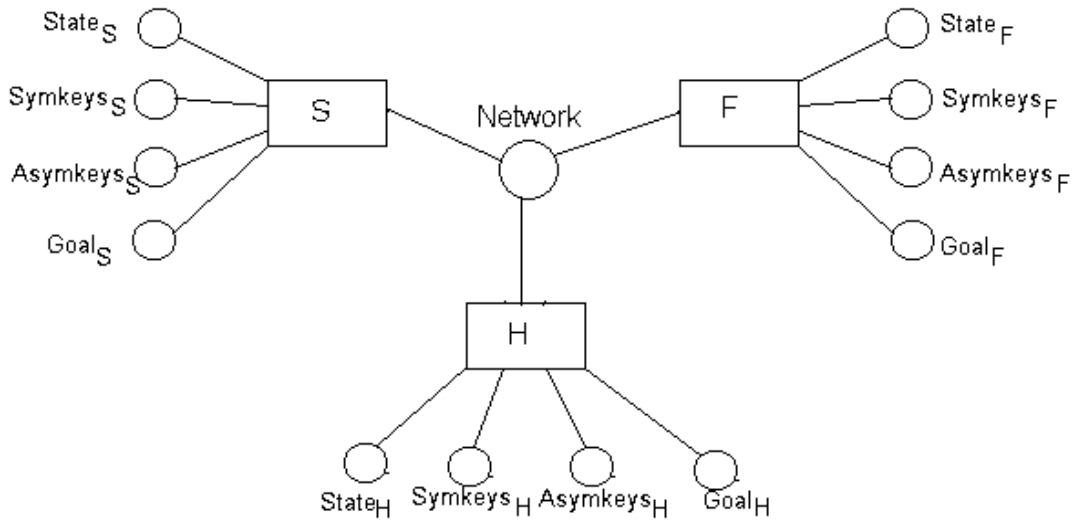

Abbildung 6.1: Darstellung des PC-Login-Protokolls als APA nach [53]

Nun muss man die Menge der Nachrichten (insbesondere die kryptographischen Algorithmen und Schlüssel), die Zustandsmengen für alle Komponenten und die Zustands-Übergangs-Relationen der elementaren Automaten genau beschreiben. Desweiteren wird der Anfangszustand der globalen Zustandskomponenten festgelegt. $Network$ dient dazu, dass im späteren Analyse-APA (siehe 6.2.4 und Abb 6.2) eine $Network$-Komponente in zwei Teile zerlegt werden kann: eine, aus der der Angreifer liest und eine, in die er schreibt. Die weiteren globalen Zustands-Komponenten dienen dazu neue Zufallszahlen, Schlüssel oder Labels zu erzeugen. Auf sie hat jeder Teilnehmer Zugriff.

Bei der Spezifikation wird im Folgenden zuerst der Protokoll-Schritt angegeben, dann seine Umsetzung in der Spezifikation.

Für den Aufbau der Spezifikation schlägt [54] folgende Vorgehensweise vor:

1. Initialzustand der globalen Komponenten
   Deklaration der Rollen
   Initialzustand der Zustandskomponenten

2. Transitionsmuster zur Spezifikation der Protokollschritte

In [54] werden die Rollen mit Variablen als Namen definiert und als dritter Schritt mit konkreten Agentennamen belegt. Da sich diese Arbeit jedoch auf die Analyse des PC-Login-Protokolls beschränkt, werden in der folgenden Spezifikation die Agenten schon bei der Rollendefinition mit ihren konkreten Namen belegt.

Die Transitionsmuster zur Spezifikation der Protokollschritte sind folgendermassen aufgebaut:

- Angeben der auftretenden Variablen
- Test der Vorbedingungen, unter denen eine Transition stattfinden kann, angezeigt durch $'?'$ und $' ='$
- Aktionen des Teilnehmers, welche sein können:
  - Lesen der Nachricht aus $Network$, angezeigt durch $' << Network'$
  - Zuweisungen, angezeigt durch $' :='$
  - Generieren einer Zufallszahl, eines Schlüssels oder Labels, angezeigt durch $' << new\_(nonce \| key \| label)'$
  - Speichern eines Tupels, angezeigt durch $' >> State'_i$
  - Speichern eines Sicherheitsziels, angezeigt durch $'>> S\_Goal'$
  - Abschicken einer neuen Nachricht, angezeigt durch $' >> Network'$

6.3.3 Spezifikation des Protokolls (SHVT)

Die Spezifikation für das SHVT ist folgendermassen aufgebaut:

1. Die globalen Zustandskomponenten werden definiert.
   Dazu gehört immer die Komponente $Network$:

   ```
   def_state Network: net_elem_seq := ::, Network_send, Network_rec;
   ```

   Ausserdem werden hier globale Komponenten definiert die z.B. neue Schlüssel liefern:

   ```
   def_state new_key: Nonce_seq := [111,222,333,444,555];
   ```

2. Die Teilnehmer und ihre Rollen werden festgelegt.
   Dabei werden die Rollen als Kommentar hinter der Definition des Teilnehmers angegeben. z.B.:

   ```
   def_role S; /* Initator des ersten Protokoll-Teils */
   def_role H; /* Reagierender des ersten Protokoll-Teils; Initator des zweiten Protokoll-Teils
   */
   ```

3. Das Ausgangswissen der einzelnen Teilnehmer wird festgelegt.
   Dieses besteht aus:

   - dem lokalen Speicher.
     z.B.:
     def_state H State: Messages_seq :=
     $[S,'agent'].[R,'server']['respond',S].[Zertifikat,[crypt,(R,priv,1),[(H,'pub',1)]]]$;
     Im lokalen Speicher stehen zum einen die dem Teilnehmer bekannten weiteren Teilnehmer am Protokoll und deren Rolle, z.B. [S,'agent']. Zum anderen stehen hier Zustände wie z.B ['respond',S], was aussagt, dass H auf eine Kommunikationsanfrage von S reagieren kann. Ausserdem können noch Zertifikate gespeichert werden;
     z.B.[Zertifikat,[crypt,(R,priv,1),[(S,'pub',1)]]].
   - den symmetrischen Schlüsseln:
     z.B. def_state H Symkeys: Symkeys_seq := (F, 'sym', ((H,F), 'sym', 1));
     Diese Schlüsseln sind folgendermassen aufgebaut:
     ('Schlüsselpartner', Art des Schlüssels, ((Schlüssel), Schlüsselart, natürliche Zahl)).
   - den asymmetrischen Schlüsseln:
     z.B. (H,'priv',(H_s,'priv',1)).(H,'pub',(H_p,'pub',1)).(R,'pub',(R_p,'pub',1));
     Diese sind ähnlich aufgebaut wie die symmetrischen Schlüssel. Lediglich steht nun anstelle des 'Schlüsselpartners' der Besitzer des Schlüssels.
   - den Sicherheitszielen:
     z.B. def_state H Goal: Goals_seq := : : ;
     Zu Beginn ist diese Komponente immer leer. Im Laufe des Protokolls werden hierin die Sicherheitsziele gespeichert, die das Protokoll erreichen soll.

4. Die einzelnen Protokollschritt werden spezifiziert. Dies soll am Beispiel des 3. Protokollschrittes erläutert werden. Ein Spezifikation des gesamten Kommunikations-Protokolls aus Tabelle 3.6 ist in Anhang B angegeben.

/* 3. Protokoll-Schritt: $S \longrightarrow H$: $R_s(S_p)\|\|S_s(H_p[RND_S]\|\|RND_H\|\|Host)$ */	
def_trans_pattern S step_3	*In Schritt 3 ist S der Sender.*

(X,H,M,Zert,M_list,SS,SP,HP,HP1,RP,Keys,RND_H,RND_S)	*Angeben aller in Schritt 3 verwendeten Variablen*
$/*\ X?Agents$, $H?Agents$, $M?Messages$, $M\_list?Messages$, $R_p?Keys$, $SS?Keys$, $SP?Keys$, $HP?Keys$, $HP1?Keys$, $Keys?Nonce\_seq$, $RND\_H?Nonce\_seq$, $RND\_S?Sessionkeys\ */$	*Zwischen /* und */ werden nun als Kommentar die Rollen der Variablen im Protokollschritt näher erklärt.*
(X,S,M_list) << Network,	*S liest aus Network eine Nachricht M_list von einem Teilnehmer X.*
$[Zertifikat, Zert]$? S_State,	*Prüft, ob das Zertifikat in der State-Komponente von S enthalten ist.*
$(S,'priv', SS)?S\_Asymkeys$, $(S,'pub', SP)?S\_Asymkeys$, $(R,'pub', RP)?S\_Asymkeys$,	*Weitere Prüfungen*
RND_H := elem(1,M_list),	*Der Variable RND^H wird das erste Element aus M_list zugewiesen.*
$M := elem(2, M\_list)$, $HP := elem(3, M\_list)$, $H := elem(4, M\_list)$, $HP1 := elem(1, decrypt(RP, M))$,	*Weitere Zuweisungen*
HP = HP1,	*Der nächste Schritt wird nur ausgeführt wenn diese Bedingung erfüllt ist*
(H,'pub',HP) >> S_Asymkeys,	*Der öffentliche Schlüssel von H wird in der Menge der S bekannten asymmetrischen Schlüssel gespeichert.*
/* IF HP == HP1 THEN (H,'pub',HP) >> H_Asymkeys, ELSE ABBRUCH */	*Prüfen, ob der Schlüssel aus dem Zertifikat mit dem übermittelten Schlüssel übereinstimmt.*
Keys << new_nonce,	*Die globale Komponente new_nonce wird ausgelesen.*
RND_S :=('sessionkey','sym',head(Keys)),	*Das erste Element von Keys wird für einen Sessionkey verwendet und dieser als RND^S gespeichert.*

tail(Keys) >> new_nonce,	*Der Rest von Keys wird zurück in new_nonce geschrieben.*
$[H, RND\_H]$ >> S_State,	*RND^H wird zusammen mit der Information, dass sie von H stammt in der State-Komponente von S gespeichert.*
$[new\_session\_key, H, RND\_S]$ >> S_State,	*Der Sessionkey RND^S, der mit H vereinbart werden soll, wird in der State-Komponente gespeichert.*
(S,H,[Zert,SP,crypt(SS,[crypt(HP,[RND_S]),RND_H])]) >> Network;	*S erzeugt eine Nachricht für H und legt diese in Network ab. Die Nachricht besteht aus dem Zertifikat, dem öffentlichen Schlüssel von S und einem mit dem geheimen Schlüssel von S verschlüsselten Nachrichtenteil.*

Analyse des APA (ohne Angreifer) durch das SHVT

Abb 6.2 zeigt die Ausgabe des SHVT nach der Analyse des PC-Login-Protokolls. Aussage dieser Mitteilung ist lediglich, dass die angefertige Protokoll-Spezifikation keine syntaktischen Fehler enthält. Eine Aussage über die Sicherheit des Protokolls wird an dieser Stelle noch nicht getroffen.
Abb.6.3 zeigt den Ausschnitt des vom SHVT erzeugten Reachability Graph für die ersten drei Protokoll-Schritte.
Dabei stellen M-1, M-2, M-3 die Zustände nach bzw. vor den Schritten dar.

Das SHVT kann auch die zum Reachability Graph gehörenden Operationen ausgeben. Diese sind für die ersten beiden Schritte im Folgenden dargestellt.

M-1 S_step_1 M-2	Beschrieben wird der Zustand M-1 bevor Schritt 1 des Protokolls ausgeführt wird, der in Zustand M-2 führt.
F_Asymkeys: 1<::> F_State: $1< [H, agent].[respond, H] >$ F_Symkeys: $1< (H, sym, ((H, F), sym, 1)) >$	Beschreibt den Initialzustand des Teilnehmer F (siehe 6.3.3)
Goals: 1<::>	Die Komponente Goals ist im Initialzustand leer.
H_Asymkeys: $1< (H, priv, (H, priv, 1)).(H, pub, (H, pub, 1)).$ $(R, pub, (R, pub, 1)) >$ H_State: $1< [F, agent].[R, server].[S, agent].[Zertifikat,$ $[crypt, (R, priv, 1), [(H, pub, 1)]]].[respond, S].[start, F] >$ H_Symkeys: $1< (F, sym, ((H, F), sym, 1)) >$	Beschreibt den Initialzustand des Teilnehmer H (siehe 6.3.3)

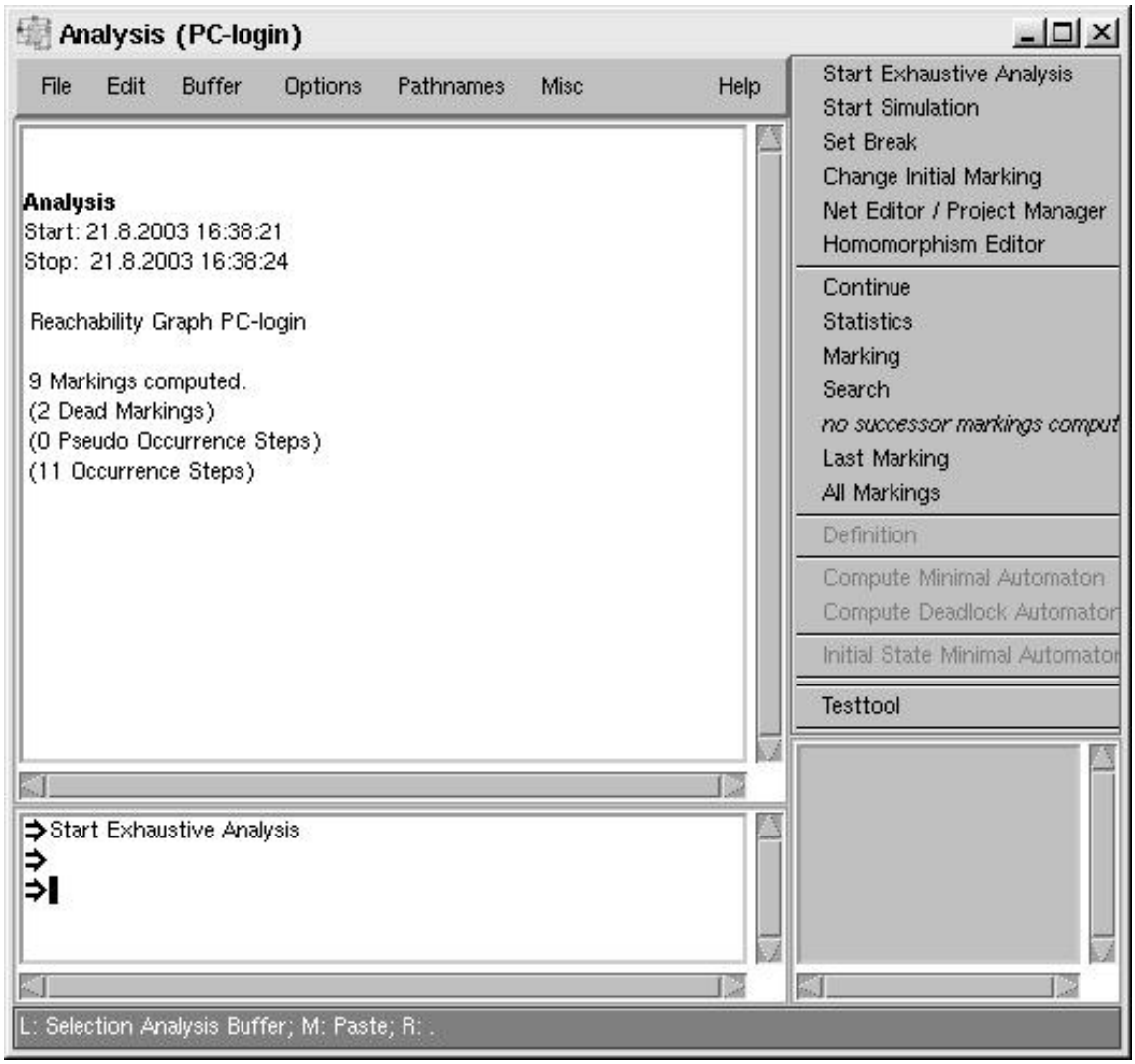

Abbildung 6.2: Screenshot der Ausgabe der SHVT nach der Analyse

Network: $1<::>$	Die Komponente Network ist im Initialzustand leer.
R_Asymkeys: $1< (R, priv, (R, priv, 1)).(R, pub, (R, pub, 1)) >$ R_State: $1<::>$ R_Symkeys: $1<::>$	Beschreibt den Initialzustand des Teilnehmers R (siehe 6.3.3)
S_Asymkeys: $1< (R, pub, (R, pub, 1)).(S, priv, (S, priv, 1)).$ $(S, pub, (S, pub, 1)) >$ S_State: $1< [H, agent].[R, server].[Zertifikat,$ $[crypt, (R, priv, 1), [(S, pub, 1)]]].[start, H] >$ S_Symkeys: $1<::>$	Beschreibt den Initialzustand des Teilnehmers S (siehe 6.3.3)

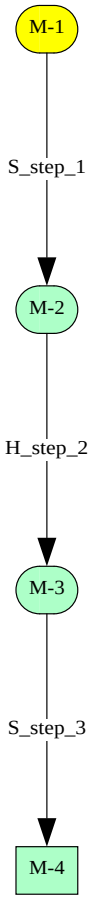

Abbildung 6.3: Ausschnitt aus dem Reachability Graph der Analyse des APA durch das SHVT

new_key: $1< [111, 222, 333, 444, 555] >$ new_label: $1< [1234, 1234] >$ new_nonce: $1< [13, 17, 19, 23] >$	Initialzustand der Komponente new_key ist eine Liste von Schlüsseln Initialzustand der Komponente new_label ist eine Liste von Zahlen Initialzustand der Komponente new_nonce ist eine Liste von Zahlen
M-1 S_step_1 M-2 H_step_2 M-3	Beschrieben wird der Zustand M-2 nachdem Schritt 1 ausgeführt wurde und bevor Schritt 2 ausgeführt wird, der in Zustand M-3 führt.

F_Asymkeys: $1<::>$ F_State: $1< [H, agent].[respond, H] >$ F_Symkeys: $1< (H, sym, ((H, F), sym, 1)) >$	Da Teilnehmer F an den ersten Schritten des Protokolls nicht beteiligt ist ändert sich dessen Zustand nicht.
Goals: $1<::>$	Die Sicherheitsziele die das Protokoll erfüllen soll, sind an dieser Stelle noch nicht definiert.
H_Asymkeys: $1< (H, priv, (H, priv, 1)).(H, pub, (H, pub, 1)).$ $(R, pub, (R, pub, 1)) >$ H_State: $1< [F, agent].[R, server].[S, agent].[Zertifikat,$ $[crypt, (R, priv, 1), [(H, pub, 1)]]].[respond, S].[start, F] >$ H_Symkeys: $1< (F, sym, ((H, F), sym, 1)) >$	Teilnehmer H hat zu diesem Zeitpunkt noch nicht am Protokoll teilgenommen, deshalb ist sein Zustand unverändert.
Network: $1< (S, H, [S]) >$	Teilnehmer S hat in Schritt 1 für Teilnehmer H die Nachricht [S] in Network abgelegt.
R_Asymkeys: $1< (R, priv, (R, priv, 1)).(R, pub, (R, pub, 1)) >$ R_State: $1<::>$ R_Symkeys: $1<::>$	Teilnehmer R nimmt nie am Protokoll teil und liefert nur das benötigte Root-Schlüsselpaar. Deshalb verändert sich der Zustand des Teilnehmers R nie.
S_Asymkeys: $1< (R, pub, (R, pub, 1)).(S, priv, (S, priv, 1)).$ $(S, pub, (S, pub, 1)) >$ S_State: $1< [H, agent].[R, server].[Zertifikat, [crypt,$ $(R, priv, 1), [(S, pub, 1)]]] >$ S_Symkeys: $1<::>$	Teilnehmer S hat einen Protokolllauf mit Teilnehmer H gestartet, deswegen hat er [start,H] aus seiner Zustandsmenge gelöscht.
new_key: $1< [111, 222, 333, 444, 555] >$ new_label: $1< [1234, 1234] >$ new_nonce: $1< [13, 17, 19, 23] >$	Aus keiner dieser globalen Komponenten wurde bisher etwas gelesen. Ihr Zustand hat sich damit nicht geändert.

Auf diese Weise gibt das SHVT alle Zustände an, die bei einem Protokolldurchlauf erreicht werden. Eine vollständige Abfolge aller durchlaufen Zustände findet sich in Anhang B.2. Ein Vergleich dieser Abfolge mit dem ursprünglichen Protokoll kann dazu verwendet werden sicher zu stellen, dass die Spezifikation richtig angefertigt wurde.

6.3.4 Erweiterung zum Analyse-APA

Nun ist das Protokoll als solches spezifiziert. Um es aber nun bzgl. seiner Sicherheit zu analysieren muss man diesen 'Protokoll-APA' nun zu einem Analyse-APA erweitern. Dieser ist in Abb.6.4 dargestellt. Dazu wird ein Angreifer A mit seinen zugehörigen Zustandskomponenten als elementarer Automat spezifiziert und zum APA hinzugefügt. Statt *Goal* verfügt der Angreifer über eine Zustands-Komponente *Sent*, welche verhindern soll, dass eine Nachricht zweimal abgeschickt wird. Der Angreifer hat ebenfalls Zugriff auf die Network-Komponente.

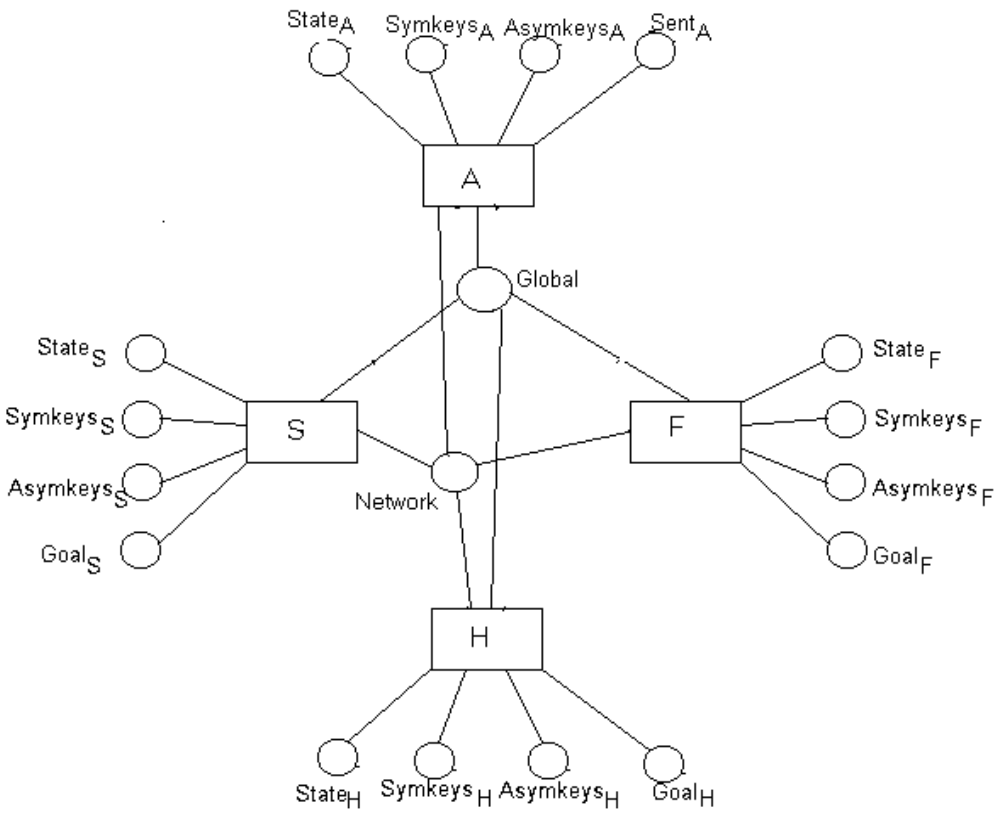

Abbildung 6.4: Darstellung des Analyse-APA für das PC-Login-Protokoll

Er kann aus ihr jedes Tupel entfernen, auch wenn er nicht der Adressat ist. Aus diesen Nachrichten kann er dann, zusammen mit seinem aktuellen Wissen, versuchen neues Wissen zu generieren. Zudem kann der Angreifer anschließend dem eigentlichen Adressaten der entnommenen Nachricht eine eigene Nachricht zukommen lassen.
Das Wissen des Angreifers läßt sich rekursiv definieren.
Der Angreifer kennt:

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

2. alle Agenten, die als Absender oder Adressat von Nachrichten auftauchen, die er aus *Network* entnommen hat (1. bzw. 2. Komponente der Tupels aus *Network*).

3. alle Nachrichten, die er gelesen hat (3. Komponente des Tupels aus $Network$).
4. alle atomare Nachrichten der von ihm gelesenen Nachrichten, wobei verschlüsselte Nachrichtenteile atomar sind.
5. alle Nachrichten, die er durch Konkatenation generieren kann.
6. jeden Klartext den er durch Anwendung von *decrypt* auf eine Nachricht erzeugen kann, sofern er den passenden Schlüssel kennt.

Zudem kann der Angreifer:

1. neue Nachrichten durch Anwenden von *encrypt* generieren, wobei als Schlüssel jede ihm bekannte Nachricht verwendet werden kann.
2. Zufallszahlen erzeugen.

Erst hier kommen nun die *Goal*-Komponenten ins Spiel. In ihnen sind die Sicherheitsbedingungen festgehalten, die erfüllt sein müssen. Die an das PC-Login-Protokoll gestellten Sicherheitsanforderungen sind:

- Niemand ausser der Smartcard und dem Host kennt zu irgendeiner Zeit den Sitzungsschlüssel RND^S.
  ($\forall P \in Agents \backslash \{S, H\}$: $not\_knows(P, RND^S)$
- Niemand außer der Smartcard und dem Host kennt zu irgendeiner Zeit das Referenztemplate RT der Smartcard.
  ($\forall P \in Agents \backslash \{S, H\}$: $not\_knows(P, RT)$

Wenn man es streng nimmt kann man noch fordern:

- Niemand ausser dem Host und dem Fingerprint-Scanner kennt zu irgendeiner Zeit das aufgenommen Template MD.
  ($\forall P \in Agents \backslash \{H, F\}$: $not\_knows(P, MD)$

Die 'Scramble Function' ist nur sicher solange sie geheim ist. Außerdem versenden die meisten Fingerprint-Scanner ihre Daten im Klartext. Deshalb wird auf diese Forderung verzichtet.
Nun kann die Analyse in Form einer Zustandsraumsuche auf den erreichbaren Zuständen an Hand des SHV-Tools durchgeführt werden.

6.3.4.1 Modellierung des Angreifers

Die entscheidende Komponente bei der Modellierung eines Angreifers ist die Zustandskomponente 'C_Structure'. Diese bringt dem Angreifer die Struktur der Protokollnachrichten bei. Nur wenn er diese kennt, kann er später versuchen aus den gelernten Informationen neue 'passende' Nachrichten zu generieren.

Der Angreifer ist so spezifiziert, dass er aus jeder neuen Nachricht soviel Information wie möglich lernt. Er versucht also eine empfangene Nachricht mit seinem Wissen zu entschlüsseln. Gelingt ihm das nicht, so versucht er es später, wenn er neues Wissen hat, nocheinmal.
Über 'C_Structure' lassen sich zwei Angreifer unterscheiden:

- ein passiver Angreifer und
- ein generischer Angreifer.

Beim passiven Angreifer ist 'C_Structure' leer.
Er generiert also keine neuen Nachrichten, sondern lernt nur die gelesenen und legt sie dann wieder zurück in die Network-Komponente. Damit lernt er soviel Information wie möglich, greift aber nicht aktiv ins Protokoll ein.
Beim generischen Angreifer hat 'C_Structure' folgende Struktur:

```
def_state C_Structure: Structure_Message_seq:=

    (1,S,H,[encrypt(atom,[atom])]).
    (1,H,F,[atom]).
    (1,F,H,[encrypt(keys,[atom,atom])]);
```

Wie man an 'C_Structure' sieht besteht die Struktur einer Nachricht aus 4 Teilen:

1. Anzahl der Elemente, die der Angreifer braucht, um die Nachricht zusammenzubauen.
2. Sender
3. Empfänger
4. Die eigentliche Struktur. Dabei wird angegeben, welche Funktionen zum Verschlüsseln verwendet werden und es können bestimmte Schlüssel oder Datentypen vorgegeben werden.

Da die Nachrichten im PC-Login-Protokoll jeweils mit einem sicheren Schüssel verschlüsselt sind den der Angreifer nie kennenlernen wird, gibt man '1' als Anzahl der Elemente an, auch wenn die eigentliche Nachricht länger ist.
Damit verhindert man, dass der Angreifer versucht eigene Nachrichten zusammenzubauen und beschränkt ihn somit darauf nur nach passenden Nachrichten zu suchen.

Die vollständige Spezifikation der beiden Angreifer findet sich in Anhang B.4.

6.3.4.2 Analyse des Analyse-APA mit einem passiven Angreifer durch das SHVT

Abb 6.5 zeigt die Ausgabe des SHVT nach der Analyse des PC-Login-Protokolls mit einem passiven Angreifer. Aussage dieser Mitteilung ist, dass kein Protokollauf gefunden wurden wurde, der die spezifizierten Sicherheitsziele verletzt. Der

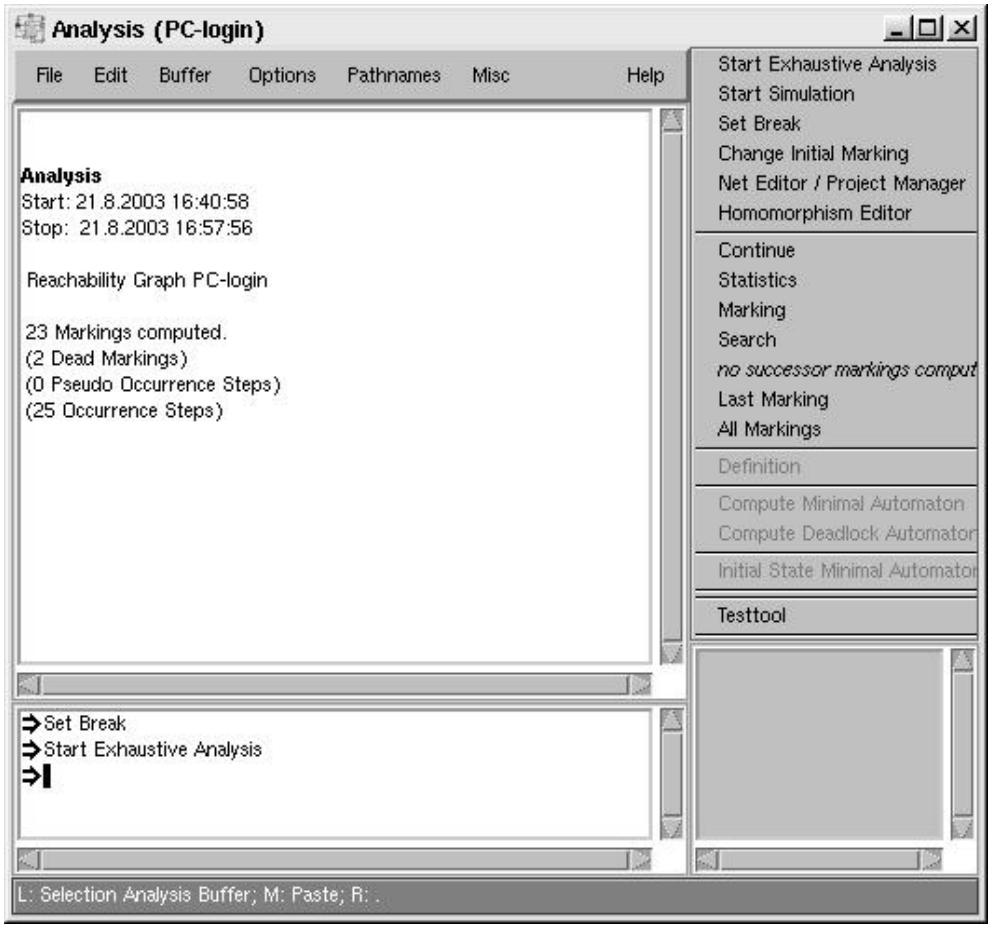

Abbildung 6.5: Screenshot der Ausgabe der SHVT nach der Analyse mit passivem Angreifer

passive Angreifer kommt also nie in den Besitz sicherheitssensibler Nachrichten.

Man sieht ebenfalls, dass sich die Anzahl der berechneten Markierungen und der aufgetretenen Schritte gegenüber der Analyse ohne Angreifer erhöht hat. Es werden nun 23 statt 9 Markierungen und 25 statt 11 Schritte berechnet. Die Berechnungszeit stieg damit von 4 Sekunden auf ca. 17 Minuten. Dies vermittelt einen Eindruck von der Vergrößerung des zu durchsuchenden Zustandsraumes.
Wie Abb.6.6 zeigt verändert die Anwesenheit eines (passiven) Angreifers die Zustandsfolge. Zwischen jedem Protokoll-Schritt tritt nun mindestens ein Lese-Schritt und ein Schreib-Schritt des Angreifers (hier genannt 'Charly') auf.
Im Folgenden werden nun auch für diese Analyse die ersten Schritte des Reachability Graph erläutert. Dabei stehen M-1 bis M-4 wieder für Zustände.

Der erste Zustand sieht genauso aus wie beim Ablauf ohne Angreifer, nur das noch zwei Kompontenen hinzugekommen sind, die den Anfangszustand des Angreifers

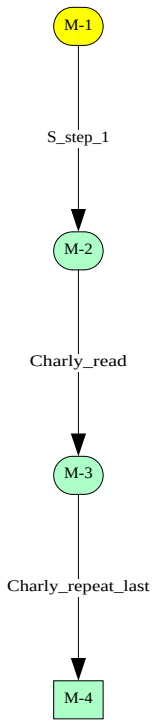

Abbildung 6.6: Ausschnitt aus dem Reachability Graph der Analyse des Analyse-APA mit passivem Angreifer durch das SHVT

beschreiben:

- Charly_State: 1< $[[], attack\_data]$ >
- Charly_Structure: 1<::>

Ausserdem gibt es jetzt eine Komponente Global, die den Beginn, bzw. den Erfolg eines Angriffs speichert.
Deshalb beginnt die Beschreibung mit dem zweite Zustand des Graphen aus Abb.6.6.

M-1 S_step_1 M-2 Charly_read M-3	Der Angreifer liest den Inhalt von Network_send, nachdem Teilnehmer S Schritt 1 ausgeführt hat.

Charly_State: 1< $[[], attack\_data]$ > Charly_Structure: 1<::>	Der Angreifer hat noch nichts getan; sein Zustand hat sich nicht geändert.
F_Asymkeys: 1<::> F_State: 1< $[H, agent].[respond, H]$ > F_Symkeys: 1< $(H, sym, ((H, F), sym, 1))$ >	Teilnehmer F ist auch noch im Initialzustand.
Global: 1< $[start\_attack]$ >	Der Angreifer nimmt am Protokoll teil, deshalb steht in Global [start,attack].
Goals: 1<::>	Protokoll-Ziele sind noch keine definiert.
H_Asymkeys: 1< $(H, priv, (H, priv, 1)).(H, pub, (H, pub, 1)).$ $(R, pub, (R, pub, 1))$ > H_State: 1< $[F, agent].[R, server].[S, agent].[Zertifikat,$ $[crypt, (R, priv, 1), [(H, pub, 1)]]].[respond, S].[start, F]$ > H_Symkeys: 1< $(F, sym, ((H, F), sym, 1))$ >	Teilnehmer H ist immernoch im Initialzustand.
Network_rec: 1<::> Network_send: 1< $(S, H, [S])$ >	Network ist nun in zwei Komponenten unterteilt. Network_send, woraus der Angreifer liest und Network_rec, worin der Angreifer die Nachricht, nach dem Lesen, unverändert wieder ablegt. In Network_send liegt in diesem Zustand die Nachricht, die der Teilnehmer S im ersten Schritt dort für Teilnehmer H abgelegt hat.
R_Asymkeys: 1< $(R, priv, (R, priv, 1)).(R, pub, (R, pub, 1))$ > R_State: 1<::> R_Symkeys: 1<::>	Der Zustand vom Root R ändert sich nie.
S_Asymkeys: 1< $(R, pub, (R, pub, 1)).(S, priv, (S, priv, 1)).$ $(S, pub, (S, pub, 1))$ > S_State: 1< $[H, agent].[R, server].[Zertifikat, [crypt,$ $(R, priv, 1), [(S, pub, 1)]]]$ > S_Symkeys: 1<::>	Aus der Zustands-Menge von Teilnehmer S wurde die Nachricht [start,H] entfernt, da S einen Protokolllauf mit Teilnehmer H begonnen hat.
new_key: 1< $[111, 222, 333, 444, 555]$ > new_label: 1< $[1234, 1234]$ > new_nonce: 1< $[13, 17, 19, 23]$ >	Diese globalen Komponenten sind unverändert.
M-2 Charly_read M-3 Charly_repeat_last M-4	Der Angreifer speichert die letzte gelesene Nachricht.
Charly_State: 1< $[S, H, [S], last\_message].[[S], attack\_data]$ > Charly_Structure: 1<::>	Der Angreifer hat die letzte gelesene Nachricht seiner Wissensmenge hinzugefügt. Dazu speichert er die gelesene Nachricht [S] in [[S],attack_data].

F_Asymkeys: 1<::> F_State: 1< $[H, agent].[respond, H]$ > F_Symkeys: 1< $(H, sym, ((H, F), sym, 1))$ >	unverändert
Global: 1< $[start\_attack]$ >	unverändert
Goals: 1<::>	unverändert
H_Asymkeys: 1< $(H, priv, (H, priv, 1)).(H, pub, (H, pub, 1)).$ $(R, pub, (R, pub, 1))$ > H_State: 1< $[F, agent].[R, server].[S, agent].[Zertifikat,$ $[crypt, (R, priv, 1), [(H, pub, 1)]]].[respond, S].[start, F]$ > H_Symkeys: 1< $(F, sym, ((H, F), sym, 1))$ >	unverändert
Network_rec: 1<::> Network_send: 1<::>	Der Angreifer hat die Nachricht aus Network_send entfernt.
R_Asymkeys: 1< $(R, priv, (R, priv, 1)).(R, pub, (R, pub, 1))$ > R_State: 1<::> R_Symkeys: 1<::>	unverändert
S_Asymkeys: 1< $(R, pub, (R, pub, 1)).(S, priv, (S, priv, 1)).$ $(S, pub, (S, pub, 1))$ > S_State: 1< $[H, agent].[R, server].[Zertifikat,$ $[crypt, (R, priv, 1), [(S, pub, 1)]]]$ > S_Symkeys: 1<::>	unverändert
new_key: 1< $[111, 222, 333, 444, 555]$ > new_label: 1< $[1234, 1234]$ > new_nonce: 1< $[13, 17, 19, 23]$ >	unverändert

Im nächsten Schritt legt nun der Angreifer die gelesene Nachricht wieder in Network_rec ab.
Anschließend liest Teilnehmer H diese und erzeugt eine neue Nachricht für Teilnehmer S, die er wiederum in Network_rec ablegt.

Eine Auflistung der gesammten Abfolge der Zustände dieser Analyse findet sich in Anhang B.3.

6.3.4.3 Analyse des Analyse-APA mit einem generischen Angreifer durch das SHVT

Der generische Angreifer versucht, im Gegensatz zum passiven Angreifer, neue 'passende' Nachrichten selber zu erzeugen. Deshalb probiert er alle möglichen Kombinationen aus und der Zustandsraum wächst gewaltig. Auf dem am Lehrstuhl vorhandenen System war eine Analyse des Protokolls mit einem generischen Angreifer selbst nach 9h noch nicht beendet.
Carsten Rudolph von der Fraunhofer Gesellschaft für Sichere Telekooperation

berichtet, dass eine Analyse mit einer etwas eingeschränkten Version des generischen Angreifers auf einem 1,8 GHz Pentium nach ca. 3h beendet war, ohne einen Angriff zu finden.
Demnach liegt die Vermutung nahe, dass auch ein generischer Angreifer nicht in der Lage ist die Sicherheitsziele des Protokolls zu verletzen.

Folgerung

Die Ergebnisse der Analyse des PC-Login-Protokolls mit dem SHVT lassen darauf schließen, dass

1. die Spezifikation des PC-Login-Protokoll syntaktisch korrekt ist.
2. für diese Spezifikation keine der aufgestellten Sicherheitsziele verletzt werden.

Nicht überprüft werden kann, ob die Spezifikation das PC-Login-Protokoll korrekt abbildet und ob die geforderten Sicherheitsziele ausreichend sind.
Jedoch wurde in den vorigen Kapiteln argumentiert warum davon ausgegangen wird, das dies der Fall ist.

Zusammenfassung

Die Eingabe-Einheit eines wissensbasierten Authentifikations-Systems kann nicht einfach durch einen Fingerprint-Scanner ersetzt werden, um daraus ein biometrisches Authentifikations-System zu machen. Dies liegt vorallem daran, dass die Eigenschaften biometrischer Merkmale einen anderen Matching-Algorithmus und einen anderen Aufbau des Authentifikations-Systems erfordern. So spielen bei biometrischen Systemen Begriffe wie FAR,FRR und EER eine Rolle und man muss einen Kompromiss zwischen Komfort und Sicherheit des Systems machen. Außerdem bringen die Eigenschaften biometrischer Merkmale auch Angriffsmöglichkeiten mit sich die wissenbasierte Authentifikations-Systeme nicht bieten.
Aus diesen Gründen muss bei der Ersetzung eines wissensbasierten durch ein biometrisches Authentifikations-System das gesamte Authentifikations-Protokoll geändert werden. Zudem müssen die Sicherheitsmassnahmen neu überdacht werden, da biometrische Merkmale eines Menschen nur in beschränkter Anzahl verfügbar sind und damit bei Kompromittierung nicht beliebig oft ausgetauscht werden können. Außerdem sind biometrische Daten aus Sicht des Datenschutzes hochsensible Daten, die eines besonderen Schutzes gegen Fremdzugriff bedürfen.

Um sichere biometrische Authentifikations-Protokolle zu erstellen, bietet sich der Ansatz von Abadi und Needham [21] an. Dessen Ziel ist es durch das Aufstellen von Prinzipien schon beim Entwurf der Protokolle bekannte Fehler zu vermeiden. Um bestehende Protokolle zu analysieren und Fehler zu finden werden formale Methoden verwendet, die Catherine Meadows in 4 Typen einteilt. Problem dabei ist es, dass nur bekannte Fehler gefunden werden können und die Methoden oft sehr speziell auf ein bestimmtes Protokoll zugeschnitten sind. Die Arbeit verwendet deshalb Model-checking für die Analyse des entworfenen Protokolls, da dabei zwar eine Abstraktion des Protokolls analysiert wird, die Methoden jedoch Protokoll-unspezifisch einsetzbar erscheinen.
Für die Analyse wurde zum einen der Ansatz der *Asynchronen Produktautomaten* der Fraunhofer-Gesellschaft [53] und zum anderen der *Model-checker* der CMU [51] mit den dazugehörigen Tools verwendet. Die Umsetzung des Protokolls in die Spezifikation ist bei beiden Tools übersichtlich und die Tools sind für akademische Zwecke verfügbar.

Die in dieser Arbeit untersuchten Sicherheitsziele beschränken sich auf Ziele, die durch ein Protokoll erfüllt werden können. Für das gesamte System ist jedoch ein geeigneter Matching-Algorithmus mit einer FAR/FRR, die an die Anforderungen des jeweiligen Systems angepasst ist, mindestens genauso sicherheitsrelevant.

Literaturverzeichnis

[1] Common Criteria - Common Methodology for Information Technology Security Evaluation: *Biometric Evaluation Methodology Supplement.* <www.cesg.gov.uk/technology/biometrics/media/BEM_10.pdf>(21.07.2003)

[2] Jahresbericht des Berliner Datenschutzbeauftragten (1998): *3.5 Biometrie - Sesam öffne Dich?* <www.datenschutz-berlin.de/jahresbe/98/teil3-5.htm>(21.07.2003)

[3] Thomas Petermann und Arnold Sauter - Büro für Technologiefolgen-Abschätzung beim Deutschen Bundestag: *Biometrische Identifikationssysteme - Sachstandsbericht.* <www.tab.fzk.de/de/projekt/zusammenfassung/ab_76.pdf>(21.07.2003)

[4] Michael Behrens und Richard Roth: *Biometrische Identifikation.* DuD-Fachbeiträge, ISBN 3-528-05786-6, Vieweg-Verlag, Braunschweig/Wiesbaden, 2001

[5] Helmut Bäumler, Lukas Gundermann und Thomas Probst - Unabhängiges Landeszentrum für Datenschutz Schleswig-Holstein, Kiel:: *Stand der nationalen und internationalen Diskussion zum Thema Datenschutz bei biometrischen Systemen.* <www.datenschutzzentrum.de/download/tabga.pdf>(21.07.2003)

[6] Arbeitsgruppe 6 - Biometrische Identifikationsverfahren: *Bewertungskriterien zur Vergleichbarkeit biometrischer Verfahren.* <www.teletrust.de/down/kritkat_2-0.zip>(21.07.2003)

[7] Bruno Struif, Dirk Scheuermann und Lothar Müller: *TB1 - BioNorm - Need for Specifications and Standardisation to Achive Interoperability in the Field of Smart Cards and Biometrics.* <eeurope-Smartcards.org/Documents/04-3%20TB1%20BIONORM.pdf>(21.07.2003)

[8] Utimaco: *Match on Card als Weg zur komfortablen Sicherheit.* <www.utimaco.com/ger/content_pdf/wp_sg_biometrics_long.pdf>(21.07.2003)

[9] SecCommerce Technologies AG: *Smartcards: Physikalische Sicherheit.* <www.seccommerce.de/de/fachwissen/technologie/Smartcards_physikalische_sicherheit.html>(21.07.2003)

[10] Bruce Schneier und Adam Shostack: *Sicherheitsprobleme von Smartcards. (Deutsch von P.Gühring)* <www.counterpane.com/smart-card-threats.html> (Original) (21.07.2003) <www.futureware.at/Smartcard> (Deutsch)) (21.07.2003)

[11] Gaël Hachez , François Koeune und Jean-Jacques Quisquater: *Biometrics, Access Control, Smartcards: A not so simple combination.* <www.dice.ucl.ac.be/crypto/publications/2000/Biometrics.pdf>(21.07.2003)

[12] Lisa Thalheim, Jan Krissler und Peter Ziegler: *Körperkontrolle.* <www.heise.de/ct/02/11/114/>(21.07.2003)

[13] Bics Wiessaw et. al.: *Ultrasonic setup for fingerprint patterns detection and evaluation.* <www.optel.pl/article/deutsch/article2.htm>(21.07.2003)

[14] Bics Wiessaw: *Wie kann die Zukunft des Geldautomaten aussehen?* <www.optel.pl/acrticle/deutsch/terminal.htm>(21.07.2003)

[15] Christoph Vogel, Stephan Beinlich und Ullrich Martini: *Die virtuelle PIN - Biometrie für real life Anwendungen.* <www.cifro.com/download/weissbuch_vpin_v_1_0.pdf>(21.07.2003)

[16] Dirk Scheuermann, Scarlet Schwiderski-Grosche und Bruno Struif: *Usability of Biometrics in Relation to Electronic Signature.* <www.gmd.de/publications/report/0118/>(21.07.2003)

[17] ISO/IEC JTC 1/SC 27: *Information Technology - Security techniques - Entity authentication - Part 3: Mechanisms using asymmetric encipherment algorithms*

[18] FINREAD (1999): *Financial transactional IC card reader project.* <www.finread.com/spip/index.php>(21.07.2003)

[19] FIPS: *Advanced Encryption Standard (AES), Data Encryption Standard (DES), Triple-DES, and Skipjack Algorithms.* <csrc.nist.gov/cryptval/des.htm>(21.07.2003)

[20] BSI: *EAL 5+ Zertifikat für Philips Smart Card Controller P8WE6017 V1I.* <www.bsi.de/presse/newslet/newslett/nl301.htm>(21.07.2003)

[21] Martin Abadi und Roger Needham: *Prudent Engineering Practice for Cryptographic Protocols.* <www.cse.ncsc.edu/~abadi/Papers/gep-ieee.ps>(21.07.2003)

[22] Roger Needham and Michael Schroeder: *Using encryption for authentication in large networks of computers.* Communications of the ACM, 21(12):993-999, December 1978

[23] Jennifer G. Steiner, Clifford Neuman und Jeffrey I. Schiller: *Kerberos: An authentication service for open network systems.* Usenix Conference Proceedings, pages 105-119, May 1992

[24] Dorothy Denning und Giovanni Sacco: *Timestamps in key distribution protocols.* Communications of the ACM, 24(8):533-535, August 1981

[25] Roger Needham und Michael Schroeder: *authentication revisited.* Operating Systems Review, 21(7), Januar 1987

[26] Catherine Meadows: *Applying formal methods to the analysis of a key managment protocol.* Journal of Computer Security, 1(1):5-35, 1992

[27] Aviel D. Rubin und Peter Honeyman: *Formal Methods for the Analysis of authentication Protocols.* <avirubin.com/survey.ps>(21.07.2003)

[28] Deepinder P. Sidhu: *Authenification protocols for computer networks: I.* Computer Networks and ISDN Systems, 11:297-310, 1986

[29] Vijay Varadharajan: *Use of a formal description technique in the specification of authentication protocols.* Computer Standards and Interfaces, 9:203-215, 1990

[30] Richard A. Kemmerer: *Analyzing encryption protocols using formal verification techniques.* IEEE Journal on Selected areas in Communications, 7(4):448-457, May 1989

[31] D. Longley and S. Rigby: *Use of expert systems in the analysis of key managment systems.* Security and Protection in Information Systems, pages 213-224, 1989

[32] Paul Syverson and Catherine Meadows: *A logical language for specifying cryptographic protocol requirements.* Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy, pages 165-177, May 1993

[33] Jonathan K. Millen, Sidney C. Clark and Sheryl B. Freedman: *The interrogator: Protocol security analysis.* IEEE Transactions on Software Engineering, SE-13(2):274-288, February 1987

[34] Michael Burrows, Michael Abadi and Roger Needham: *A logic of authentication.* ACM Transactions on Computer Systems, 8, February 1990

[35] Li Gong, Roger Needham and Raphael Yahalom: *Reasoning about belief in cryptograhic protocols.* Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy, pages 234-248, May 1990

[36] Pierre Bieber: *A logic of communication in a hostile environment.* Proceedings of the Computer Security Foundation Workshop III, page 14-22, June 1990

[37] Einar Snekkens: *Roles in cryptographic protocols.* Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy, pages 105-119, May 1992

[38] P. Venkat Rangan: *An axiomatic basis of trust in distributed systems.* Proceedings of the 1988 IEEE Computer Society Symposium on Research in Security and Privacy, pages 204-211, May 1988

[39] Paul Syverson: *A logic for the analysis of cryptograhic protocols.* Technical Report 9305, Naval Research Laboratory, December 19.

[40] Paul Syverson: *Formal semnatics for logics of cryptographic protocols.* Proceedings of the Computer Security Foundation Workshop III, pages 32-41,June 1910

[41] R. Kailar and V.D. Gilgor: *On belief evolution in authentication protocols.* Proceedings of the Computer Security Foundation Workshop IV, pages 103-116, June 1991

[42] Louise E. Moser: *A logic of knowledge and belief for reasoning about computer security.* Proceedings of the Computer Security Foundation Workshop II, pages 57-63, 1989

[43] Virgil D. Gligor, Rajashekar Kailar, Stuart G. Stubblebine and Li Gong: *Logics for cryptographic protocols - virtues and limitations.* Proceedings of the Computer Security Foundation Workshop IV, pages 219-226. June 1991

[44] Danny Dolev and Andrew C. Yao: *On the security of public-key protocols.* Communications of the ACM, 29198-208, August 1983

[45] Michael J. Merritt: *Cryptographic Protocols.* PhD thesis, Georgia Institute of Technology, 1983

[46] Catherine Meadows: *Using narrowing in the analysis of key management protocols.* Proceedings of the 1989 IEEE Computer Society Symposium on Research in Security and Privacy, pages 138-147, May 1989

[47] Catherine Meadows: *Representing partial knowledge in an algebraicsecurity model.* Proceedings of the Computer Security Foundation Workshop III, pages 23-31, June 1990

[48] Catherine Meadows: *A system for the specification and analysis of key management protocols.* Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy, pages 182-195, May 1991

[49] Edmund Clarke und Holger Schlingloff: *Model checking.* Chapter 21, Alan Robinson and Andrei Voronkov (eds.), Handbook of Automated Reasoning *Elesvier Science Publisher B.V., pp. 1367 - 1522 (2000)* <www.informatik.hu-berlin.de/~hs/Publikationen/2000_Handbook-of-Automated-Reasoning_Clarke-Schlingloff_Model-Checking.ps> (12.08.2003)

[50] Edmund Clarke, Will Marrero und Somesh Jha: *Model Checking for Security Protocols (1997).* <www-2.cs.cmu.edu/~marrero/tr.ps.gz>(13.08.2003)

[51] Edmund Clarke, Will Marrero und Somesh Jha: *Using State Space Exploration and a Natural Deduction Style Message Derivation Engine to Verify Security Protocols (1998).* <www-2.cs.cmu.edu/~marrero/procomet.ps.gz>(13.08.2003)

[52] Thomas Y.C. Woo und Simon S. Lam: *A semantic model for authentication protocols.* (Proceedings of the IEEE Symposium on Research in Security and Privacy, 1993)

[53] Sigrid Gürgens, Peter Ochsenschläger und Carsten Rudolph: *Role based specification and security analysis of cryptographic protocols using asynchonous product automata.* In DEXA 2002 International Workshop on Trust and Privacy in Digital Business. DEXA 2002. <www.sit.fhg.de/english/META/meta_publications/doc/Role_based_specification_and_analysis_with_APA.pdf> (18.07.2003)

[54] Carsten Rudolph and Sang-Hyeun Park: *Valikrypt - Spezifikation und Analyse kryptographischer Protokolle mit dem SHVT. (Tutorial)* <Direkt erhalten von: rudolphc@sit.fraunhofer.de>

[55] Peter Ochsenschläger, J. Repp and R.Rieke: *Abstraction and composition - a verification method for co-operating systems.* Journal of Experimental and Theoretical Artificial Intelligence, 12:447-459, June 2000.

Anhang A

Protokoll für das Marrero-Tool

A.1 Spezifikation des Protokolls (Marrero)

Lokaler Speicher des Angreifers zu Beginn

(S H F *Angreifer* (pubkey S) (pubkey H) (pubkey *Angreifer*) (privkey *Angreifer*))

Smartcard

```
[
(beginit      (*var* H))
(send         (*var* H)
              (S))
(receive      (*var* H)
              (concat          (*var* RND^H)
                               (encrypt (privkey R)(pubkey (*var* H)))
                               (pubkey (*var* H)))
(newsecret    (*var* RND^S))
(send         (*var* H)
              (concat          (encrypt (privkey R)(pubkey S))
                               (pubkey S)
                               (encrypt (privkey S)(concat          (encrypt (pubkey(*var* H)(*var* RND^S))
                                                                    (*var* RND^H)))))
(receive      (*var* H)
              (concat          (encrypt (privkey R)(pubkey (*var* H)))
                               (encrypt (privkey (*var* H))(concat  (encrypt (pubkey S)(*var* RND^S))
                                                                    (*var* RND^H)))))
(newsecret    (*var* RT))
(send         (*var* H)
              (encrypt         (*secret* RND^S)(*var* RT)))
(endinit      (*var* H))
]
```

Host

```
[
(begrespond   (*var* S))
(receive      (*var* S)
              (*var* S)
(newsecret    (*var* RND^H))
(send         (*var* S)
              (concat          (*var* RND^H)
                               (encrypt (privkey R)(pubkey H))
                               (pubkey H)))
(receive      (*var* S)
              (concat          (encrypt(privkey R)(pubkey *var* S))
                               (pubkey (*var* S))
                               (encrypt (privkey (*var* S))(concat    (encrypt(pubkey H)(*var* RND^S))
                                                                      (*var* RND^H))))
(send         (*var* S)
              (concat          (encrypt (privkey H)(concat            (encrypt (pubkey (*var* S)) (*var* RND^S))
                                                                      (*var* RND^H)))))
(receive      (*var* S)
              (encrypt         (*secret* RND^S)(*var* RT)))
(endrespond   (*var* S))
]
[
(beginit      (*var* F))
(newnonce     (*var* RND)
(send         (*var* F)
              (*var* RND))
(receive      (*var* F)
              (*var* SD))
(endinit      (*var* F))
]
```

Fingerprintscanner

```
[
(begrespond   (*var* H))
(receive      (*var* H)
              (*var* RND))
(newnonce     (*var* SD))
(send         (*var* H)
              (*var* SD))
(endrespond   (*var* H))
]
```

Die hier angegebene Protokollspezifikation wurde an Hand von [50, 51] erstellt. Leider gelang bisher nicht das zugehörige Tool von W. Marrero names 'public-Brutus' auf dem Rechnersystem der Universität zu installieren. Somit konnte die hier angegebene Protokollspezifikation nicht auf ihre Korrektheit und Sicherheit getestet werden.
Deshalb kann an dieser Stelle leider keine Aussage darüber gemacht werden, ob das Benutzer-Verifikations-Protokoll aus Tabelle 3.5 korrekt in die Spezifikationssprache umgesetzt wurde.

Anhang B

Protokoll für das SHVT

B.1 Spezifikation des Protokolls (SHVT)

```
/* PC-Login-Protokoll */

/* include 'functions.vsp'; */

/* Globale Zustandskomponenten */

def_state Network: net_elem_seq := ::, Network_send, Network_rec;

def_state new_nonce: Nonce_seq := [13,17,19,23];
def_state new_key: Nonce_seq := [111,222,333,444,555];
def_state new_label: Nonce_seq := [1234,1234];
def_state Goals: Goals_seq := ::;


/* Rollen der Teilnehmer */

def_role S; /* Initator des ersten Protokoll-Teils */
def_role H; /* Reagierender des ersten Protokoll-Teils; Initator des zweiten Protokoll-Teils */
def_role F; /* Reagierender des zweiten Protokoll-Teils */
def_role R; /* Protokoll-Root */


/* Definition der Ausgangszustände und des Ausgangswissens der einzelnen Teilnehmer */

def_state S State: Messages_seq := [H,'agent'].[R,'server']['start',H].[Zertifikat,[crypt,(R,priv,1),[(S,'pub',1)]]];
def_state S Symkeys: Symkeys_seq := : : ;
def_state S Asymkeys: Asymkeys_seq := (S,'priv',(S,'priv',1)).(S,'pub',(S,'pub',1)).(R,'pub',(R,'pub',1));
def_state S Goal: Goals_seq := : : ;

def_state H State: Messages_seq := [S,'agent'].[F,'agent'].[R,'server'].['respond',S].[crypt,(R,priv,1),[(H,'pub',1)]].['start',F];
def_state H Symkeys: Symkeys_seq := (F, 'sym', ((H,F), 'sym', 1));
def_state H Asymkeys: Asymkeys_seq := (H,'priv',(H,'priv',1)).(H,'pub',(H,'pub',1)).(R,'pub',(R,'pub',1));
def_state H Goal: Goals_seq := : : ;

def_state F State: Messages_seq := [H,'agent'].['respond',H];
def_state F Symkeys: Symkeys_seq := (H,'sym',((H,F),'sym',1));
def_state F Asymkeys: Asymkeys_seq := : : ;
def_state F Goal: Goals_seq := : : ;

def_state R State: Messages_seq := : : ;
def_state R Symkeys: Symkeys_seq := : : ;
def_state R Asymkeys: Asymkeys_seq := (R,'priv',(R,'priv',1)).(R,'pub',(R,'pub',1));
def_state R Goal: Goals_seq := : : ;


/* Spezifikation der Protokoll-Schritte */

/* 1. Protokoll-Schritt: S ⟶ H : S */
```

```
def_trans_pattern S step_1
(H)
/* H ? Agents,*/
['start', H] ? S_State,
[H,'agent'] ? S_State,
['start', H] << S_State,
(S,H,[S]) >> Network;

/* 2. Protokoll-Schritt: H --> S : RND^H||R_s(H_p)||H_p */
def_trans_pattern H step_2
(X,M,HP,RND_H,Nonce,Zert)
/* X ? Agents,
M ? Messages,
RND_H ? Nonce_seq */
(X,H,M) << Network,
[S,'agent'] ? H_State,
S = elem(1,M),
[Zertifikat, Zert] ? H_State,
(H,'pub',HP) ? H_Asymkeys,

    Nonce << new_nonce,
RND_H := head(Nonce),
tail(Nonce) >> new_nonce,
(RND_H,S) >> H_State,
['respond', S] << H_State,
[H, S, (RND_H, Zert, HP)] >> Network;

/* 3. Protokoll-Schritt: S --> H : R_s(S_p)||S_s(H_p[RND^S]||RND^H) */
def_trans_pattern S step_3
(X,M,Zert,M_list,SS,SP,HP,HP1,RP,Keys,RND_H,RND_S)
/* X ? Agents,
M ? Messages,
M_list ? Messages,
R_p ? Keys,
SS ? Keys,
HP ? Keys,
HP1 ? Keys,
Keys ? Nonce_seq,
RND_H ? Nonce_seq,
RND_S ? Sessionkeys */
(X,S,M_list) << Network,
[Zertifikat, Zert] ? S_State,
(S,'priv',SS) ? S_Asymkeys,
(S,'pub',SP) ? S_Asymkeys,
(R,'pub',RP) ? S_Asymkeys,
RND_H := elem(1,M_list),
M := elem(2,M_list),
HP := elem(3,M_list),
HP1 := elem(1,decrypt(RP,M)),
HP=HP1,
(H,'pub',HP) >> S_Asymkeys,

/* IF (HP == HP1)
THEN (H,'pub',HP) >> H_Asymkeys,
ELSE ABBRUCH
(Fuer den Abbruch kann nun durch ein Transitionsmuster definiert werden, wie der Folgezustand nach einem
Abbruch aussieht)*/

Keys << new_nonce,
RND_S :=('sessionkey','sym',head(Keys)),
tail(Keys) >> new_nonce,
[H, RND_H] >> S_State,
[new_session_key, H, RND_S] >> S_State,
(S,H,[Zert,SP,crypt(SS,[crypt(HP,[RND_S]),RND_H])]) >> Network;

/* 4. Protokoll-Schritt: H --> S: H_s(S_p[RND^S]||RND^H) */
def_trans_pattern H step_4
```

(X,M1,M2,M3,M_list,RND_H,RND_S,SP,SP1,HS,RP)
/* X ? Agents,
M1 ? Messages,
M2 ? Messages,
M3 ? Messages,
M_list ? Messages,
RND_H ? Nonce_seq,
RND_S ? Sessionkeys,
HS ? Keys,
SP ? Keys,
RP ? Keys */
(X,H,M_list) << Network,
$[RND\_H, S]$? H_State,
(H,'priv',HS) ? Asymkeys,
(R,'pub',RP) ? Asymkeys,
M1 := elem(1,M_list),
SP1 := elem(2,M_list),
M2 := elem(3,M_list),
SP := elem(1,crypt(RP,M1)),
M3 := elem(1,crypt(SP,M2)),
elem(2,crypt(SP,M2)) = RND_H,

SP=SP1,
/* IF (SP == SP1)
THEN (S,'pub',SP) >> H_Asymkeys,
ELSE ABBRUCH */

RND_S := decrypt(HS,M3),
$[session\_key, S, RND\_S]$ >> H_State,
/*($\forall P \in Agents \backslash \{S, H\} : not\_knows\{P, RND\_S\}$) >> H_Goal,*/
(H,conf,RND_S) >> Goals,
(H,S,[crypt(HS,[crypt(SP,$[RND\_S]$),RND_H])]) >> Network;

/* 5. Protokoll-Schritt: $S \longrightarrow H : RND^S(RT)$ */
def_trans_pattern S step_5
(X,M,M_list,RND_H,RND_S,HP,SS,RT,Label)
/* X ? Agents,
M ? Messages,
M_list ? Messages,
RND_H ? Nonce_seq,
RND_S ? Sessionkeys,
HP ? Keys,
SS ? Keys */
(X,S,M_list) << Network,
$[new\_session\_key, H, RND\_S]$? S_State,
$[H, RND\_H]$? S_State,
(S,'priv',SS) ? S_Asymkeys,
(H,'pub',HP) ? S_Asymkeys,
M := elem(1,M_list),
elem(2,crypt(HP,M)) = RND_H,
Label << new_label,
RT := head(Label),
tail(Label) >> new_label,
$[H, RT]$ >> S_State,
$[new\_session\_key, H, RND\_S]$ << S_State,
$[session\_key, H, RND\_S]$ >> S_State,
/*($\forall P \in Agents \backslash \{S, H\} : not\_knows\{P, RND\_S\}$) >> S_Goal,
($\forall P \in Agents \backslash \{S, H\} : not\_knows\{P, RT\}$) >> S_Goal,*/
(S,conf,RT) >> Goals,
(S,conf,RND_S) >> Goals,
(S,H,[encrypt(RND_S,[RT])]) >> Network;

/* 6. Protokoll-Schritt: $H \longrightarrow F : RND$ */
def_trans_pattern H step_6
(X,M,RND_S,RT,RND,Nonce)
/* X ? Agents,
M ? Messages,

```
RND_S ? Sessionkey,
RT ? Label,
RND ? Nonce_seq */
(X,H,M) << Network,
[session_key, S, RND_S] ? H_State,
RT := elem(1,decrypt(RND_S,elem(1,M))),
[S, RT] >> H_State,
/*(∀P ∈ Agents\{S, H} : not_knows{P, RT}) >> H_Goal,*/
(H,conf,RT) >> Goals,
['start', F] ? H_State,
[F,' agent'] ? H_State,
['start', F] << H_State,
Nonce << new_nonce,
RND := head(Nonce),
tail(Nonce) >> new_nonce,
[F, RND] >> H_State,
(H,F,[RND]) >> Network;

/* 7. Protokoll-Schritt: F ⟶ H : SF(RND, MD) */
def_trans_pattern F step_7a
(X,M,RND,MD,SF,Label)
/* X ? Agents,
M ? Messages,
RND ? Nonces,
SF ? Keys */
(X,F,M) << Network,
['respond', H] ? F_State,
[H,' agent'] ? F_State,
[H,' sym', SF] ? F_Symkeys,
['respond', H] << F_State,
Label << new_label,
[MD] := head(Label),
tail(Label) >> new_label,
RND := elem(1,M),
(F,H,[encrypt(SF,[RND,MD])]) >> Network;

defcase


        check_step7: pro(nat_0,nat_0) >> boolean
        check_step7(MD,RT) = if MD=RT then 'true' else 'false';



def_trans_pattern H step_7b
(X,M1,M2,M_list,SF,RND,MD,S,RT)
/* X ? Agents,
M1 ? Messages,
M2 ? Messages,
M_list ? Messages,
SF ? Keys */
(X,H,M_list) << Network,
[F, RND] ? H_State,
(F,'sym',SF) ? H_Symkeys,
M1 := elem(1,M_list),
M2 := decrypt(SF,M1),
MD := elem(2,M2),
[S, RT] ? H_State,
check_step7(MD,RT) >> H_State;
```

B.2 Ausgabe des SHVT für die Untersuchung des APA ohne Angreifer

```
 M-1
 S_step_1 M-2
F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[respond,S].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[start,H]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[13,17,19,23]>


 M-1 S_step_1
 M-2
 H_step_2 M-3
F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[respond,S].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network: 1<(S,H,[S])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[13,17,19,23]>


 M-2 H_step_2
 M-3
 S_step_3 M-4
F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network: 1<(H,S,[13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1)])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]]>
```

```
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[17,19,23]>


 M-3 S_step_3
 M-4
 H_step_4 M-5
F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network: 1<(S,H,[[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),
    [(sessionkey,sym,17)]],13]]])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[new_session_key,H,
    (sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[19,23]>


 M-4 H_step_4
 M-5
 S_step_5 M-6
F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Goals: 1<(H,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].
[session_key,S,(sessionkey,sym,17)].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network: 1<(H,S,[[crypt,(H,priv,1),[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13]]])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[new_session_key,H,
    (sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[19,23]>


 M-5 S_step_5
 M-6
 H_step_6 M-7
F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Goals: 1<(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].
[session_key,S,(sessionkey,sym,17)].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
```

```
 Network: 1<(S,H,[[encrypt,(sessionkey,sym,17),[1234]]])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[session_key,H,
    (sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234]>
 new_nonce: 1<[19,23]>


 M-6 H_step_6
 M-7
 F_step_7a M-8
F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].
    [session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network: 1<(H,F,[19])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[session_key,H,
    (sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234]>
 new_nonce: 1<[23]>


 M-7 F_step_7a
 M-8
 H_step_7b M-10+  H_step_7b M-10+  H_step_7b M-9+  H_step_7b M-9+
F_Asymkeys: 1<::>
 F_State: 1<[H,agent]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].
    [session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network: 1<(F,H,[[encrypt,((H,F),sym,1),[19,1234]]])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[session_key,H,
    (sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[]>
 new_nonce: 1<[23]>


 M-8 H_step_7b
 M-8 H_step_7b
 M-9+
+++ dead +++
```

```
F_Asymkeys: 1<::>
 F_State: 1<[H,agent]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<true.[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),
[(H,pub,1)]]].[session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].
[session_key,H,(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[]>
 new_nonce: 1<[23]>


 M-8 H_step_7b
 M-8 H_step_7b
 M-10+
+++ dead +++

F_Asymkeys: 1<::>
 F_State: 1<[H,agent]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<false.[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),
[(H,pub,1)]]].[session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].
[session_key,H,(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[]>
 new_nonce: 1<[23]>
```

B.3 Ausgabe des SHVT für die Untersuchung des APA mit passivem Angreifer

```
M-1
 S_step_1 M-2
Charly_State: 1<[[],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[respond,S].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[start,H]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[13,17,19,23]>


 M-1 S_step_1
 M-2
 Charly_read M-3
Charly_State: 1<[[],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[respond,S].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<(S,H,[S])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[13,17,19,23]>


 M-2 Charly_read
 M-3
 Charly_repeat_last M-4
Charly_State: 1<[S,H,[S],last_message].[[S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
```

```
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[respond,S].
[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[13,17,19,23]>


 M-3 Charly_repeat_last
 M-4
 H_step_2 M-5
Charly_State: 1<[[S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[respond,S].
[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<(S,H,[S])>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[13,17,19,23]>


 M-4 H_step_2
 M-5
 Charly_read M-6
Charly_State: 1<[[S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<(H,S,[13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1)])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]]>
```

```
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[17,19,23]>


 M-5 Charly_read
 M-6
 Charly_repeat_last M-7
Charly_State: 1<[H,S,[13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1)],last_message].[[13,[crypt,(R,priv,1),
    [(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,(H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[start,F]>
 H_Symkeys: 1<(F,sym,(H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[17,19,23]>


 M-6 Charly_repeat_last
 M-7
 S_step_3 M-8
Charly_State: 1<[[13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,(H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[start,F]>
 H_Symkeys: 1<(F,sym,(H,F),sym,1))>
 Network_rec: 1<(H,S,[13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1)])>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[17,19,23]>


 M-7 S_step_3
 M-8
 Charly_read M-9
Charly_State: 1<[[13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
```

```
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<(S,H,[[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),
[(sessionkey,sym,17)]],13]])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[new_session_key,H,
(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[19,23]>


 M-8 Charly_read
 M-9
 Charly_repeat_last M-10
Charly_State: 1<[S,H,[[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),
    [(sessionkey,sym,17)]],13]]],last_message].[[[crypt,(R,priv,1),[(S,pub,1)]],
    (S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],13,
    [crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[new_session_key,H,
(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[19,23]>


 M-9 Charly_repeat_last
 M-10
 H_step_4 M-11
Charly_State: 1<[[[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),
[(sessionkey,sym,17)]],13]],13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<::>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[start,F]>
```

```
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<(S,H,[[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),
   [(sessionkey,sym,17)]],13]]])>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[new_session_key,H,
(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[19,23]>


 M-10 H_step_4
 M-11
 Charly_read M-12
Charly_State: 1<[[[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),
[(sessionkey,sym,17)]],13]],13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[session_key,S,
(sessionkey,sym,17)].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<(H,S,[[crypt,(H,priv,1),[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13]]])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[new_session_key,H,
(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[19,23]>


 M-11 Charly_read
 M-12
 Charly_repeat_last M-13
Charly_State: 1<[H,S,[[crypt,(H,priv,1),[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13]]],last_message].
    [[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,[crypt,(H,priv,1),[[crypt,(S,pub,1),
    [(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),
    [[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],13,[crypt,(R,priv,1),[(H,pub,1)]],
    (H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[session_key,S,
(sessionkey,sym,17)].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<::>
```

```
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[new_session_key,H,
(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[19,23]>


 M-12 Charly_repeat_last
 M-13
 S_step_5 M-14
Charly_State: 1<[[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,[crypt,(H,priv,1),[[crypt,(S,pub,1),
    [(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,
    (S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],13,[crypt,(R,priv,1),
    [(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].
[session_key,S,(sessionkey,sym,17)].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<(H,S,[[crypt,(H,priv,1),[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13]]])>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[new_session_key,H,
(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234,1234]>
 new_nonce: 1<[19,23]>


 M-13 S_step_5
 M-14
 Charly_read M-15
Charly_State: 1<[[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,[crypt,(H,priv,1),[[crypt,(S,pub,1),
    [(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,
    (S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],13,[crypt,(R,priv,1),
    [(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].
[session_key,S,(sessionkey,sym,17)].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<(S,H,[[encrypt,(sessionkey,sym,17),[1234]]])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
```

```
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].
[session_key,H,(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234]>
 new_nonce: 1<[19,23]>


 M-14 Charly_read
 M-15
 Charly_repeat_last M-16
Charly_State: 1<[S,H,[[encrypt,(sessionkey,sym,17),[1234]]],last_message].[[[encrypt,(sessionkey,sym,17),
    [1234]],[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,[crypt,(H,priv,1),[[crypt,(S,pub,1),
    [(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),
    [[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],
attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[session_key,S,
(sessionkey,sym,17)].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[session_key,H,
(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234]>
 new_nonce: 1<[19,23]>


 M-15 Charly_repeat_last
 M-16
 H_step_6 M-17
Charly_State: 1<[[[encrypt,(sessionkey,sym,17),[1234]],[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,
    [crypt,(H,priv,1),[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),
    [(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],
13]],13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,agent].[R,server].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].[session_key,S,
(sessionkey,sym,17)].[start,F]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<(S,H,[[encrypt,(sessionkey,sym,17),[1234]]])>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[session_key,H,
```

```
(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234]>
 new_nonce: 1<[19,23]>


 M-16 H_step_6
 M-17
 Charly_read M-18
Charly_State: 1<[[[encrypt,(sessionkey,sym,17),[1234]],[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,
    [crypt,(H,priv,1),[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),
    [(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],
13]],13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),
[(H,pub,1)]]].[session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<(H,F,[19])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].
[session_key,H,(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234]>
 new_nonce: 1<[23]>


 M-17 Charly_read
 M-18
 Charly_repeat_last M-19
Charly_State: 1<[H,F,[19],last_message].[[19,[encrypt,(sessionkey,sym,17),[1234]],[crypt,(S,pub,1),
    [(sessionkey,sym,17)]],13,[crypt,(H,priv,1),[[crypt,(S,pub,1),[(sessionkey,sym,17)]],
13]],[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),
    [(sessionkey,sym,17)]],13]],13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),
[(H,pub,1)]]].[session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].
[session_key,H,(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
```

```
 new_label: 1<[1234]>
 new_nonce: 1<[23]>


 M-18 Charly_repeat_last
 M-19
 F_step_7a M-20
Charly_State: 1<[[19,[encrypt,(sessionkey,sym,17),[1234]],[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,
    [crypt,(H,priv,1),[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),
    [(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],
13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent].[respond,H]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].
[session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<(H,F,[19])>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[session_key,H,
(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[1234]>
 new_nonce: 1<[23]>


 M-19 F_step_7a
 M-20
 Charly_read M-21
Charly_State: 1<[[19,[encrypt,(sessionkey,sym,17),[1234]],[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,
    [crypt,(H,priv,1),[[crypt,(S,pub,1),[(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),
    [(S,pub,1)]],(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],
13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),[(H,pub,1)]]].
[session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<(F,H,[[encrypt,((H,F),sym,1),[19,1234]]])>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].[session_key,H,
(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[]>
 new_nonce: 1<[23]>
```

```
 M-20 Charly_read
 M-21
 Charly_repeat_last M-22
Charly_State: 1<[F,H,[[encrypt,((H,F),sym,1),[19,1234]]],last_message].[[[encrypt,((H,F),sym,1),
    [19,1234]],19,[encrypt,(sessionkey,sym,17),[1234]],[crypt,(S,pub,1),
    [(sessionkey,sym,17)]],13,[crypt,(H,priv,1),[[crypt,(S,pub,1),
    [(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),[(S,pub,1)]],(S,pub,1),
    [crypt,(S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],13,
    [crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),
[(H,pub,1)]]].[session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].
[session_key,H,(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[]>
 new_nonce: 1<[23]>

 M-21 Charly_repeat_last
 M-22
 H_step_7b M-24+  H_step_7b M-24+  H_step_7b M-23+  H_step_7b M-23+
Charly_State: 1<[[[encrypt,((H,F),sym,1),[19,1234]],19,[encrypt,(sessionkey,sym,17),[1234]],
[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,[crypt,(H,priv,1),[[crypt,
(S,pub,1),[(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),[(S,pub,1)]],
(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],
13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),
[(H,pub,1)]]].[session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<(F,H,[[encrypt,((H,F),sym,1),[19,1234]]])>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].
[session_key,H,(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[]>
 new_nonce: 1<[23]>
```

```
 M-22 H_step_7b
 M-22 H_step_7b
 M-23+
+++ dead +++

Charly_State: 1<[[[encrypt,((H,F),sym,1),[19,1234]],19,[encrypt,(sessionkey,sym,17),[1234]],
[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,[crypt,(H,priv,1),[[crypt,
(S,pub,1),[(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),[(S,pub,1)]],
(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],
13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<false.[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),
[(H,pub,1)]]].[session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].
[session_key,H,(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[]>
 new_nonce: 1<[23]>

 M-22 H_step_7b
 M-22 H_step_7b
 M-24+
+++ dead +++

Charly_State: 1<[[[encrypt,((H,F),sym,1),[19,1234]],19,[encrypt,(sessionkey,sym,17),[1234]],
[crypt,(S,pub,1),[(sessionkey,sym,17)]],13,[crypt,(H,priv,1),[[crypt,
(S,pub,1),[(sessionkey,sym,17)]],13]],[crypt,(R,priv,1),[(S,pub,1)]],
(S,pub,1),[crypt,(S,priv,1),[[crypt,(H,pub,1),[(sessionkey,sym,17)]],13]],
13,[crypt,(R,priv,1),[(H,pub,1)]],(H,pub,1),S],attack_data]>
 Charly_Structure: 1<::>
 F_Asymkeys: 1<::>
 F_State: 1<[H,agent]>
 F_Symkeys: 1<(H,sym,((H,F),sym,1))>
 Global: 1<[start_attack]>
 Goals: 1<(H,conf,1234).(H,conf,(sessionkey,sym,17)).(S,conf,1234).(S,conf,(sessionkey,sym,17))>
 H_Asymkeys: 1<(H,priv,(H,priv,1)).(H,pub,(H,pub,1)).(R,pub,(R,pub,1))>
 H_State: 1<true.[13,S].[F,19].[F,agent].[R,server].[S,1234].[S,agent].[Zertifikat,[crypt,(R,priv,1),
[(H,pub,1)]]].[session_key,S,(sessionkey,sym,17)]>
 H_Symkeys: 1<(F,sym,((H,F),sym,1))>
 Network_rec: 1<::>
 Network_send: 1<::>
 R_Asymkeys: 1<(R,priv,(R,priv,1)).(R,pub,(R,pub,1))>
 R_State: 1<::>
 R_Symkeys: 1<::>
 S_Asymkeys: 1<(H,pub,(H,pub,1)).(R,pub,(R,pub,1)).(S,priv,(S,priv,1)).(S,pub,(S,pub,1))>
 S_State: 1<[H,13].[H,1234].[H,agent].[R,server].[Zertifikat,[crypt,(R,priv,1),[(S,pub,1)]]].
[session_key,H,(sessionkey,sym,17)]>
 S_Symkeys: 1<::>
 new_key: 1<[111,222,333,444,555]>
 new_label: 1<[]>
 new_nonce: 1<[23]>
```

B.4 Spezifikation der Angreifer

B.4.1 Spezifikation des passiven Angreifers

```
/* Generischer Angreifer. Alle Daten werden gelernt und soweit moeglich entschluesselt.
Nachrichten werden entsprechend der vorgegebenen Struktur konstruiert */


/* include "functions.vsp"; */

/* Mengen zur Spezifikation der Struktur einer Nachricht */

defset Structure_Types = {atom,agents,label,nat,nonce,constants,keys,funcs};

defset Structure_M = Message || Structure_Types;

defset Structure_list = [Structure_M];

defset Structure_Message = pro(nat_0,Agents,Agents,Structure_list);

defset Structure_Message_seq = seq(Structure_Message);


def_role C; /* malicious agent (attacker) */

def_role A;  /* Initiator of the protocol */

def_role B;  /* Responder */

/* def_state C State: Messages_seq := [[(sessionkey,sym,111)],'attack_data'];*/
def_state C State: Messages_seq := [[],'attack_data'];


def_state C Structure: Structure_Message_seq :=
/* (1,S,H,[S]). */
/* (4,H,S,[atom,atom,keys,H]). */
/* (3,S,H,[atom,keys,crypt(keys,[crypt(keys,[atom]),atom,'Host'])]). */
(1,H,S,[crypt(keys,[crypt(keys,[atom]),atom,'SMC'])]).
(1,S,H,[encrypt(atom,[atom])]).
(1,H,F,[atom]).
(1,F,H,[encrypt(keys,[atom,atom])]);




/* def_state Network: net_elem_seq := ::; */
def_state Network_rec: net_elem_seq := ::;
def_state Network_send: net_elem_seq := ::;

def_state Global: Messages_seq := ['start_attack'];


defrecl not_in: pro(Message_list,Message) >> boolean
        not_in([],a) = 'true',
        not_in(cons(m,rest),a) = if m=a then 'false' else not_in(rest,a)
                                 if get_f_type(m)~='no_crypt_function' & m~=a then not_in(rest,a)
   else and(not_in(m,a),not_in(rest,a));

defrecl list_not_in: pro(Message_list,Message_list) >> Message_list
        list_not_in([],a) = a,
        list_not_in(cons(m,rest),a) = if m=a then [] else list_not_in(rest,a)
                                      if m=a then [] else list_not_in(rest,a);
```

```
defrecl learn_new: pro(Message_list,Message_list) >> Message_list
learn_new([],m) = [],
learn_new(cons(a,rest),m) = if not_in(m,a)='true' then pushnew(a,learn_new(rest,m)) else learn_new(rest,m)
                        if get_f_type(a)='no_crypt_function' then
                            append(learn_new(a,m),learn_new(rest,m))
                        else  cons(a,learn_new(rest,m));


defrecl try_to_decrypt: pro(Message_list,Message_list) >> Message_list
        try_to_decrypt([],ciphertext)=[],
        try_to_decrypt(cons(key,keylist),ciphertext)= if get_key(ciphertext)=inverse(key)
 then get_plaintext(ciphertext)
                                                        else try_to_decrypt(keylist,ciphertext)
                                                    if get_key(ciphertext)=inverse(key)
 then get_plaintext(ciphertext)
                                                        else try_to_decrypt(keylist,ciphertext);

defrecl try_to_decrypt_list: pro(Message_list,Message_list) >> Message_list
        try_to_decrypt_list([],keylist)=[],
        try_to_decrypt_list(cons(first,rest),keylist) = []
                                                         if get_f_type(first) ? Cryptfuncs then
                                                           append(try_to_decrypt(keylist,first),
 try_to_decrypt_list(rest,keylist))
                                                         else   try_to_decrypt_list(rest,keylist);


defrecn mutual_decrypt: pro(nat_0,Message_list,Message_list) >> Message_list
        mutual_decrypt(0,data1,data2) = append(data1,data2),
        mutual_decrypt(n+1,data1,data2) = if get_f_type(data2)~=no_crypt_function &
try_to_decrypt_list(data2,data1)=[]
then append(data1,data2)
                                          else mutual_decrypt(n,append(try_to_decrypt_list(data2,data1),
       data2),data1);


defcase learn: pro(Message_list,Message_list) >> Message_list
        learn(old_data,new_data)= if get_f_type(new_data)=no_crypt_function
       then mutual_decrypt(1000,new_data,old_data)
                                      else mutual_decrypt(1000,old_data,new_data);


/* Functions for the construction of messages */

defrecl message_count : Structure_list >> nat_0
        message_count([])=0,
        message_count(cons(first,rest))= if first ? Constants| first ? Funcs
                                            then message_count(rest)
                                       else 1+message_count(rest),
                                       message_count(first)+message_count(rest);

defcase struct_to_mesg: Structure_M >> Message
        struct_to_mesg(M) = M else_error 'undefined';



defcase correct_type: pro(Structure_M,Messages) >> boolean
        correct_type(type,M) = if type='atom' |
                                (type='agents' & M ? Agents) |
                                (type='label' & M ? Label)|
                                (type='nat' & M ? nat_0) |
                                (type='nonce' & M ? Nonce) |
                                (type='constants' & M ? Constants) |
                                (type='keys' & M ?  Keys) |
```

```
                                (type='funcs' & M ?   Funcs) |
                                type = M
                              then 'true'
                              else 'false';


defcase msgseq_to_msg: Messages_seq >> Messages
        msgseq_to_msg(M) = M else_error 'undefined';

defcase msgseq_to_msglist: Messages_seq >> Message_list
        msgseq_to_msglist(M) = M else_error ['undefined'];

defcase gnull : nat_0 >> nat_1
        gnull(X) = if X>0 then X else_error 1;

defcase get_key_struct: Structure_list >> Messages
        get_key_struct(message) = if head(message) ? Cryptfuncs
                        then head(tail(message))
                        else 'access_error';

defcase correct_key: pro(Structure_list,Messages) >> boolean
        correct_key(Struct,M) = if get_key_struct(Struct)=get_key(M) then 'true' else 'false';

defrecl construct_message: pro(Structure_list,Messages_seq) >> Message_list
        construct_message([],M)=[],
        construct_message(cons(head,tail),M) = if (head ? Constants | head ? Funcs) &
   get_f_type(seg(M,1,1))= 'no_crypt_function'
                                               then cons(struct_to_mesg(head),construct_message(tail,M))
                                               if head ? Funcs & head(seg(M,1,1))=head &
   correct_key(cons(head,tail),seg(M,1,1))='true'
                                               then msgseq_to_msglist(seg(M,1,1))
                                               if correct_type(head,seg(M,1,1))='true'
                                               then cons(msgseq_to_msg(seg(M,1,1)),
     construct_message(tail,seg(M,2,l(M))))
                                               else ['undefined'],
                                               cons(construct_message(head,seg(M,1,message_count(head))),
                                                    construct_message(tail,seg(M,gnull(message_count(head)
+1),l(M))));


defrecl undef: Message_list >> boolean
        undef([])=false,
        undef(cons(head,tail)) = if head='undefined' then 'true' else undef(tail),
                                 or(undef(head),undef(tail));


defrecs delete_last: Messages_seq >> Messages_seq
        delete_last(::)=::,
        delete_last(M.M1) = if elem(4,M1)='last_message' then delete_last(M) else delete_last(M).M1;

/* def_trans_pattern C forward
(M,A,B,d)
(A,B,M) << Network_send,
[d,'attack_data'] << C_State,
[learn(d,learn_new(M,d)),'attack_data'] >> C_State,
(A,B,M) >> Network_rec; */

def_trans_pattern C read
(M,A,B,d)
(A,B,M) << Network_send,
C_State:= delete_last(C_State),
[d,'attack_data'] << C_State,
[learn(d,learn_new(M,d)),'attack_data'] >> C_State,
[A,B,M,'last_message'] >> C_State;
```

```
def_trans_pattern C repeat_last
(P,Q,M)
Network_rec=::,
[P,Q,M,'last_message'] << C_State,
(P,Q,M) >> Network_rec;

def_trans_pattern C send_generic1
  (datalist,P,Q,M1,M,Struct)
  ['start_attack'] ? Global,
  [datalist,'attack_data'] ? C_State,
  Network_rec=::,
  (1,P,Q,Struct) ? C_Structure,
  M1:=list_to_seq(datalist),
  M:=construct_message(Struct,M1),
  undef(M)='false',
  l(sdelete([P,Q,M],C_State))=l(C_State),
  (P,Q,M) >> Network_rec,
  [P,Q,M] >> C_State ;

def_trans_pattern C send_generic2
  (datalist,P,Q,M1,M2,M,Struct)
  ['start_attack'] ? Global,
  [datalist,'attack_data'] ? C_State,
  Network_rec=::,
  (2,P,Q,Struct) ? C_Structure,
  M1:=list_to_seq(datalist),
  M2:=list_to_seq(datalist),
  M:=construct_message(Struct,M1.M2),
  undef(M)='false',
  l(sdelete([P,Q,M],C_State))=l(C_State),
  (P,Q,M) >> Network_rec,
  [P,Q,M] >> C_State ;


def_trans_pattern C send_generic3
  (datalist,P,Q,M1,M2,M3,M,Struct)
  ['start_attack'] ? Global,
  [datalist,'attack_data'] ? C_State,
  Network_rec=::,
  (3,P,Q,Struct) ? C_Structure,
  M1:=list_to_seq(datalist),
  M2:=list_to_seq(datalist),
  M3:=list_to_seq(datalist),
  M:=construct_message(Struct,M1.M2.M3),
  undef(M)='false',
  l(sdelete([P,Q,M],C_State))=l(C_State),
  (P,Q,M) >> Network_rec,
  [P,Q,M] >> C_State ;

def_trans_pattern C send_generic4
  (datalist,P,Q,M1,M2,M3,M4,M,Struct)
['start_attack'] ? Global,
  [datalist,'attack_data'] ? C_State,
  Network_rec=::,
  (4,P,Q,Struct) ? C_Structure,
  M1:=list_to_seq(datalist),
  M2:=list_to_seq(datalist),
  M3:=list_to_seq(datalist),
  M4:=list_to_seq(datalist),
  M:=construct_message(Struct,M1.M2.M3.M4),
  undef(M)='false',
  l(sdelete([P,Q,M],C_State))=l(C_State),
  (P,Q,M) >> Network_rec,
  [P,Q,M] >> C_State ;
```

```
def_trans_pattern C send_generic5
  (datalist,P,Q,M1,M2,M3,M4,M5,M,Struct)
['start_attack'] ? Global,
  [datalist,'attack_data'] ? C_State,
  Network_rec=::,
  (5,P,Q,Struct) ? C_Structure,
  M1:=list_to_seq(datalist),
  M2:=list_to_seq(datalist),
  M3:=list_to_seq(datalist),
  M4:=list_to_seq(datalist),
  M5:=list_to_seq(datalist),
  M:=construct_message(Struct,M1.M2.M3.M4.M5),
  undef(M)='false',
  1(sdelete([P,Q,M],C_State))=1(C_State),
  (P,Q,M) >> Network_rec,
  [P,Q,M] >> C_State ;

def_pattern_bind C := {'Charly' };
def_pattern_bind A := { 'Alice' };

def_pattern_bind B := { 'Bob', 'Alice' };
```

B.4.2 Spezifikation des generischen Angreifers

```
/* Generischer Angreifer. Alle Daten werden gelernt und soweit moeglich entschluesselt.
Nachrichten werden entsprechend der vorgegebenen Struktur konstruiert */


/* include "functions.vsp"; */

/* Mengen zur Spezifikation der Struktur einer Nachricht */

defset Structure_Types = {atom,agents,label,nat,nonce,constants,keys,funcs};

defset Structure_M = Message || Structure_Types;

defset Structure_list = [Structure_M];

defset Structure_Message = pro(nat_0,Agents,Agents,Structure_list);

defset Structure_Message_seq = seq(Structure_Message);


def_role C; /* malicious agent (attacker) */

def_role A;  /* Initiator of the protocol */

def_role B;  /* Responder */

/* def_state C State: Messages_seq := [[(sessionkey,sym,111)],'attack_data'];*/
def_state C State: Messages_seq := [[],'attack_data'];


def_state C Structure: Structure_Message_seq :=
/* (1,S,H,[S]). */
/* (4,H,S,[atom,atom,keys,H]). */
/* (3,S,H,[atom,keys,crypt(keys,[crypt(keys,[atom]),atom,'Host'])]). */
(1,H,S,[crypt(keys,[crypt(keys,[atom]),atom,'SMC'])]).
(1,S,H,[encrypt(atom,[atom])]).
(1,H,F,[atom]).
(1,F,H,[encrypt(keys,[atom,atom])]);
```

```
/* def_state Network: net_elem_seq := ::; */
def_state Network_rec: net_elem_seq := ::;
def_state Network_send: net_elem_seq := ::;

def_state Global: Messages_seq := ['start_attack'];


defrecl not_in: pro(Message_list,Message) >> boolean
        not_in([],a) = 'true',
        not_in(cons(m,rest),a) = if m=a then 'false' else not_in(rest,a)
                                  if get_f_type(m)~='no_crypt_function' & m~=a then not_in(rest,a) else
                                 and(not_in(m,a),not_in(rest,a));

defrecl list_not_in: pro(Message_list,Message_list) >> Message_list
        list_not_in([],a) = a,
        list_not_in(cons(m,rest),a) = if m=a then [] else list_not_in(rest,a)
                                  if m=a then [] else list_not_in(rest,a);



defrecl learn_new: pro(Message_list,Message_list) >> Message_list
learn_new([],m) = [],
learn_new(cons(a,rest),m) = if not_in(m,a)='true' then pushnew(a,learn_new(rest,m)) else learn_new(rest,m)
                        if get_f_type(a)='no_crypt_function' then
                            append(learn_new(a,m),learn_new(rest,m))
                        else  cons(a,learn_new(rest,m));


defrecl try_to_decrypt: pro(Message_list,Message_list) >> Message_list
        try_to_decrypt([],ciphertext)=[],
        try_to_decrypt(cons(key,keylist),ciphertext)= if get_key(ciphertext)=inverse(key)
 then get_plaintext(ciphertext)
                                                        else try_to_decrypt(keylist,ciphertext)
                                                    if get_key(ciphertext)=inverse(key)
 then get_plaintext(ciphertext)
                                                        else try_to_decrypt(keylist,ciphertext);

defrecl try_to_decrypt_list: pro(Message_list,Message_list) >> Message_list
        try_to_decrypt_list([],keylist)=[],
        try_to_decrypt_list(cons(first,rest),keylist) = []
                                                      if get_f_type(first) ? Cryptfuncs then
                                                        append(try_to_decrypt(keylist,first),
 try_to_decrypt_list(rest,keylist))
                                                      else   try_to_decrypt_list(rest,keylist);


defrecn mutual_decrypt: pro(nat_0,Message_list,Message_list) >> Message_list
        mutual_decrypt(0,data1,data2) = append(data1,data2),
        mutual_decrypt(n+1,data1,data2) = if get_f_type(data2)~=no_crypt_function &
try_to_decrypt_list(data2,data1)=[]
then append(data1,data2)
                                        else mutual_decrypt(n,append(try_to_decrypt_list(data2,data1),
       data2),data1);


defcase learn: pro(Message_list,Message_list) >> Message_list
        learn(old_data,new_data)= if get_f_type(new_data)=no_crypt_function
       then mutual_decrypt(1000,new_data,old_data)
                                    else mutual_decrypt(1000,old_data,new_data);


/* Functions for the construction of messages */
```

```
defrecl message_count : Structure_list >> nat_0
        message_count([])=0,
        message_count(cons(first,rest))= if first ? Constants| first ? Funcs
                                            then message_count(rest)
                                         else 1+message_count(rest),
                                         message_count(first)+message_count(rest);

defcase struct_to_mesg: Structure_M >> Message
        struct_to_mesg(M) = M else_error 'undefined';



defcase correct_type: pro(Structure_M,Messages) >> boolean
        correct_type(type,M) = if type='atom' |
                                  (type='agents' & M ? Agents) |
                                  (type='label' & M ? Label)|
                                  (type='nat' & M ? nat_0) |
                                  (type='nonce' & M ? Nonce) |
                                  (type='constants' & M ? Constants) |
                                  (type='keys' & M ?  Keys) |
                                  (type='funcs' & M ?   Funcs) |
                                  type = M
                                then 'true'
                                else 'false';


defcase msgseq_to_msg: Messages_seq >> Messages
        msgseq_to_msg(M) = M else_error 'undefined';

defcase msgseq_to_msglist: Messages_seq >> Message_list
        msgseq_to_msglist(M) = M else_error ['undefined'];

defcase gnull : nat_0 >> nat_1
        gnull(X) = if X>0 then X else_error 1;

defcase get_key_struct: Structure_list >> Messages
        get_key_struct(message) = if head(message) ? Cryptfuncs
                        then head(tail(message))
                        else 'access_error';

defcase correct_key: pro(Structure_list,Messages) >> boolean
        correct_key(Struct,M) = if get_key_struct(Struct)=get_key(M) then 'true' else 'false';

defrecl construct_message: pro(Structure_list,Messages_seq) >> Message_list
        construct_message([],M)=[],
        construct_message(cons(head,tail),M) = if (head ? Constants | head ? Funcs) &
   get_f_type(seg(M,1,1))= 'no_crypt_function'
                                               then cons(struct_to_mesg(head),construct_message(tail,M))
                                               if head ? Funcs & head(seg(M,1,1))=head &
         correct_key(cons(head,tail),seg(M,1,1))='true'
                                               then msgseq_to_msglist(seg(M,1,1))
                                               if correct_type(head,seg(M,1,1))='true'
                                               then cons(msgseq_to_msg(seg(M,1,1)),
     construct_message(tail,seg(M,2,l(M))))
                                               else ['undefined'],
                                               cons(construct_message(head,seg(M,1,message_count(head))),
                                                    construct_message(tail,seg(M,gnull(message_count(head)
+1),l(M))));


defrecl undef: Message_list >> boolean
        undef([])=false,
        undef(cons(head,tail)) = if head='undefined' then 'true' else undef(tail),
                                 or(undef(head),undef(tail));
```

```
defrecs delete_last: Messages_seq >> Messages_seq
        delete_last(::)=::,
        delete_last(M.M1) = if elem(4,M1)='last_message' then delete_last(M) else delete_last(M).M1;

/* def_trans_pattern C forward
(M,A,B,d)
(A,B,M) << Network_send,
[d,'attack_data'] << C_State,
[learn(d,learn_new(M,d)),'attack_data'] >> C_State,
(A,B,M) >> Network_rec; */

def_trans_pattern C read
(M,A,B,d)
(A,B,M) << Network_send,
C_State:= delete_last(C_State),
[d,'attack_data'] << C_State,
[learn(d,learn_new(M,d)),'attack_data'] >> C_State,
[A,B,M,'last_message'] >> C_State;

def_trans_pattern C repeat_last
(P,Q,M)
Network_rec=::,
[P,Q,M,'last_message'] << C_State,
(P,Q,M) >> Network_rec;

def_trans_pattern C send_generic1
  (datalist,P,Q,M1,M,Struct)
  ['start_attack'] ? Global,
  [datalist,'attack_data'] ? C_State,
  Network_rec=::,
  (1,P,Q,Struct) ? C_Structure,
  M1:=list_to_seq(datalist),
  M:=construct_message(Struct,M1),
  undef(M)='false',
  l(sdelete([P,Q,M],C_State))=l(C_State),
  (P,Q,M) >> Network_rec,
  [P,Q,M] >> C_State ;

def_trans_pattern C send_generic2
  (datalist,P,Q,M1,M2,M,Struct)
  ['start_attack'] ? Global,
  [datalist,'attack_data'] ? C_State,
  Network_rec=::,
  (2,P,Q,Struct) ? C_Structure,
  M1:=list_to_seq(datalist),
  M2:=list_to_seq(datalist),
  M:=construct_message(Struct,M1.M2),
  undef(M)='false',
  l(sdelete([P,Q,M],C_State))=l(C_State),
  (P,Q,M) >> Network_rec,
  [P,Q,M] >> C_State ;


def_trans_pattern C send_generic3
  (datalist,P,Q,M1,M2,M3,M,Struct)
  ['start_attack'] ? Global,
  [datalist,'attack_data'] ? C_State,
  Network_rec=::,
  (3,P,Q,Struct) ? C_Structure,
  M1:=list_to_seq(datalist),
  M2:=list_to_seq(datalist),
  M3:=list_to_seq(datalist),
  M:=construct_message(Struct,M1.M2.M3),
  undef(M)='false',
```

```
  l(sdelete([P,Q,M],C_State))=l(C_State),
  (P,Q,M) >> Network_rec,
  [P,Q,M] >> C_State ;

def_trans_pattern C send_generic4
  (datalist,P,Q,M1,M2,M3,M4,M,Struct)
['start_attack'] ? Global,
  [datalist,'attack_data'] ? C_State,
  Network_rec=::,
  (4,P,Q,Struct) ? C_Structure,
  M1:=list_to_seq(datalist),
  M2:=list_to_seq(datalist),
  M3:=list_to_seq(datalist),
  M4:=list_to_seq(datalist),
  M:=construct_message(Struct,M1.M2.M3.M4),
  undef(M)='false',
  l(sdelete([P,Q,M],C_State))=l(C_State),
  (P,Q,M) >> Network_rec,
  [P,Q,M] >> C_State ;

def_trans_pattern C send_generic5
  (datalist,P,Q,M1,M2,M3,M4,M5,M,Struct)
['start_attack'] ? Global,
  [datalist,'attack_data'] ? C_State,
  Network_rec=::,
  (5,P,Q,Struct) ? C_Structure,
  M1:=list_to_seq(datalist),
  M2:=list_to_seq(datalist),
  M3:=list_to_seq(datalist),
  M4:=list_to_seq(datalist),
  M5:=list_to_seq(datalist),
  M:=construct_message(Struct,M1.M2.M3.M4.M5),
  undef(M)='false',
  l(sdelete([P,Q,M],C_State))=l(C_State),
  (P,Q,M) >> Network_rec,
  [P,Q,M] >> C_State ;

def_pattern_bind C := {'Charly' };
def_pattern_bind A := { 'Alice' };

def_pattern_bind B := { 'Bob', 'Alice' };
```