Studienarbeit

Konzept und exemplarische Implementation eines gesicherten Kanals zur Übertragung biometrischer Daten

Willem Froehling

Matrikelnummer: 4608832

15. Januar 2003

Betreuer:

Prof. Dr. K. Brunnstein
Universität Hamburg
Fachbereich Informatik
Arbeitsbereich AGN

mit Unterstützung von
Dipl. Inf. A. Brömme
Universität Hamburg
Fachbereich Informatik
Arbeitsbereich AGN

Inhaltsverzeichnis

1 Einleitung 3

1.1 Biometrische Systeme 3

1.2 Notwendigkeit der Datenübermittlung 4

1.3 Überblick über die Studienarbeit 5

2 Grundlagen 6

2.1 Biometrische Verfahren 6

2.2 Notwendigkeit von Kommunikationsprotokollen 14

2.3 Netz Architektur 18

2.3.1 ISO/OSI: 18

2.3.2 TCP/IP: 20

2.4 Sicherheit 26

2.5 Grundlagen der Kryptographie 29

2.5.1 Grundlegende Begriffe 29

2.5.2 Ziele 30

2.5.3 Der Begriff der Chiffre 31

2.5.4 Symmetrische Chiffren 31

2.5.5 Asymmetrische Chiffren 38

2.5.6 Weitere notwendige Algorithmen 41

2.5.7 Weiteres wichtiges Zubehör 45

2.5.8 Kryptographische Protokolle 46

3 Konzept und Implementation 47

3.1 Voraussetzung und geforderte Funktionalität 49

3.1.1 Ressourcen der Erfassung von biologischen Charakteristika 49

3.1.2 Die Erfassungseinheit 51

3.1.3 Aufnahmeparametrisierung des Servers 52

3.1.4 Rollenbasierende Authentikation 54

3.1.5 Meta- und Kalibrierungsdaten 55

3.1.6 Authentikationsergebnisübertragung 56

3.2 Konzept der Implementation. 56

3.2.1 SSL/TLS 56

3.2.2 Das Biometrische Protokoll 73
3.3 Nachweis der Implementation 82
3.3.1 Aufbau der Clientapplikation 82
3.3.2 Aufbau der Serverapplikation 83
3.3.3 SSL-Trace 83

4 Zusammenfassung und Ausblick 86

1 Einleitung

In dieser Arbeit geht es um die Konzeption und exemplarische Implementation eines Kommunikationsprotokolls, für den Austausch von Daten zwischen mehreren Erfassungseinheiten (Clients) und einem Authentikationsserver (Server), zum Zwecke einer biometrischen Authentikation. Da die Daten, die bei einem solchen Vorgang ausgetauscht werden, bestimmten natürlichen Personen zugeordnet werden können und der Informationsgehalt der Daten weit mehr Rückschlüsse auf das Individium beinhalten kann, als dieses für den Prozess notwendig wäre, bestehen zusätzliche Anforderungen an die Kommunikation.

> „Zur Wahrung des informationellen Selbstbestimmungsrechts der Betroffenen sollten biometrische Verfahren eingesetzt werden, bei denen sich aus den Identifikationsdaten kein überschießender Informationsgehalt ergibt. Bedenken ruft hier z.B. ein Verfahren wie die Genomanalyse hervor. Auch dann, wenn sie nur zur Identifikation eingesetzt wird, ist es nicht auszuschließen, dass - unter Umständen auch durch zukünftige wissenschaftliche Erkenntnisse - aus den zu Zwecken der Identifikation gewonnenen Daten Rückschlüsse auf bestimmte Veranlagungen gezogen werden können.“[GK98]

Ausserdem sollte das zu erstellende Protokoll nicht nur die zu übertragenden Daten schützen, sondern auch zusätzliche Mechanismen beinhalten, die den Authentikationsprozess vor bekanten Angriffen absichert.

Eine weitergehende Forderung des zu entwickelnden Protokolls ist die Einsatzfähigkeit bei unterschiedlichen biometrischen Verfahren. Das Protokoll sollte nicht fest an ein biometrisches Verfahren gebunden sein, sondern mit den unterschiedlichesten Erfassungsmethoden und deren aufgezeichneten Daten umgehen können.

1.1 Biometrische Systeme

Der unauthorisierte Zugang zu Gebäuden oder Räumen, aber auch der unerlaubte Gebrauch von Resourcen wird in der Regel durch verschiedene Verfahren mehr oder minder gut gewährleistet. Bei einem Gebäude ist es häufig eine Tür mit einem Schloss. Auch elektronische Zugangskontrollsysteme mittels Nummerkombinationen oder Smartcards werden genutzt. Bei den Resourcen kann es sich aber auch um EDV-Systeme handeln. Hier möchte man meistens die Nutzung von Rechenzeit, wie auch die Belegung von Speicherplatz verbieten oder limitieren. Persönliche und vertrauliche Daten wollen vor nicht genehmigtem Zugriff geschützt werden.

In diesem Bereich scheint ein großes Potential für biometrische Systeme zu liegen. Seit dem 11. September 2001 erfuhr die Biometrik, dank tatkräftiger Unterstützung der Bundesregierung, einen rasanten Anstieg des Medieninteresses. Biometrische Systeme zur Authentisierung werben mit erhöhter Sicherheit, da es nicht nur darauf ankommt,

- Was man besitzt (z.B. Smartcard),
- Was man weiß (z.B. PIN),

sondern diese Systeme darauf basieren,

- Wer man ist oder besser welche Charakteristik man hat.

Es hat sich in den letzten Jahren gezeigt, dass der Mensch viele unterschiedliche Merkmale hat, die für biometrische Methoden genutzt werden können. Diese Verfahren zur Erfassung und Berechnung von biometrischen Merkmalen können sich im ablaufenden Prozess unterscheiden, wobei auch unterschiedliche Daten anfallen und verarbeitet werden müssen.

1.2 Notwendigkeit der Datenübermittlung

Da ein biometrisches System mit hoch sensiblen Daten arbeitet, ist es bei diesen Anwendungen ernorm wichtig, dass das Gesamtsystem als sicheres System ausgelegt ist. Sowohl bei einem *stand-alone* System, wie auch bei geographisch Verteilten Systemen müssen Daten übertragen werden. Daten werden zwischen Anwendungen/Tasks sowohl auf einem einzelnen Rechner ausgetauscht, aber auch die Kommunikation über Rechnernetze kann notwendig sein. Dabei ist es bei verläßlichen EDV-Systemen notwendig, sicher zu stellen, dass die Daten bei deren Übertragung nicht modifiziert werden, und somit die empfangene Instanz andere

Daten erhält, als die sendende Instanz übermitteln wollte. Bei sensiblen Daten ist es zudem wichtig, dass beide Kommunikationspartner feststellen können, ob die Gegenstelle wirklich die Instanz ist, mit der Daten ausgetauscht werden sollen. Auch sollte ein Dritter, der die Kommunikation zwischen den Stationen mitverfolgt, möglichst keine Informationen aus den zu übertragenden Daten ziehen können.

1.3 Überblick über die Studienarbeit

Zunächst werden die zum Verständnis notwendigen Grundlagen erläutert. Es werden notwendige Begriffe erklärt und Modelle aus unterschiedlichen Fachgebieten vorgestellt.

Das Grundlagenkapitel beschäftigt sich mit biometrischen Verfahren. Es beschreibt die zugrundeliegende Netzarchitektur TCP/IP im Vergleich zum ISO-OSI Modell und gibt einen Überblick zu den verwendeten Bereichen aus der Kryptographie. Anschliessend wird im nächsten Kapitel das Konzept eines sicheren Kanals für die Übertragung von biometrischen Daten beschrieben. Dieses Kapitel besteht aus zwei Teilen. Der erste Teil behandelt den sicheren Kanal in seiner Umsetzung durch das SSL/TLS-Protokoll. Der zweite Teil erklärt das neu entworfene Protokoll für einen biometrischen Authentikationsprozess mit Hilfe von Erfassungseinheiten und einem zentralen Server.

Eine zusätzliche Aufgabe dieser Studienarbeit ist es, noch eine lauffähige Version des Protokolls in Form eines Prototypen zu programmieren. Es wird kurz auf das Programm eingegangen. Die Studienarbeit endet dann mit einer Zusammenfassung und einem Ausblick.

2 Grundlagen

In diesem Kapitel werden die notwendigen Grundlagen vermittelt, die zum besseren Verständnis des Themas notwendig sind. Außerdem ist es erforderlich einige Definitionen einzuführen, um einen begrifflichen Widerspruch auszuschliessen.

2.1 Biometrische Verfahren

Zunächst ist der Begriff **Biometrik** zu definieren. Er setzt sich aus den Bestandteilen „Biologie " und „Metrik " zusammen, diese werden im Folgenden genauer untersucht.

Biologie [LaFr]

- 1. Wissenschaft von den Lebewesen und den Gesetzmäßigkeiten des Lebens
- 2. Beschaffenheit eines Lebewesens
- 3. der Natur entsprechende Beschaffenheit

Metrik [Teu96][S.254]

Der mathematische Begriff Metrik verallgemeinert das Verhältnismaß „Abstand" auf beliebige Mengen:

Eine nichtleere Menge X heißt genau dann ein metrischer Raum, wenn jedem geordneten Paar(x, y) von Punkten x und y aus X stets eine reelle Zahl $d(x, y) \geq 0$ zugeordnet wird.

$$d : X \times X \to \mathbb{R} \text{ mit } (x, y) \mapsto d(x, y)$$

$d(x, y)$ heißt Abstand oder Distanz zwischen den Punkten x und y. Für alle $x, y, z \in X$ muß gelten:

- (M1) Der Abstand zwischen x und y ist gleich dem Abstand y und x (Symmtrie). $d(x, y) = d(y, x)$
- (M2) Der Abstand zwischen x und z ist kleiner oder gleich der Summe der Abstände zwischen x, y und y, z (Dreiecksungleichung). $d(x, z) \leq d(x, y) + d(y, z)$

- (M3) Wenn der Abstand zwischen x und y gleich null ist, folgt daraus, dass x und y gleich sind. $d(x, y) = 0$ genau dann, wenn $x = y$

Eine Menge, auf die eine Metrik definiert wurde, heißt metrischer Raum.

Biometrik beschreibt eine Klasse von sogenannten biometrischen Algorithmen, die eine computergestützte Vergleichbarkeit von verschiedenen digitalen Aufzeichnungen biologischer Personenmerkmale herstellen. Biometrische Algorithmen spannen einen metrischen Raum auf, in dem die Abstände verschiedener biometrischer Merkmale mathematisch eindeutig definiert sind. Daraus resultiert die folgende Definition:

> **Biometrischer Algorithmus** : Ein Verfahren, das aus einem biologischen Merkmal, eine vergleichbare Kenngröße generiert. Diese Kenngröße heißt „*biometrische Signatur*“. [Barg02][S.6]

Die Eignung eines biometrischen Algorithmus wird daran gemessen, dass die Signaturen zweier unterschiedlicher Abbildungen desselben biologischen Merkmals nahe beieinander liegen, während der Abstand der Signaturen zweier verschiedener biologischer Charakteristika groß sein sollte.

Eigenschaften von biologischen Merkmalen: Man hat festgestellt, dass sich eine Vielzahl an unterschiedlichen biologischen Merkmalen für die Biometrik eignen könnten. Diese Merkmale haben folgende Eigenschaften:

- **Eindeutigkeit für jede Person**: Es können nur biologische Merkmale , welche sich zwischen einzelnen Personen deutlich unterscheiden, für biometrische Anwendungen herangezogen werden. Dabei müssen diese Merkmale auch bei eineiigen Zwillingen genug Varianz erzeugen. Nur so kann zugesichert werden, dass für zwei Individuen auch unterschiedliche biometrische Signaturen vorliegen.

- **Unveränderlichkeit**: Je nachdem welches Anwendungsfeld der Biometrik betrachtet wird, kann die Forderung, dass sich die biologischen Merkmale gar nicht oder zumindest in einer akzeptablen Toleranz verändern, notwendig sein.

Sinnvolle weitergehende Forderungen sind:

- **Technische Eignung**: Das Merkmal hat hinsichtlich der Messmethode, Messdauer und Messkosten technisch geeignet zu sein.
- **Anwender Akzeptanz**: Die Messmethode und das Merkmal sollten akzeptabel für die Anwender sein. Insbesondere darf es zu keinerlei Beeinträchtigung der Gesundheit kommen.
- **Merkmalverbreitung**: Das Merkmal sollte weit verbreitet, dass heißt, bei fast allen potenziellen Nutzern vorhanden sein.

Bei der Klassifizierung biologischer Merkmale unterscheidet man zwischen *phänotypischen* und *genotypischen* Merkmalen.

Phänotypisch Die Ausprägungen phänotypischer Merkmale werden von den Erbanlagen und auch von den Umwelteinflüssen, denen der Träger zu einem früheren Zeitpunkt seiner Entwicklung ausgesetzt war, bestimmt. Dieses führt dazu, dass diese Merkmale sich auch bei Menschen mit den selben Erbanlagen unterscheiden. Ein Beispiel für *phänotypische* Merkmale sind die Strukturen der menschlichen Iris. [Barg02][S.8]

Genotypisch Die Erscheinung genotypischer Merkmale ist vollständig durch die Erbanlagen des Trägers festgelegt. Dieses führt dazu, dass genotypische Merkmale bei eineiigen Zwillingen die gleiche Signatur erzeugen. Ein Beispiel für ein genotypisches Merkmal ist die DNA eines Menschen. [Barg02][S.8]

Wie man schon an den Beispielen sieht, nimmt ein biometrisches System physiologische und/oder verhaltenstypische Merkmale einer Person auf. Zu den physiologischen Merkmalen gehören statische Körpermerkmale, wie bei der Finger-, Gesichts- oder Iriserkennung. Verhaltenstypische Merkmale hingegen beschreiben dynamische Kennwerte. Diese werden meist aus einem zeitlichen Verlauf von statischen Merkmalen erzeugt. Unterschriftensysteme erfassen nicht nur das Bild der Unterschrift, sondern auch deren Erzeugung. Dabei zeichnen sie den Druck und die Schreibgeschwindingkeit des Stiftes auf.

Ein biometrisches System kann zur Erhöhung der Sicherheit mehrere biometrische Indikatoren heranziehen. Eine Möglichkeit wäre es, Gesichtserkennung mit zusätzlicher Sprach- und Fingererkennung zu koppeln. Nur wenn diese drei, für sich genommen autonome Verifikationsverfahren, positiv übereinstimmen, wird das Gesamtsystem ein positives Feedback liefern. Hierbei unterscheidet man den zeitlichen Verlauf der Aufzeichnung der biometrischen Rowdaten:

- **Monomodal**: Jedes biometrische Merkmal wird zeitlich nacheinander aufgezeichnet und eventuell verarbeitet.
- **Multimodal**: Alle biometrischen Merkmale werden gleichzeitig erfasst und eventuell auch verarbeitet.

Bewertung eines biometrischen Algorithmus - FAR /FRR [Barg02][S.9]

Die Ergebnisse der Anwendung eines biometrischen Algorithmus auf unterschiedliche Aufnahmen desselben Merkmals (beispielsweise verschiedene Bilder derselben Iris) sind i.A. nicht zu 100 Prozent identisch, sondern liegen nur nahe beieinander. Aus diesem Grund muss ein biometrischer Algorithmus innerhalb eines gewissen Toleranzrahmens Signaturen als zu demselben Merkmal (bzw. zu demselben Träger) gehörend erkennen. Ähnliche Signaturen werden als „gleich“ und Signaturen, deren Abstand außerhalb des Toleranzrahmens liegt, als „unterschiedlich“ bewertet. Die Festlegung dieses Toleranzrahmens besitzt signifikanten Einfluss auf die Güte eines biometrischen Algorithmus. Legt man den Toleranzrahmen zu großzügig fest, kann es leichter zu Fehlurteilen kommen, ist er zu eng gewählt, kann es passieren, dass auch der Träger der Signatur nicht korrekt erkannt wird (z.B. bei zu schlechter Qualität der Bilddaten).

In diesem Zusammenhang spricht man von zwei Kenngrößen: der FAR und der FRR.

False Acception Rate (FAR): Bezeichnet die Rate der fehlerhaften Zuordnungen einer Signatur zu einem Träger. Je höher also die FAR, desto größer ist die Wahrscheinlichkeit, dass ein Betrüger erfolgreich eine falsche Identität vortäuschen kann. Die FAR berechnet sich wie folgt:

$$FAR = \frac{NFA}{NIA} * 100\%$$

Wobei NFA (Number of False Acceptances) die Anzahl der fälschlich Akzeptierten und NIA (Number of Imposter Attempts) die Gesamtzahl der unberechtigten Zutrittsversuche angibt.

False Rejection Rate (FRR): Bezeichnet die Rate der fehlerhaft fehlgeschlagenen Zuordnungen von Signaturen, also die Wahrscheinlichkeit, mit der z.B. einem Kontobevollmächtigten der Zugriff verweigert wird. Die FRR berechnet sich wie folgt:

$$FRR = \frac{NFR}{NEA} * 100\%$$

Wobei NFR (Number of False Rejections) die Anzahl der fälschlichen Rückweisungen und NEA (Number of Enroll Attempts) die Gesamtzahl der berechtigten Zutrittsversuche angibt.

Die Kenngrößen FAR und FRR sind von einander abhängig. Eine Verbesserung der einen Größe hat im Allgemeinen eine Verschlechterung der anderen zur Folge. Wenn der Toleranzrahmen für die Signaturen eingeengt wird, führt dieses z.B. zu einer niedrigeren FAR, da das Kriterium für die Erkennung einer Signatur verschärft wurde. Es resultiert aber ebenso in einer Erhöhung der FRR, da nun auch die Anforderungen an die Aufnahme des Merkmals erhöht wurden. Durch geschickte Parameterwahl kann man die FAR oder die FRR so anpassen, dass der Wert über oder unter einer bestimmten festgelegten Grenze fällt.

Deswegen muß das Verhältnis von FAR und FRR mitbetrachtet werden um, ein biometrisches System zu beurteilen.
Trägt man das Verhältnis von FAR/FRR zu der Zulassungstoleranz in ein Diagramm ein, kann man einen Schnittpunkt sehen.

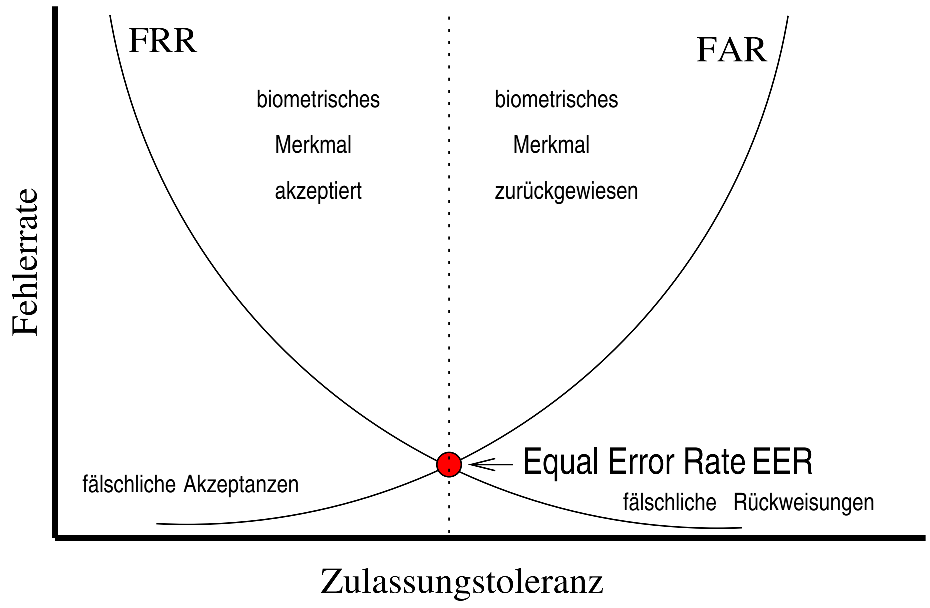

Abbildung 1: FAR/FRR

Im Schnittpunkt der beiden Kurven von FAR und FRR gilt die Gleichung FAR=FRR. Diesen Schnittpunkt bezeichnet man **Equal Error Rate** (EER). Die Güte eines biometrischen Systems kann man mit dieser beschreiben.
Es sollte aber beachtet werden, dass es möglich ist, jeweils einen der Werte (FAR/FRR) zu minimieren. Deswegen muss bei biometrischen Systemen, das Einsatzgebiet mit betrachtet werden.
Bei einem Geldautomat wird man versuchen, die FRR, bei noch tolerierbaren FAR, zu minimieren. Denn keine Bank könnte ihren Kunden zumuten, dass sie beim Vorgang des Geldabhebens nicht als legitimer Kontobesitzer authorisiert werden.
Anderseits wird es eher akzeptiert, das berechtigte Personen nicht in Hochsicherheitsbereiche kommen, als das ein unberechtigter Zugang erhält.

Zusätzlich zu der *EER* kann man noch eine weitere Kenngröße spezifizieren **ARE**. Dabei betrachtet man die Umgebung von *EER*. Besitzen die FAR- und FRR-Kurve ein großes Tal und die Kurven steigen/fallen langsam an, so wird dieses System im praktischem Einsatz eine kleinere Fehlerrate aufweisen. Steigen/fallen die beiden Kurven stark an, so dass sich das Tal verkleinert, dann kann

man davon ausgehen, dass sich die Fehlerrate erhöht.
ARE ist definiert als ein Fläche, die von den zwei Fehlerkurven, sowie einer Waagerechten eingeschlossen wird. Die Waagerechte ist definiert durch:

$$y = EER + z\%$$

Je größer die Fläche, desto einfacher kann die optimale Zulassungstoleranz für ein System geschätzt werden. Bei gleicher Fehlerrate ist das System mit dem größeren **ARE**-Wert zu bevorzugen.

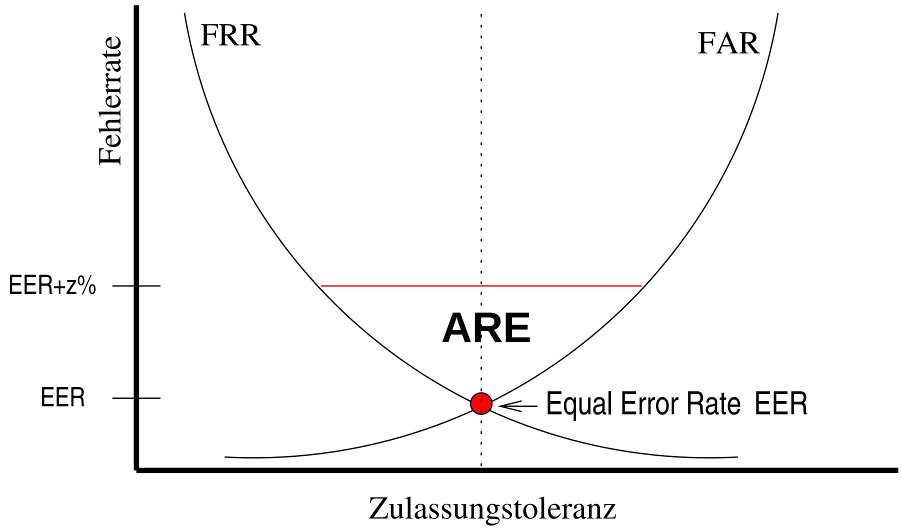

Abbildung 2: ARE

Anwendungsgebiete der Biometrik - Verifikation und Identikation

Biometrische Verfahren lassen sich in ihrer Anwendung in zwei Klassen einteilen, *Verifikation* und *Identikation*. Obwohl diese beiden Klassen zunächst sehr ähnlich erscheinen, ergeben sich jedoch Unterschiede bei der Anwendung für den Nutzer.

Biometrische Verifikation: Ein Authentikationssystem ist ein Prozess, bei dem es darum geht, eine zuvor angegebene Identität zu überprüfen. Durch die erzeugte biometrische Signatur weist das Subjekt dem System nach, dass er wirklich die Person ist, welche er vorgibt zu sein.

Es findet also ein *1:1* Vergleich statt. Das Authentikationssystem sucht nicht das passende Referenztemplate in seiner vorgehaltenen biometrischen Siganturdatenbank, sondern weiß anhand der Identität, welches Referenztemplate es zur Verifikation benutzen soll.

Biometrische Identikation: Ein Identikationssystem ist ein Prozess, bei dem die Aufgabe besteht, anhand einer erzeugten biometrischen Signatur, die zugehörige Person (Identität) zu ermitteln. In diesem Fall muss die zu überprüfende biometrische Signatur mit der gesamten Datenbank aller in Frage kommenden biometrischen Signaturen verglichen werden. Sollte eine hinreichende Übereinstimmung der aufgenommenen biometrischen Signatur mit einem Referenztemplate festgestellt werden, so läßt sich diese biometrische Signatur einem Subjekt zuordnen.
Es findet ein *1:n* Vergleich statt. Dabei kann man unterscheiden, ob durch die erfolgreiche Identikation das erkannte Subjekt anonymisiert ist oder mit Zusatzinformationen schon voher einmal erfasst wurde. Es ist ein wesentlicher Unterschied, ob man zuvor biometrisch erfasste Straftäter auf öffentlichen Plätzen zu erkennen versucht, oder Kundentracking in einer Einkaufsstrasse macht. Bei letzteren kann man zwar das Subjekt wiederfinden, aber keine weitergehenden Informationen zu der Identität des Subjekts abrufen.

2.2 Notwendigkeit von Kommunikationsprotokollen

Zunächst soll der Begriff Kommunikation definiert werden.

Kommunikation: Austausch von Information. Meist legt man fünf „Staionen" für die Übertragung einer Information fest; Sender, Codierung, Übertragungsmedium (*Kanal*), Decodierung und Empfänger. [DuInf93]
Bei Kommunikationen unterscheidet man verschiedene Formen und Ebenen.
Formen: Die Informationen fließen in nur einer (*unidirektional*), in zwei (*bidirektional*) oder in vielen Richtungen; sie können gezielt an einen Empfänger (wie im Gespräch zweier Personen), an festgelegte Empfänger (wie bei einem Vortrag oder bei der Einladung zu einer Sitzung) oder anonym an beliebig viele angeschlossene Empfänger (wie bei Funk und Fernsehen) gerichtet sein.
Ebenen: Kommunikation kann sich auf einer, rein physikalischen Ebene, auf den darüberliegenden Schichten des ISO/OSI-Schichtenmodells oder über einer semantischen Ebene abspielen.

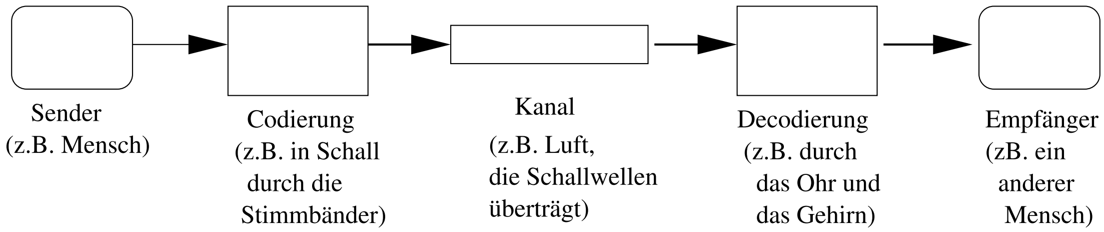

Abbildung 3: Kommunikation

Schaut man sich biometrische Authentikations-/Identikationssysteme an, erkennt man bei den Verfahren eine Vielzahl von Kommunikationsstellen. Nicht alle unterschiedliche Systeme erfassen und verarbeiten die gleichen Daten. Aber bestimmte Grundsätzlichkeiten lassen sich feststellen. Irgendwo muß die biometrische Charakteristik digital aufgezeichnet werden. Anschliessend findet eine Verarbeitung und Modifikation der Daten statt. Alle Verfahren lassen sich schematisch gleich auffassen, wie folgende Abbildung illustriert:

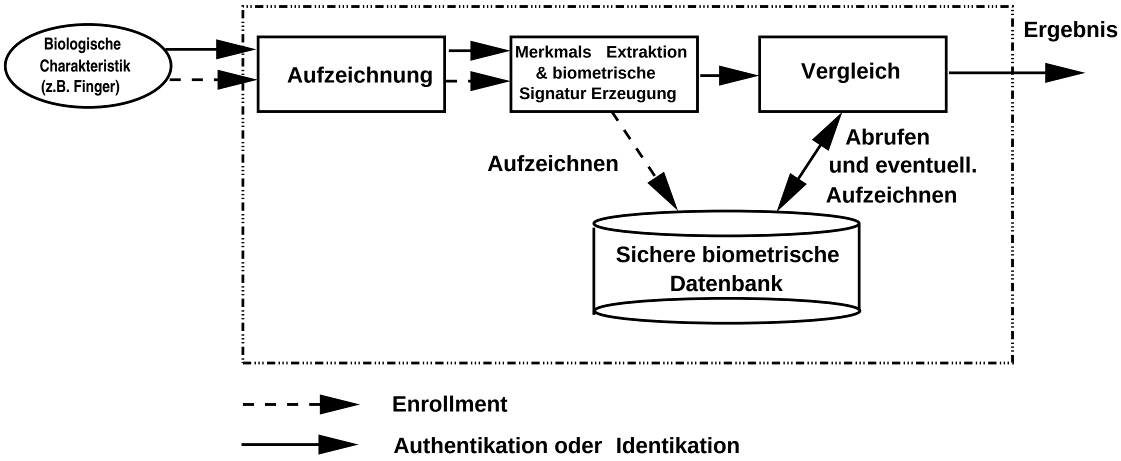

Abbildung 4: Datenfluss

Damit zwischen zwei Kommunikationspartnern ein Austausch von Information stattfinden kann, benötigt man ein Kommunikationsprotokoll.

Protokoll:Vereinbarung über den geordneten Ablauf einer Kommunikation, wobei die Vereinbarung in der Informatik diktatorischen Charakter besitzt: Wer sich nicht an sie hält, wird von der Kommunikation ausgeschlossen. Entsprechend der aufeinander aufbauenden Schichten bei der Übertragung von Informationen wird für jede Ebene ein eigenes Protokoll vereinbart, dessen Realisierung sich auf das Protokoll der nächst tieferen Ebene abstützt. Protokolle sind bei der Kopplung von Systemen und in „Offenen Systemen“ (bisher) unverzichtbar.
Für die präzise Definition von Protokollen eignen sich endliche Automaten oder Petri-Netze. Hierbei nimmt man an, dass sich jeder Kommunikationspartner und das übertragende Medium in Zuständen befinden, die man zu einem Geamtzustand zusammenfaßt. [DuInf93]

In einem Kommunikationsprotokoll wird also exakt definiert, wie ein Datenaustausch zwischen zwei Parteien auszusehen hat. Das Protokoll legt Format, Bedeutung und Reihenfolge der zu übertragenden Nachrichten fest. Somit sind die Stationen der Kommunikation genau festgelegt, und beide Parteien können die Informationen verwerten, welche sie austauschen.

Man unterscheidet bei Kommunikation zwischen verbindungsorientierten und verbindungslosen Protokollen.
Bei Lienemann [Lie00][S.45f] werden die beiden Typen wie folgt unterschieden:

Eigenschaften eines verbindungslosen Protokolls

- Der Datenaustausch findet über einen einfachen Datenpaket-Verkehr zwischen den Kommunikationspartnern statt. Die Pakete sind über ihre Quell- und Zieladresse identifizierbar und können so zugestellt werden. Es gibt keine Sicherungsmechanismen, die gewährleisten, dass Pakete tatsächlich ankommen und unverfälscht sind.

- Aus dem Fehlen der Sicherungsmechanismen resultiert ein, verglichen mit verbindungsorientierten Protokollen, schnelles Protokoll.

- Aus dem Fehlen der Sicherungsmechanismen resultiert aber auch eine höhere Störanfälligkeit:
  a) Pakete können verloren gehen.
  b) Pakete erreichen ihr Ziel in falscher Reihenfolge.
  c) Pakete können doppelt empfangen werden.

- Um eine verlässliche Kommunikation zu gewährleisten, müssen übergeordnete Protokolle ausreichende Kontrolle und Sicherung bieten.

Eigenschaften eines verbindungsorientierten Protokolls

- Es findet eine Bestätigung der übermittelten Informationen statt. Dieses kann von einer einfachen Empfangsbestätigung bis zur kompletten Überprüfung der Datenintegrität reichen. Durch diese Sicherung des Datenaustausches kann verhindert werden, dass Datenpakete verloren gehen.

- Der Datenaustausch vollzieht sich in drei Phasen:
  a) Verbindungsaufbau
  b) Datenübertragung
  c)Verbindungsabbau.

- Beide Kommunikationspartner besitzen lokal Statusinformationen der Verbindung. Somit müssen zu keinem Zeitpunkt Statusinformationen angefordert oder übertragen werden. Es herrscht eine „gute Orientierung“ innerhalb der Kommunikation.

2.3 Netz Architektur

In 2.2 wurde erläutert, warum ein Kommunikationskanal für biometrische Anwendungen benötigt wird. Da es in dieser Arbeit um einen speziellen Kommunikationskanal geht, verwirklicht über das Netzwerksystem TCP/IP, wird dieses zugrundeliegende Protokoll genauer erläutert.

2.3.1 ISO/OSI:

Zur Beschreibung der Struktur und Funktion von Protokollen für die Datenkommunikation wird häufig ein Architekturmodell herangezogen. Zum Vergleich von unterschiedlichen Netzwerkprotokollen hat sich das ISO/OSI-Referenzmodell [1] bewährt. Das Modell unterteilt die Netzkommunikation in 7 Schichten(layers). Jede Schicht des OSI-Modells repräsentiert eine Funktion, die beim Austausch von Daten zwischen Anwendungen über ein dazwischenliegendes Netzwerk hinweg ausgeführt wird. Diese Art der Darstellung erscheint wie ein „Haufen von Ziegelsteinen", die übereinander gestapelt wurden - deshalb spricht man oft von einem *Stack* (Stapel) oder *Protokollstack*.

[1] Architekturmodell der Internationalen Standard Organisation (ISO) für offene Netze (Open System Interconnection, OSI)

Schicht	Schichtname	Beschreibung
7	Anwendungsschicht	besteht aus den Anwendungen, mit denen man das Netz nutzen kann
6	Darstellungsschicht	standardisiert das Format der Daten auf dem Netz
5	Kommunikationsschicht	verwaltet die Verbindung zwischen den Anwendungen
4	Transportschicht	garantiert die fehlerfreie Datenübertragung durch Fehlererkennung und -korrektur
3	Vermittlungsschicht	verwaltet die Verbindungen zwischen den Rechnern im Netz für die höheren Schichten
2	Sicherungsschicht	sorgt für die zuverlässige Übertragung der Daten über die physikalische Verbindung
1	Bitübertragungsschicht	definiert die physikalischen Eigenschaften der Übertragung

Tabelle 1: ISO/OSI Schichtenmodell

Anwendungsschicht: Die Anwendungsschicht, die höchste Schicht des Referenzmodells, stellt die Mittel zur Kooperation zwischen verteilten Anwendungsprozessen zur Verfügung. Der Anwendungsprozeß hat ausschließlich über die Schicht 7 Zugang zum OSI-Netzwerk.

Darstellungsschicht: Die Darstellungsschicht stellt Ausdrucksmittel (z.B. Zeichenvorrat, Datentypen) zur Verfügung, die es den Anwendungsinstanzen ermöglichen, Begriffe eindeutig zu benennen, und legt im Darstellungsprotokoll die Regeln fest, wie die in der gemeinsamen Sprache dargestellte Information auszutauschen ist.

Kommunikationssteuerungsschicht: Die Kommunikationssteuerungsschicht regelt den Gesprächswechsel zwischen kommunizierenden Partnern und liefert Vorkehrungen zum Wiederanlaufen einer unterbrochenen Übertragung ab einem zwischen den Partnern vorher vereinbarten Punkt im Datenstrom (Synchronisationspunkt). Sie regelt den Betriebsablauf.

Transportschicht: Die Transportschicht nimmt die Anforderung der Anwenderprozesse hinsichtlich der Übertragungsqualität entgegen, erstellt entsprechende Aufträge an die darunterliegenden Schichten und gleicht gegebenenfalls ungenügende Leistungen dieser Schichten aus.

Vermittlungsschicht: Die Vermittlungsschicht transportiert Pakete über die Teilstrecken des Netzes von Endsystem zu Endsystem.

Sicherungsschicht: Die Sicherungsschicht verbessert ungesicherte Verbindungen auf Teilstrecken zu gesicherten Verbindungen. Gesichert heißt dabei eine Verbindung, die Fehler in der Datenübertragung erkennt und korrigiert.

Bitübertragungsschicht: Die Bitübertragungsschicht stellt ungesicherte Verbindungen zwischen Systemen für die Übertragung von Bits zur Verfügung.

Jede einzelne Schicht definiert nicht etwa ein Protokoll, sondern sie stellt vielmehr eine Funktionalität der Datenkommunikation dar, die von beliebig vielen Protokollen ausgeführt werden kann. Jede Schicht kann also mehrere Protokolle enthalten, von denen jedes solche Dienste bereitstellt, wie sie für die Erfüllung der Funktion dieser Schicht benötigt werden. Jedes Protokoll kommuniziert mit seinem Peer(Gegenüber). Ein *Peer* ist die Implementierung desselben Protokolls in der entsprechenden Schicht des Kommunikationspartners.

2.3.2 TCP/IP:

Der Name TCP/IP bezeichnet eine ganze Reihe von Protokollen für die Datenkommunikation. Diese Protokollfamilie bezieht ihren Namen von zweien der enthaltenen Protokolle - dem Transmission Control Protocol (TCP) und dem Internet Protocol (IP). Obwohl noch einige andere Protokolle enthalten sind, sind TCP und IP sicherlich zwei der wichtigsten. In diesem Text wird nur auf die noch aktuelle Version 4 von TCP/IP eingegangen.

TCP/IP wurde für das ARPANET (Advanced Research Projects Agency), dem Vorgänger des heutigen Internets, entwickelt. Das ARPANET war in seiner ursprünglichen Nutzung ein rein militärisches Netz, das zur Koordination von weiträumig verteilten militärischen, behördlichen und wissenschaftlichen Einrichtungen dienen sollte. Die Entwicklung der TCP/IP Architektur, welche die bis dahin verwendeten Protokolle ablösen sollte, begann etwa 1978, 1981 wurde TCP/IP im RFC[2] 793 standardisiert. Im Jahre 1983 wurden die TCP/IP-Protokolle zu „Military Standards "(MIL STD) erhoben und alle Institutionen, die am ARPANET partizipierten, mußten auf die neuen Protokolle umstellen. Eine der Hauptvorgaben bei der Entwicklung des ARPANETs und der damit verbundenen Protokolle war, dass auch beim Ausfall einiger Knoten (durch z.B. äußere Einwirkung des Feindes) das restliche Netz weiterhin funktionsfähig bleiben sollte.

Aus dieser Vorgabe resultieren einige Eigenschaften des TCP/IP Modells.

- Offene Protokollspezifikationen, die frei zugänglich und unabhängig von der Hardware und dem Betriebssystem sind. Aufgrund seiner weitreichenden Unterstützung eignet sich TCP/IP in idealer Weise dazu, unterschiedliche Hardware und Software miteinander zu verbinden.
- Unabhängigkeit von einer bestimmten Netzwerkhardware. So ist es möglich, mittels TCP/IP viele unterschiedliche Netze miteinander zu verbinden. TCP/IP kann über ein ETHERNET, einen Token-Ring, eine Wählleitung, ein X.25-Netz und beinahe jedes andere physikalische Übertragungsmedium betrieben werden.
- Ein einheitliches Adressierungsschema, das es jedem Rechner in einem TCP/IP-Netz ermöglicht, jeden beliebigen anderen Rechner in diesem Netz eindeutig zu identifizieren.
- Standardisierte Protokolle der höheren Schichten, die dem Benutzer einen einheitlichen und weithin verfügbaren Dienst zur Verfügung stellen.

TCP/IP lässt sich in vier Schichten einteilen.

[2]RFC steht für: request for comment. Die RFCs sind Arbeitspapiere, Protokollspezifikationen oder auch nur Kommentare zu aktuellen Themen der Internet Community. RFCs werden von der Internet Engineering Task Force (IETF) erstellt und von der Internet Engineering Steering Group (IESG) formell abgesegnet. Siehe auch http://www.ietf.org

Schichtebene	Schichtname	Beschreibung
4	Anwendungsschicht	enthält Anwendungen und Prozesse die auf das Netzwerk zugreifen
3	Transportschicht	stellt End-zu-End-Datendienste zur Verfügung
2	Internetschicht	definiert den Aufbau von Datagrammen und routet Daten
1	Netzzugangsschicht	enthält Routinen für den Zugriff auf physikalische Netze

Tabelle 2: IP Schichtenmodell

Wie man sieht, besteht der Protokollstack von TCP/IP nicht aus 7, sondern nur aus 4 Schichten. Dieses Modell mit vier Schichten basiert auf den drei Ebenen, die im Protocol Model des DOD (US-Verteidigungsministrerium) im *DDN Protocol Handbook* beschrieben werden.

Es fehlen die Darstellungsschicht und die Kommunikationssteuerungsschicht aus dem OSI-Modell. Die Kommunikationssteuerungsschicht ist bei TCP/IP hauptsächlich in der Transportschicht untergebracht. Es exisitieren auch keine *Sessions* wie bei OSI, sondern Verbindungen werden bei TCP/IP über *sockets* und *port* referenziert und somit die zuständigen Applikationsendpunkte festgelegt. Falls Applikationen bei TCP/IP zusätzliche Kommunikationssteuerungsdienste benötigen, müssen diese auf Applikationsebene erbracht werden. Ein Beispiel sei „Network File System“ (NFS), welches seine eigene Kommunikationssteuerung mitbringt, die „Remote Procedure Call“ (RPC).
Auch die Darstellungsschicht fehlt bei TCP/IP. Diese Funktionalität liegt auf Applikationsebene. Ein Beispiel sei „Multipurpose Internet Mail Extensions “ (MIME).

Die Tranportschicht teilt sich vertikal bei TCP/IP auf in „User Datagram Protocol “ (UDP) und „Transmission Control Protocol “ (TCP).

UDP *UDP* baut einen verbindungslosen Kommunikationskanal auf. Pakete können verloren gehen, sich verdoppeln oder korrupt sein. Die darunter liegen-de Netzwerkschicht behandelt nämlich jedes Paket wie eine eigene Einheit und deswegen kann jedes Daten-Paket bei einem paketvermittelnden Netz, wie IP, unterschiedliche Netzrouten nehmen. Wenn bei der Übertragung von Daten ein hoher Durchsatz erfoderlich ist, wobei nicht jedes Datum relevant ist, dann nutzt man meist den *UDP* Dienst.

TCP *TCP* ist für Verbindungen gedacht, die verlässlich bei der Übertragung der Daten arbeiten sollen. *TCP* ist zuständig für die Datenwiederherstellung von verlorenen, korrupten oder auch in falscher Reihenfolge erhaltenen Daten-Paketen. Dies gewährleistet *TCP* durch Sequenznummern in jedem Daten-Paket. TCP erfordert, dass jedes übertragene Daten-Paket quittiert wird.

TCP und UDP kommunizieren beide mit dem Konzept der **Ports**.Ein Port ist eine virtuelle Schnittstelle, welche auf einer Netzschnittstelle geöffnet werden kann. Viele Ports sind bei TCP/IP mit dem zuständigen Dienst assoziiert. Über einen Port kann man sozusagen bestimmen, welche Applikation die Daten erhalten sollen. Durch Ports besteht die Möglichkeit zu multiplexieren, also mehrere paralelle Verbindungen zu öffnen.

Die Portnummern zusammen mit der Quell- und Zieladresse spezifizieren einen **Socket**. Jede Maschine, welche über TCP/IP kommuniziert, wird ein Socket zu der empfangenden Maschine hin öffnen. Sind die Sockets verbunden, steht bei TCP ein verlässlicher Dienst zu Verfügung. Eine Applikation kann mehrere Sockets öffnen und über diese parallel kommunizieren.

Die Internet-Schicht bei TCP/IP ist fast identisch zu der Vermittlungsschicht vom OSI-Modell. Ein Unterschied ist, das bei IP nur verbindungslose Kanäle existieren. Die Vermittlungsschicht ist zuständig für den Weg, welchen die Pakete von einem Kommunikationspartner zum anderen nehmen. Dabei wird für jedes Daten-Paket unabhängig entschieden, welche Route dieses durch das Netz nehmen soll.
Ein weiterer Unterschied zu OSI ist, dass IP eine feste Länge für die Adresse (32 Bit) verlangt.

Die Vermittlungsschicht kümmert sich darum, die Daten, die sie von der Transportschicht erhalten hat, in Pakete mit bestimmten Längen zu unterteilen, welche den Beschränkungen der physikalischen Schicht Rechnung tragen (Fragmentierung).

Über die Sicherungsschicht und Bitübertragungsschicht wird bei TCP/IP nichts weiter erwähnt, da TCP/IP auf unterschiedlichen darunterliegenden Netzwerktypen aufgebaut ist. Es verlangt nur irgendein Protokoll, welches dem Rechner ermöglicht irgendwelche IP-Pakete zu verschicken. Da also die unteren Protokolle nicht näher spezifiziert sind, variieren diese von Rechner zu Rechner und Netzwerk zu Netzwerk.

Im Folgenden wird die zugrundeliegende Netz-Architektur zusammengefasst als Grafik dargestellt:

Schicht	ISO/OSI	TCP/IP
7	Anwendungsschicht	Andwendungsschicht
6	Darstellungsschicht	
5	Kommunikationsschicht	
4	Transportschicht	Transportschicht
3	Vermittlungsschicht	Internet-Schicht
2	Sicherungsschicht	Subnetz-Schicht
1	Bitübertragungsschicht	

Tabelle 3: TCP/IP-ISO

Genau wie im OSI-Modell werden die Daten im Stack nach unten weitergereicht, wenn Daten verschickt werden. Beim Empfang verläuft der Weg genau umgekehrt, von Netzzugangsschicht zu der Anwendungsschicht. Jede Schicht fügt ihre eigenen Kontrollinformationen hinzu. Diese Information nennt man *Header* (Kopf), da sie den eigentlichen Daten vorangestellt wird. Jede Schicht betrachtet die gesamte Information, die sie von der darüberliegenden Schicht empfängt, als zu übertragende Daten. Bei dem Empfangen von Paketen wird der umgekehrte

Weg genommen. Jede Schicht trennt ihren Header von dem Datenpaket ab und reicht, falls keine Fehler aufgetaucht sind, den restlichen Datenteil eine Schicht höher.

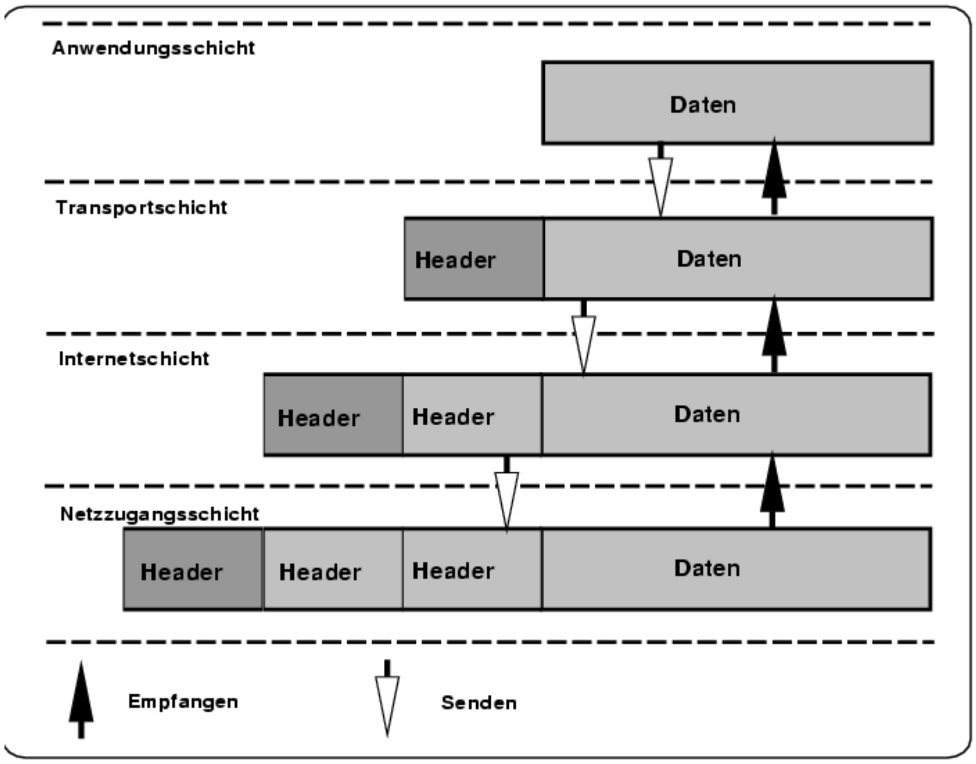

Abbildung 5: IP Paketschachtelung

2.4 Sicherheit

Der Begriff der Sicherheit im IT-Bereich läßt sich nicht so leicht und kurz definieren. In der Regel muss man zunächst beschreiben, welche Dienste man betreibt und was für Reaktionen diese im System selber haben, als auch welche Wechselwirkung diese auf die Umgebung des Systems haben.

Sicherheit [MBK00][S.22] Ein System ist im Sinne der IT-Technologie sicher , wenn folgendes gewährleistet wird:

- Alle gewollten und im Leistungsprofil des Systems spezifizierten Aktivitäten an, mit und durch das System sind möglich. Das System verhält sich, wie von ihm erwartet.
- Alle unautorisierten Aktivitäten an, mit und durch das System werden verhindert. Ein Missbrauch ist unmöglich.

In diesem Zusammenhang muss natürlich im Vorfeld klar spezifiziert werden, welche Aktivitäten das System zur Verfügung stellt und was unautorisierte Aktivitäten sind.

Hier fällt schon auf, das bei einem komplexen System, dieses eine sehr umfangreiche Arbeit ist, welche nicht ohne weiteres auf Vollständigkeit zu verifizieren ist.

Deswegen gibt es verschiedene Vorgehensweisen um solch ein Modell aufzubauen. Reinhard Betram [SS00][S.103f] beschreibt 4 unterschiedliche Lösungswege:

- **Grundschutz und Basissicherheit**: Hierbei wird versucht, jede Komponente einer IT-Infrastruktur zu identifizieren, um dann ein getrenntes Sicherheitskonzept für diese zu erarbeiten.
- **Maßnahmenbasierte Sicherheit**: Entgegen der recht abstrakten Aufspaltung in Komponenten wird hier ein System als Ganzes betrachtet und Maßnahmen(Controls) werden definiert, die das System sicher gegenüber bekannten Angriffen machen. Die maßnahmenbasierte Sicherheit definiert somit Controls, die gegen unsterschiedliche Schwachstellen und Gefährdungen wirken.

- **Gefährdungsbasierte Sicherheit**: Ein System kann oftmals auf sehr unterschiedliche Weise angegriffen werden. Die Gefährdung steigt, wenn nicht ensprechende Maßnahmen unternommen wurden. Die gefährdungsbasierte Sicherheit listet dementsprechend zu jeder Schwachstelle eine mögliche Maßnahme auf.

- **Motivationsbasierte Sicherheit**: Da ein System auch aus den Anwendern und den Betreibern, Administratoren usw. besteht, versucht die motivationsbasierte Sicherheit das kreative Verhalten potentieller Angreifer nachzustellen und hierüber Einsichten in das Gesamtsystem zu gewinnen.

Bei meiner Betrachtung des Systems geht es um die Sicherheit einer Netzkommunikation. Die beiden kommunizierenden Einheiten werden als Blackboxen betrachtet und es wird der Fokus auf den Kommunikationskanal gelegt. Es wird also nicht überprüft, ob eine physische Unsicherheit gegenüber den Kommunikationseinheiten besteht oder ob der nicht netzwerkgebundene Programmablauf korrekt durchgeführt wird. Es wird auch nicht analysiert, ob die Kommunikationseinheit gegen Ausspähug oder Entwendung gesichert ist.

Welche Sicherheitskriterien kann ein Kommunikationskanal besitzen:

- Identifizierung (authentication)
- Vertraulichkeit (confidentiality)
- Integrität (integrity)
- Verbindlichkeit (nonrepudiation)
- Verfügbarkeit (availability)

Wir versuchen Schutzziele gegen

- unbefugten Informationsgewinn (Verlust der Vertraulichkeit)
- unbefugte Modifikation von Information (Verlust der Integrität)

- unbefugte Annerkenung als Kommunikationspartner (Verlust der Identifizierung)

zu entwickeln.

2.5 Grundlagen der Kryptographie

Die Kryptographie bietet mächtige und vielseitige Hilfsmittel, die es ermöglichen, die Sicherheit in vernetzten Systemen zu erhöhen. Im folgenden werden die wichtigsten kryptographischen Prinzipien und Algorithmen vorgestellt. Dabei liegt der Schwerpunkt auf den Chiffren (sowohl den symmetrischen wie auch den asymmetrischen), aber auch andere notwendige Aspekte (wie Einweg-Hashfunktionen oder digitale Signaturen) werden erläutert. Die genauen Algorithmen werden nicht erklärt. Für eine tiefere Betrachtung der Algorithmen, verweise ich auf das Buch „Handbook of Applied Cryptography " von A. Menezes, P. van Oorschot und S. Vanstone, CRC Press, 1996 oder auf „Angewandte Kryptographie " von Bruce Schneier, Addison-Wesley 1996.

2.5.1 Grundlegende Begriffe

Zum besseren Verständnis hier zuerst einige Defintionen und Abkürzungen von Begriffen, die im folgenden häufiger Verwendung finden werden:

- *Kryptologie*: Wissenschaft der Kryptographie und Kryptoanalysis.
- *Kryptographie*: Lehre vom geheimen Schreiben.
- *Kryptoanalysis*: Lehre vom "Brechen" von Chiffren.
- *Chiffre (cipher)* : Methode des Verschlüsselns.
- *Klartext (plaintext)*: unverschlüsselter Text (*M* für "Message", *K* für "Klartext" oder *P* für "Plaintext").
- *Chiffretext (ciphertext)*: verschlüsselter Text (*C* für "Chiffretext").
- *Chiffrieren (encrypt)*: verschlüsseln ($E_K(M) = C$).
- *Dechiffrieren (decrypt)*: entschlüsseln ($D_K(C) = M$).

Weiterhin wird später von sogenannten unsicheren Kanälen zu lesen sein. Darunter sind Datenwege zu verstehen, bei denen nicht ausgeschlossen werden kann, dass dritte Personen Zugriff auf die auf diesen Datenwegen übermittelten Daten erhalten. Zur Erläuterung von einigen Vorgehensweisen wird in diesem Text außerdem mit den Namen Alice und Bob gearbeitet. Dabei handelt es sich natürlich nicht um real existierende Personen, sondern vielmehr um Statthalter für jede Art von handelnden Akteuren der digitalen Welt (also sowohl Menschen wie auch Maschinen).

2.5.2 Ziele

Der Einsatz von kryptographischen Methoden in der Praxis verfolgt i.A. das Ziel der Durchsetzung einer oder mehrerer Anforderungen:

- *Vertraulichkeit, Geheimhaltung*: Hierunter ist zu verstehen, dass die betreffenden Dokumente bzw. die betreffende Kommunikation vor den neugierigen Blicken nicht autorisierter Personen zu schützen sind. Dieses wird normalerweise mittels Verschlüsselung der Dokumente erreicht.
- *Authentifizierung*: Besonders auf elektronischen Kommunikationswegen ist es sehr einfach, sich falsche Identitäten zuzulegen. Durch das Fehlen von persönlichem Kontakt ist man darauf angewiesen, den digitalen Informationen zu vertrauen. Ein vergleichbares Problem ist auch das Feststellen der Identitäten von Kommunikationspartnern. Was benötigt wird, sind Methoden zur zweifelsfreien Identifikation von Akteuren in der digitalen Welt (wobei hier Akteure selbstverständlich nicht nur Menschen, sondern z.B. auch Rechner oder Prozesse sein können).
- *Integrität*: Es muß möglich sein festzustellen, ob Daten von dritter Seite aus manipuliert wurden. Dieses bezieht sich sowohl auf Manipulation, die auf unsicheren Kommunkationswegen erfolgt sein mag, wie auch auf Datenmanipulation, die ein Eindringling eventuell im System durchgeführt hat (um z.B. Sytemsoftware durch eigene Programme zu ersetzen, die Hintertüren enthalten). In diesem Kontext ist unter Datenmanipualtion Modifikation, Vernichtung und Ersetzung von Daten zu verstehen.
- *Unwiderrufbarkeit*: Es sollte Methoden geben, die verhindern, dass ein Teilnehmer an einem Kommunikationsaustausch dessen Stattfinden im Nachhinein leugnen kann.

2.5.3 Der Begriff der Chiffre

Verschlüsselung von Daten erfolgt mittels eines Algorithmus, der Chiffre genannt wird. Dieser Algorithmus teilt sich auf in zwei Transformationen, eine zum Verschlüsseln der Daten (to encrypt, in Zeichen E) und einer zum Entschlüsseln (to decrypt, In Zeichen D). Sowohl die Ver- wie auch die Entschlüsselung wird unter Anwendung von Schlüsseln durchgeführt. Dabei können die Schlüssel zum Ver- und Entschlüsseln gleich (symmetrische Chiffren) oder unterschiedlich (asymmetrische Chiffren) sein. Um praktikabel zu sein, gibt es noch weitere Anforderungen an Chiffrieralgorithmen. Sie sollten einfach zu benutzen sein, d.h. es sollte kein Problem sein, passende Schlüssel zu generieren, und anhand der Kenntnis des Schlüssels sollte es einfach sein, aus der Chiffriertransformation die Dechiffriertransformation zu berechnen.

Weiterhin sollte die Sicherheit des Algorithmus ausschließlich von der Geheimhaltung des Schlüssels abhängen und nicht auf der Geheimhaltung des Algorithmus beruhen (keine "security through obscurity"). Denn man kann davon ausgehen, dass jeder noch so gut geschützte Algorithmus eines Tages aufgedeckt wird. Und schließlich sollten die Chiffrier- und Dechiffrier- Transformationen für alle Schlüssel effizient berechnet werden können, denn niemandem ist mit einem Algorithmus geholfen, der zwar nahezu unberechenbar ist, aber unverhältnismäßig aufwendig in Zeit und Systemressourcen ist (man kann sich Szenarien vorstellen, wo solch ein Algorithmus akzeptabel wäre, aber im Mittelpunkt dieses Aufsatzes stehen kryptographische Methoden, die in erster Linie Anwendung in der Kommunikation in Datennetzen finden).

2.5.4 Symmetrische Chiffren

Definition Bei Symmetrischen Chiffren trifft die Analogie des Schlüssels zu. Wie bei der Tür, die vom gleichen Schlüssel zu- und auch wieder aufgeschlossen werden kann, wird auch der Text bei symmetrischen Chiffren mit dem gleichen Schlüssel ver- und wieder entschlüsselt.

Bei einem sicheren Nachrichtenaustausch zwischen verschiedenen Parteien muß dieser Schlüssel allen Beteiligten bekannt sein. Und hier liegt auch das Hauptproblem, auf das man bei der Benutzung von symmetrischen Chiffren stößt:

Der sichere Schlüsselaustausch. Bevor man sicher vor ungewollten Lauschern mittels verschlüsselter Botschaften kommunizieren kann, muß man den Schlüssel an alle Gesprächspartner verteilen, falls er ihnen noch nicht bekannt ist. Doch dieses ist kein triviales Problem, da ja offensichtlich kein sicherer Kanal zu den Gesprächspartnern vorhanden ist (sonst müßte man den Nachrichtenaustausch schließlich nicht verschlüsseln).

In der Familie der symmetrischen Chiffren gibt es zwei Kategorien: Die *Stromchiffren*, welche die Daten bitweise verschlüsseln, und die *Blockchiffren*, die den Text in Blöcken fester Größe verschlüsseln.

Stromchiffren

Prinzip der Stromchiffren: Eine Stromchiffre verschlüsselt den Klartext i.A. bitweise (manchmal aber auch in größeren Einheiten). Dieses geschieht in den meisten Fällen durch eine bitweise exklusive Veroderung (XOR) mit einem Schlüsselstrom ($C_i = K_i \oplus S_i$, wobei C_i, K_i, S_i jeweils das *i*-te Bit des Chiffretextes, Klartextes und Schlüsselstroms sind). Die Entschlüsselung findet ebenfalls durch bitweises XOR mit dem gleichen Schlüsselstrom statt. ($C_i \oplus S_i = K_i \oplus S_i \oplus S_i = K_i \oplus 0 = K_i$). Der Schlüsselstrom wird durch einen Algorithmus (dem sogenannten Schlüsselstrom-Generator) erzeugt, der als Eingabe zur Initialisierung einen Wert (den eigentlichen Schlüssel) bekommt. Der gleiche Schlüssel erzeugt immer den gleichen Schlüsselstrom, so dass den einzelnen Kommunikationspartnern lediglich der Schlüssel bekannt sein muß, um den Strom an Chiffretext-Bits zu entschlüsseln. Die Sicherheit einer Stromchiffre beruht auf der Güte des Schlüsselstrom-Generators. Der Schlüsselstrom-Generator produziert eine Folge von Bits, deren Auftreten möglichst zufällig ist. Je besser der Generator den Zufall approximiert, desto sicherer wird die Chiffrierung. Bei der Verwendung von Stromchiffren sollte man darauf achten, niemals den gleichen Schlüssel (und somit den gleichen Chiffrestrom) zweimal zu verwenden. Falls man dieses tut, könnte eine dritte Person, die die beiden Chiffretexte abgefangen hat, diese beiden miteinander exklusiv verodern. Als Ergebnis erhält sie eine exklusive Veroderung der beiden Klartexte (die Schlüsselstrom-Bits fallen bei diesem Vorgehen weg). Dieses ist leicht zu entschlüsseln, da das Vorkommen der einzelnen

Zeichen in Sprache alles andere als zufällig ist. Durch die Kenntnis eines der Klartexte kann man nun den Schlüsselstrom berechnen (durch XOR des Klar- und des Chiffretextes) und hat dadurch die Möglichkeit, alle weiteren Nachrichten, die mit diesem Schlüssel chiffriert wurden, problemlos zu lesen.

Ebenfalls sollte man darauf achten, dass, wenn man einen Schlüsselstrom-Generator mit einem periodischen Schlüsselstrom verwendet, die Größe der zu verschlüsselnden Daten kleiner ist als die Größe der Periode. Falls dieses nicht erfüllt ist, gibt man eventuellen Lauschern einen Ansatzpunkt zum erfolgreichen Entschlüsseln des Textes: Es besteht so die Möglichkeit, den Chiffretext in Blöcke der Größe der Periode aufzuteilen und diese, ähnlich zu dem oben erwähnten Angriff, mit einander exklusiv zu verodern. Da sich der Schlüssel nach dem Durchlaufen der Periode wiederholt, fallen auch bei diesem Vorgehen die Schlüsselbits weg (s.o.).

Durch die oben beschriebenen Eigenschaften (Strom von Chiffretext-Bits durch bitweise Verschlüsselung, keine Notwendigkeit, spezielle Blockgrößen der zu verschlüsselnden Daten einzuhalten) eignen sich Stromchiffren besonders für den Einsatz an Orten, wo es auf die Verschlüsselung kontinuierlicher Datenströme ankommt (z.B. Video/Audio).

Arten von Stromchiffren: Man kann zwischen zwei unterschiedlichen Arten von Stromchiffren unterscheiden: den *synchronen* und den *selbstsynchronisierenden* Stromchiffren (vgl. Abbildung 6).

Bei den **synchronen Stromchiffren** ist der Schlüsselstrom unabhängig vom Datenstrom, d.h. der generierte Strom von Schlüsselbits wird allein bestimmt durch den verwendeten Algorithmus und den Schlüssel. Der Vorteil einer solchen Verfahrensweise ist, dass die Berechnung des Schlüsselstroms im vorhinein erledigt werden kann, und beim eigentlichen Nachrichtenaustausch nur noch die XOR-Verknüpfung mit den Daten stattfinden muß. Besonders bei rechenintensiven Algorithmen kann man so den Datendurchsatz der Kommunikation erheblich erhöhen. Der Nachteil von synchronen Stromchiffren ist, dass bei nur einem verlorengegangenen Bit der gesamte folgende Chiffretext unbrauchbar ist. Dieses ist der Fall, weil sowohl auf Sender- wie auch auf Empfängerseite die Schlüsselstrom-Generatoren die jeweiligen Daten-Bits mit dem selben

Schlüsselstrom-Bit (de)kodieren müssen, sie müssen also synchron vorgehen (daher auch der Name). Falls ein Bit verloren geht, geraten die beiden in Asynchronismus. Während der Sender schon bei Bit Nummer *i+1* ist, ist der Empfänger noch bei Bit Nummer *i* und versucht somit eine Dechiffrierung anhand eines falschen Schlüsselstrom-Bits. Dieses setzt sich auf den Rest des Chiffretextes fort. Fehlerhaft übertragene Bits wiederum machen keine größeren Probleme. Die fehlerhaften Bits werden zwar falsch dechiffriert, aber der Rest der Nachricht bleibt entschlüsselbar.

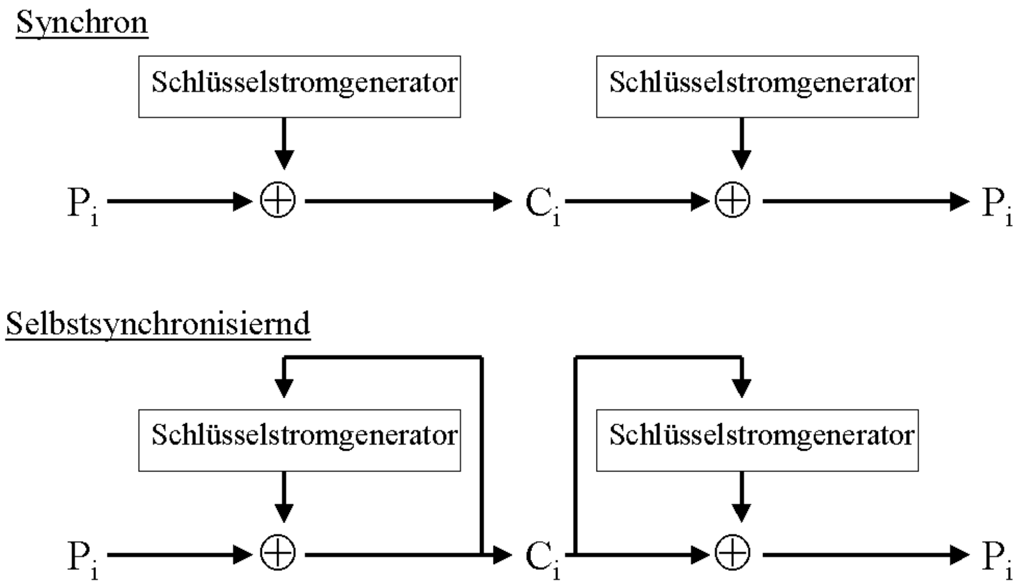

Abbildung 6: Synchrone und selbstsynchronisierende Stromchiffren

Bei **selbstsynchronisierenden Stromchiffren** ist jedes Schlüsselstrom-Bit eine Funktion einer festen Anzahl (*n*) vorhergehender Chiffretext-Bits. Dadurch bekommt die Chiffre die Eigenschaft, dass auch unter Verwendung des gleichen Schlüssels unterschiedliche Daten mit unterschiedlichen Schlüsselströmen kodiert werden. Somit fällt der Angriffsansatz aus ?? weg. Fehlerhaft übertragene oder fehlende Bits führen dazu, dass die nächsten *n* Bits falsch dechiffriert werden, da die Funktion zur Berechnung des Schlüsselstroms auf Sender- und Empfängerseite mit unterschiedlichen Chiffretext-Bits arbeitet. Nach *n* Bits haben sich die beiden aber wieder synchronisiert (deswegen selbstsynchronisierend) und arbeiten wieder korrekt (die Kommunikationspartner haben allerdings keine Möglichkeit zu entscheiden, ab welcher Stelle wieder synchron gearbeitet wird – dieses kann bestenfalls aus den empfangenen Daten geschlossen werden). Dieses eigentlich sehr wünschenswerte Verhalten ermöglicht aber auch einen Angriff auf die Kommunikation durch Wiedereinspielung. Eine dritte Partei, die aus einem früher stattgefundenen Nachrichtenaustausch Chiffretextbits gespeichert hat, kann diese später in einen Nachrichtenaustausch wieder einschleusen. Auf Empfängerseite wird dann für n Bits unbrauchbarer Datenmüll entschlüsselt, danach hat sich der Schlüsselstrom-Generator aber mit den alten Daten synchronisiert und entschlüsselt nun einwandfrei die eingeschleusten Daten. Dies funktioniert natürlich nur, wenn der Schlüssel in der Zwischenzeit nicht geändert wurde.

Ein Beispiel für eine häufig verwendete Stromchiffre ist RC4 (von Ron Rivest, 1987). RC4 ist eine synchrone Stromchiffre mit einer variablen Schlüssellänge und findet unter anderem Einsatz in Lotus Notes und Oracle Secure SQL.

Blockchiffren

Blockchiffren sind wie Stromchiffren symmetrische Chiffrieralgorithmen. Im Unterschied zu den Stromchiffren findet die Verschlüsselung aber nicht bitweise statt, sondern in Blöcken fester Größe (üblich sind im Allgemeinen 64 Bit). Bei einer Blocklänge von 64 Bit werden also immer jeweils 64 Bit Klartext in 64 Bit Chiffretext transformiert. Bei einem Klartext, dessen Größe ungleich einem Vielfachen der Blockgröße ist, muß der letzte Block mit Bits aufgefüllt werden ("padding"). Dieses kann durch einfaches Auffüllen mit Nullen oder Einsen stattfinden. Somit hat der Chiffretext nicht unbedingt dieselbe Größe wie der Klartext, was auch ein weiterer Unterschied zu den Stromchiffren ist, wo Klartext- und Chiffretext-Größe in jedem Fall gleich sind. Unterschiedliche Größe von Chiffre- und Klartext ist eine wünschenswerte Eigenschaft, da in manchen Fällen die alleinige Kenntnis über die genaue Größe einer Nachricht Hinweise auf ihren Inhalt liefern kann.

Blockchiffren - Betriebsmodi: Blockchiffren können in verschiedenen kryptographischen Modi betrieben werden. Die verschiedenen Modi ermöglichen eine der jeweiligen Situation angepaßte Nutzung der Chiffre, je nachdem ob es z.B. wichtiger ist, dass die Verschlüsselung schnell vonstatten geht oder ob der Gefahr des Entfernens oder Einschleusens von fremden Daten begegnet werden soll. Auch ist es möglich, einen Blockchiffre-Algorithmus so zu benutzen, dass er wie eine Stromchiffre arbeitet, z.B. in Fällen, in denen es wichtig ist, dass Klar- und Chiffretext dieselbe Größe haben. Kryptographische Modi fügen einer Chiffre keine weitere Sicherheit zu; die Sicherheit beruht weiterhin ausschließlich auf dem verwendeten Algorithmus. Im folgenden werden vier häufig verwendete Modi vorgestellt:

- ECB (Electronic Code Book)
- CBC (Chipher Block Chaining)
- CFB (Cipher Feedback)
- OFB (Output Feedback)

Ausgewählte Blockchiffren: Es gibt eine Vielzahl an Blockchiffren, die wohl bekanntesten und zum Teil häufig eingesetzten sind:

- DES (Data Encryption Standard (1977)): Der DES-Algorithmus ist ein klassischer Vertreter der monoalphabtischen, polygraphischen Blockchiffrierungen. Das „National Bureau of Standards " der U.S.A entwickelte diesen Algorithmus in zusammenarbeit mit IBM. DES ist eine 16-Runden Feistel-Chiffre, die mit Blockgrössen von 64 Bit arbeitet und eine Schlüssellänge von 64 Bit benutzt, wobei effektiv nur 56 Bit wichtig sind, da nach jedem 7. Bit ein Parity-Information-Bit folgt. Da im Laufe der Zeit, die 56 Bit als Schlüssel nicht mehr der Sicherheit genügten, verwendet man heute meist 3-fach DES. Hierbei wird der zu veschlüsselnde Block dreimal durch den Algorithmus geschickt, mit unterschiedlichen Schlüsseln, abwechselnd ver- und entschlüsselt.

- AES (Advanced Encryption Standard (2000)): 1997 startete das „National Institute of Standards and Technology " (NIST) ein Aufruf, kryptographische Algorithmen einzureichen, um einen Nachfolger für DES zu bestimmen, den AES. 15 Algorithmen wurden eingereicht. Auswahlkriterien waren vor allem Sicherheit, Performanz und Flexibilität. Im Jahre 2000 wurde dann ein Algorithmus von Joan Daemen und Vincent Rijmen,der Rijndael, ausgewählt. Der Rijndael-Algorithmus arbeitet mit variablen Blockgrössen und Schlüssellängen. Die beiden Parameter müssen ein Vielfaches von 32 Bit sein. Zur Zeit wird die Sicherheit des AES-Algorithmus in Krytpologie-Fachkreisen heftig diskutiert[3].

- IDEA (International Data Encryption Algorithm (1992)): Der IDEA-Algorithmus ist eine Weiterentwicklung des PES' (Proposed Encryption Standard). IDEA verschlüsselt den Klartext in Blöcken von 64 Bit und benutzt dabei einen 128 Bit großen Schlüssel. Die Verschlüsselung erfolgt in 8 Runden und ist insgesamt etwa doppelt so schnell wie der DES. IDEA findet unter anderem in PGP (Pretty Good Privacy) Verwendung.

[3]Wurden Blockchiffren in der Kryptoanalyse bisher durch statistische Angriffe (differenzielle-/lineare Kryptoanalyse) angegangen, so wird im Moment eine völlig neue Möglichkeit der Kryptoanalyse diskutiert.

Basierend auf dem XL-Algorithmus von N. Courtois, A. Shamir, J. Patarin und A. Klimov [CSPK00] , entwickelten die Mathematiker N. Courtois und J. Pieprzyk einen erweiterten Algorithmus XLS [CP02]. Das besondere an diesem Verfahren ist, dass es sich um einen algebraischen Ansatz handelt. Zunächst wird der Blockalgorithmus als geschlossene Formel dargestellt. Dies hatten N. Ferguson, R. Schroeppel und D. Whiting 2001 für AES bewerkstelligt [FSW01]. Die geschlossene Formel beinhaltet eine Billiarde Summanden. Damit wird anschliessend ein quadratisches Gleichungssystem erstellt. Bei AES besteht dieses aus 8000 Gleichungen mit 1600 Variablen. Dieses Gleichungssystem hat bestimmte Eigenschaften:

- Es ist überbestimmt.
- Es ist schwach besetzt (die meisten Koeffizienten sind null)
- Es hat eine annähernd reguläre Struktur

Der Streitpunkt dieses Verfahrens ist die Möglichkeit des Lösens dieses Gleichungssystems. Sind nämlich z.B. nicht genug Koeffizenten des Gleichungssystem unabhängig, scheint das Verfahren nicht zu funktionieren. Und selbst wenn das Verfahren funktionieren sollte, ist es zur Zeit immer noch zu umfangreich um diesen Angriff in der Praxis zu benutzen. Die Frage bleibt aber, ob durch diese neue Herangehenweise nicht noch effektivere Verfahren gefunden werden.

2.5.5 Asymmetrische Chiffren

Definition Asymmetrische Chiffren verfolgen einen anderen Ansatz als die oben erläuterten symmetrischen Chiffren. Es wird nicht der gleiche Schlüssel sowohl zu Ver- wie auch zur Entschlüsselung verwendet,sondern es gibt bei asymmetrischen Chiffren zwei oder mehr verschiedene Schlüssel[4]. Im Fall von asymmetrischen Chiffren mit zwei Schlüsseln spricht man oft auch von *public key Algorithmen*, da üblicherweise ein Schlüssel veröffentlicht wird und ein Schlüssel geheim bleibt. Bei einer Kommunikation dient der öffentliche Schlüssel (*public key*) zum Chiffrieren der Daten und der geheime Schlüssel (*private key*) zum Dechiffrieren. So ist es möglich, dass jeder dem Besitzer des geheimen Schlüssels verschlüsselte Nachrichten zukommen lassen kann, ohne dass vorher ein gemeinsamer Schlüssel vereinbart werden mußte. Man verwendet einfach den öffentlichen Schlüssel der betreffenden Person, welcher auch auf unsicheren Kanälen übermittelt werden kann und verschlüsselt die zu sendenden Daten mit diesem. Der so entstandene Chiffretext kann einzig und allein mit dem geheimen Schlüssel dechiffriert werden, der nur dem Empfänger bekannt sein darf. Damit dies auch funktioniert, muß aber gewährleistet sein, dass durch Kenntnis des öffentlichen Schlüssels und des Algorithmus keine Möglichkeit entsteht, den geheimen Schlüssel zu ermitteln. Mit anderen Worten: Es muß unmöglich (bzw. unverhältnismäßig schwer) sein, unter Kenntnis des öffentlichen Schlüssels den geheimen Schlüssel zu berechnen.

Die Rollen (öffentlich/privat) der Schlüssel sind nicht bei allen Algorithmen von vornherein festgelegt, sondern häufig ist es egal mit welchem Schlüssel ver- und mit welchem entschlüsselt wird.

RSA: Der wohl am weitesten verbreitetste public-key-Algorithmus ist RSA[5]. Anders als bei symmetrischen Chiffren wie dem DES, die auf geschickter Bit-Ersetzung und Permutation beruhen, ist das Funktionsprinzip des RSA ein rein zahlentheoretisches. Seine Schlüssellänge ist variabel, wobei Größen zwischen 512 und 1024 Bit üblich sind. Der RSA ermöglicht sowohl Verschlüsselung wie auch

[4] In diesem Aufsatz werde ich mich in meiner Beschreibung allerdings auf asymmetrische Chiffren mit zwei Schlüsseln beschränken

[5] benannt nach seinen Entwicklern Ron Rivest, Adi Shamir und Leonard Adleman (1977)

digitale Signatur (siehe 2.5.6) und hat sich weltweit zu einem Quasi-Standard für asymmetrische Verschlüsselung entwickelt.

RSA – Funktionsweise Der Algorithmus des RSA ist verblüffend einfach, deswegen will ich ihn kurz skizzieren.

Schlüsselgenerierung:

1. Wähle zwei etwa gleich große Primzahlen p,q.
2. Diese bestimmen den Modul $n = p * q$.
3. Wähle Zahl e mit ggt$(e, (p-1)(q-1)) = 1$.
4. Wähle Zahl d mit $e * d \bmod (p-1)(q-1) = 1$.
5. e und n werden öffentlich gemacht und bilden den public key.
6. d, p und q bleiben geheim (p und q werden üblicherweise aus Sicherheitsgründen nach der Schlüsselerzeugung gelöscht). d bildet den private key.

Die Ver- und Entschlüsselung funktionieren wie folgt:

Verschlüsselung: $C = M^e \bmod n$

Entschlüsselung: $M = C^d \bmod n$

Die Reihenfolge der Anwendung der Schlüssel ist beliebig, was aus der Kommutativität der Multiplikation folgt ($M = C^d = (M^e)^d = M^{ed} = M^{de} = (M^d)^e = C'^e = M$). Dies ist jedoch eine spezifische Eigenschaft des RSA.

Andere Ansätze für asymmetrische Chiffren: Neben dem oben beschriebenen Ansatz für asymmetrische Chiffren, deren Sicherheit auf der Schwierigkeit des Faktorisierens großer Zahlen beruht, existieren weitere Ansätze für asymmetrische Chiffren, in denen andere, nur mit großem Aufwand zu lösende Probleme, als Grundlage benutzt wurden:

- *Berechnen des diskreten Logarithmus über endlichen Gruppen*: Ähnlich wie das Problem des Faktorisierens ist das Berechnen des diskreten Logarithmus über endlichen Gruppen ein hartes Problem. Asymmetrische Chiffren wie z.B. der ElGamal-Algorithmus benutzen als Gruppe die ganzen Zahlen modulo einer Primzahl. Für diese Gruppe hat das Problem subexponentiellen Aufwand.

- *Lösen von NP-Vollständigen Problemen*: Es existieren mehrere Ansätze, NP-Vollständige Probleme für asymmetrische Chiffren zu verwenden. Hierbei versucht man das Problem so zu modifizieren, dass es mit Hilfe einer zusätzlichen Information einfach zu lösen ist, ohne die Information der Aufwand aber weiterhin NP-Vollständig bleibt. Man spricht in solchen Fällen auch davon, dass man eine Hintertür in das Problem einbaut (trapdoor). Das modifizierte Problem bildet den öffentlichen Schlüssel, die Zusatzinformation den geheimen. Die zu verschlüsselnden Daten werden in eine Instanz (bzw. in eine Aufgabestellung) des NP-Vollständigen Problems transformiert. Um die Daten zurückzugewinnen, muß man das Problem lösen. Das Lösen des Problems ist ohne die Kenntnis der Zusatzinformation (also dem geheimen Schlüssel) ab einer gewissen Größe des Problems unmöglich, bei Kenntnis der Information aber im Idealfall mit linearem Zeitaufwand zu erledigen. Leider wurde bis jetzt noch kein überzeugender Algorithmus dieses Ansatzes gefunden.

- *Elliptische Kurven*: Asymmetrische Chiffren, deren Sicherheit auf der Schwierigkeit des Lösens des diskreten Logarithmus über endlichen Gruppen beruht, benutzen als Gruppe i.A. die ganzen Zahlen modulo einer Primzahl. Diese Gruppe hat leider einige Besonderheiten, durch die das Berechnen des diskreten Logarithmus' nur subexponentiellen Aufwand hat. Um ausreichende Sicherheit zu bieten, ist es nötig, relativ große Schlüssel zu verwenden. Dieses wiederum erschwert den Einsatz solcher Chiffren in Anwendungen, die nur über beschränkten Speicherplatz verfügen, wie es z.B. bei Chipkarten der Fall ist. Darüber hinaus wird ein Zusammenhang zwischen dem Problem der Faktorisierung und dem des diskreten Logarithmus vermutet. Falls sich dieses als wahr herausstellt und darüber hinaus ein Durchbruch im Lösen eines der beiden Probleme erzielt wird, werden mit einem Schlag fast alle verwendeten asymmetrischen Chiffren nutzlos.

  Mit den elliptischen Kurven, deren Punkte eine "Abelsche Gruppe" bilden, glaubt man, einen potenten Ersatz für die Gruppe der ganzen Zahlen modulo einer Primzahl gefunden zu haben. Das so gewonnene Problem hat exponentiellen Aufwand und kommt somit mit kleineren Schlüsseln aus. Darüber hinaus ist das Problem des Lösens des Logarithmus über einer elliptischen Kurve unabhängig vom Problem des Faktorisierens großer Zahlen. Weiter kommt positiv hinzu, dass die bekannten und bereits als sicher bewerteten Algorithmen, die momentan noch die Gruppe der ganzen Zahlen modulo einer Primzahl verwenden, auch mit der Gruppe der Punkte einer elliptischen Kurve funktionieren. Der verbreiteten Benutzung dieses Ansatzes steht aber noch im Weg, dass die Schlüsselgenerierung für derartige Algorithmen ein noch nicht befriedigend gelöstes Problem ist.

Als Beispiel für einen Algorithmus, der erfolgreich mit elliptischen Kurven als Gruppe arbeitet, sei hier der *Elliptic Curve Digital Signature Algorithm (ECDSA)* (1992) genannt. Wie der Name schon vermuten lässt, handelt es hierbei um eine Variante des DSA (siehe 2.5.6). ECDSA wurde 1999 als ANSI-Standard (ANSI x9.62) akzeptiert.

2.5.6 Weitere notwendige Algorithmen

Einweg-Hashfunktionen Ein wichtiges Werkzeug, das in vielen kryptographischen Protokollen Verwendung findet, sind *Einweg-Hashfunktionen*. Eine Hashfunktion H erzeugt aus einem beliebig großen Datenblock D einen eindeutigen Hashwert $h = H(D)$ mit fester Länge m. Hashfunktionen finden auch in anderen Bereichen der Informatik Verwendung. Um für kryptographische Anwendungen zweckmäßig zu sein, muß aber noch mehr von der Funktion gefordert werden: Zu einem gegebenen Datenblock D muß es einfach sein, den Hashwert h zu berechnen, aber die Umkehrung, aus einem Hashwert h' einen Datenblock D' zu generieren mit $h' = H(D')$, sollte berechnungsmäßig unmöglich sein. Daher kommt auch die Bezeichnung "Einweg"-Hashfunktion. Ebenfalls berechnungsmäßig unmöglich sollte es sein, zwei Datenblöcke D und D' mit dem selben Hashwert $h = H(D) = H(D')$ zu finden. Daraus resultiert, dass auch kleinste Änderungen am Text zu unterschiedlichen Hashwerten führen müssen (Diffusion).

Authentifizierung ohne Verschlüsselung mittels Einweg- Hashfunktionen Einweg-Hashfunktionen eignen sich für eine einfache Möglichkeit zur Authentifizierung von Kommunikationspartnern. Die Voraussetzung dafür ist, dass die an der Kommunikation beteiligten Parteien ein Geheimnis G teilen. Zur Erläuterung des Vorgehens betrachte man folgende Situation: Alice will Bob überzeugen, dass eine Nachricht M von ihr stammt. Um dieses zu tun, bildet Alice $h = H(M, G)$ also den Hashwert der Konkatenation der Nachricht und des Geheimnisses, wobei H eine Einweg-Hashfunktion ist, auf die sich Alice und Bob im Vorfeld geeinigt haben und G das gemeinsame Geheimnis. Alice sendet h zusammen mit M an Bob. Bob, der G kennt, kann ebenfalls $H(M, G)$ bilden und das Ergebnis mit h vergleichen. Wenn die Werte übereinstimmen, weiß Bob, dass die Nachricht M wirklich von Alice geschickt wurde. Da nur er und Alice das Geheimnis G kennen, hätte niemand außer Alice den Hashwert h generieren können, und da H eine Einweg-Hashfunktion ist, ist es für dritte auch trotz der Kenntnis von h und M unmöglich G zu berechnen. Bob weiß außerdem, dass der Text M unverändert angekommen ist. Hätte eine dritte Partei die Nachricht M durch eine andere Nachricht M' ersetzt, wäre der resultierende Hashwert $h' = H(M', G)$ unterschiedlich zu dem mitgeschickten $h = H(M, G)$. Diese Art der Authentifizierung findet u.a. bei IPSec Verwendung.

Der Nachteil dieses Verfahrens ist, dass es Dritten gegenüber keine Authentifizierung bietet („non-repudiation“). Alice kann sich zwar von der Identität Bobs überzeugen, hat aber keine Möglichkeit einer dritten Person Clara zu beweisen, dass die Nachricht tatsächlich von Bob stammt, da Clara weder das Geheimnis G kennt, noch, bei Offenlegung des Geheimnisses, aus G auf Bob schließen kann.

Einweghashfunktionen – Beispiele Hier sollen zwei verbreitete Einweghashfunktionenkurz erwähnt werden, die als weitgehend sicher gelten:

- Secure Hash Algorithm, SHA1 (NIST, 1994): SHA1 ist als Teil des DSA (siehe 2.5.6) von der NSA entwickelt worden und berechnet einen Hashwert von 160 Bit.

- RIPE-MD 160, Europäische Union: RIPE-MD 160 ist eine sichere Weiterentwicklung von MD4 und berechnet 160 Bit große Hashwerte.

Digitale Signaturen Für viele Kommunikationsprotokolle hätte man gerne ein elektronisches Äquivalent zur menschlichen Unterschrift, also eine Möglichkeit, digitale Dokumente zu unterzeichnen. Die exakten Anforderungen an eine sogenannte digitale Signatur lassen sich wie folgt auflisten:

- *Authentizität*: Der Empfänger kann sich von der Identität des Unterzeichners überzeugen (Es muß für jede Person problemlos möglich sein festzustellen, von wem die digitale Signatur erstellt wurde. Die digitale Signatur identifiziert ihren Ersteller eindeutig).
- *Fälschungssicherheit*: Nur dem Unterzeichner ist es möglich, die Signatur zu erzeugen.
- *Überprüfbarkeit*: Eine dritte Partei kann jederzeit die Signatur verifizieren.
- *Keine Wiederverwendbarkeit*: Die Signatur bezieht sich nur auf das unterzeichnete Dokument und kann keinesfalls für andere Dokumente verwendet werden.
- *Keine Veränderbarkeit, Integrität*: Nachdem das Dokument unterzeichnet ist, kann es nicht mehr verändert werden.

Es ist ersichtlich, dass, wenn alle diese Forderungen erfüllt sind, Vorgänge wie digitales Unterzeichnen von Verträgen und ähnliches möglich sind. Digitale Signaturen kann man weiterhin zur Authentifizierung von Kommunikationspartnern oder Autoren digitaler Dokumente einsetzen.

Digitale Signaturen mittels asymmetrischer Chiffren Vorausgesetzt die Reihenfolge der Anwendungen des geheimen und öffentlichen Schlüssels ist unerheblich, eignen sich asymmetrische Chiffren gut, um digitale Signaturen zu erstellen. Dabei wird wie folgt vorgegangen: Alice chiffriert das betreffende Dokument mit ihrem geheimen Schlüssel. Danach sendet sie das so unterzeichnete Dokument an Bob. Bob dechiffriert das Dokument mit Alices öffentlichem Schlüssel. Betrachten wir kurz, ob alle unsere Anforderungen an digitale Signaturen bei einem derartigen Vorgehen erfüllt werden:

- *Authentizität*: Da Alices öffentlicher Schlüssel den Klartext ergibt, weiß Bob, dass das Dokument von Alice stammt.
- *Fälschungssicherheit*: Nur Alice kennt ihren geheimen Schlüssel. Niemand anderes hätte die Signatur erstellen können.
- *Überprüfbarkeit*: Jeder, der Alices öffentlichen Schlüssel kennt, kann die Signatur bestätigen. Alice kann sie nicht zurücknehmen.
- *Keine Wiederverwendbarkeit*: Die Unterschrift bezieht sich nur auf dieses Dokument, andere Dokumente ergeben andere Signaturen.
- *Keine Veränderbarkeit, Integrität*: Bei Veränderung des Dokuments ergibt die Dechiffrierung mit Alices öffentlichem Schlüssel kein sinnvolles Ergebnis.

In der Praxis wird oftmals nicht das gesamte Dokument unterzeichnet, (also chiffriert) sondern nur der Hashwert des Dokuments. Dieses geschieht vor allem aus Gründen der Effizienz (außerdem bleibt so das Dokument selbst, auch ohne Überprüfung der digitalen Signatur, lesbar). Man kann sich davon überzeugen, dass bei Benutzung einer Einweg-Hashfunktion mit einem ausreichend großem Hashwert sich die Sicherheit der digitalen Signatur dabei nicht ändert.

Digitale Signaturen mittels DSA Bei digitalen Signaturen mit asymmetrischen Chiffren geht mit der Signierung eine Verschlüsselung des Dokuments einher (falls das ganze Dokument und nicht nur der Hashwert signiert wird). Dies ist nicht in allen Fällen wünschenswert. Weiterhin muß man beachten, dass in manchen Ländern Benutzung oder Export von Software, die Verschlüsselung ermöglicht, restringiert ist. Also gibt es einen Bedarf nach Algorithmen, die digitale Signaturen ohne einhergehende Verschlüsselung ermöglichen. Der Digital Signature Algorithm (DSA), der 1991 von der NSA entwickelt wurde und Teil des DSS (Digital Signature Standard) ist, verfolgt einen derartigen Ansatz. Bei korrekter Implementation eignet er sich nur zum Signieren, aber nicht zum Verschlüsseln. Auch beim DSA wird mit öffentlichen und geheimen Schlüsseln gearbeitet und die Signatur wird auch hier unter Verwendung des öffentlichen Schlüssels überprüft.

Die Sicherheit des DSA beruht auf der Schwierigkeit, den diskreten Logarithmus über endlichen Körpern zu berechnen. Die Schlüssellänge ist variabel zwischen 512 und 1024 Bit. Wie auch schon beim DES wurde dem DSA bei seiner Einführung mit Mißtrauen begegnet, da an seiner Entwicklung die NSA beteiligt war. Tatsächlich besteht bei maliziöser Implementation des Algorithmus die Möglichkeit, einen verdeckten Kanal in den DSA einzubauen, der nach und nach den geheimen Schlüssel des Anwenders nach außen schleust. Da ein solcher Kanal unmöglich zu entdecken ist, sollte man nur vertrauenswürdige Implementationen des DSA verwenden.

2.5.7 Weiteres wichtiges Zubehör

Für viele kryptographische Anwendungen und Protokolle benötigt man neben den bereits vorgestellten Werkzeugen wie Chiffren und Einweg-Hashfunktionen noch weiteres Handwerkszeug.

- *Zufallszahlengeneratoren*: Als erstes seien hier die kryptographisch starken Zufallszahlen-Generatoren genannt. In vielen Anwendungen und Protokollen werden Schlüssel zur Datensicherung oder Kommunikation aus Zufallszahlen gewonnen. Für diese Fälle benötigt man einen Zufallszahlen-Generator, dessen Ausgaben nicht vorhersehbar sind. Genauer gesagt muß es unmöglich sein, ohne Kenntnis des Initialwertes des Generators, ausschließlich aus Kenntnis der vorherigen Zufallszahlen und des Algorithmus, den nächsten Zufallswert vorherzusagen. Falls ein Algorithmus Verwendung findet, der dieses nicht leistet, erschließt sich, z.B. in einer Situation, in der die Zufallszahl zur Schlüsselgenerierung benutzt wird, ein möglicher Angriffspunkt auf die Verschlüsselung. Anstatt die Chiffre oder den Schlüssel angreifen zu müssen, können potentielle Angreifer versuchen, den verwendeten Schlüssel zu „erraten“. Falls sie in Besitz von vorher verwendeten Schlüssel-Informationen sind und der Zufallszahlen-Generator noch nicht neu initialisiert wurde, ist ein derartiges Vorgehen äußerst vielversprechend.

- *Synchronisierte Uhren*: Viele kryptographische Protokolle benutzen Zeitstempel, um Angriffe, die durch das Wiedereinspielen von abgefangenen Daten stattfinden, abzuwehren. Die einzelnen Nachrichten eines Protokolls werden mit einem Zeitstempel, der Informationen über Datum und Uhrzeit enthält, versehen. Wenn eine Nachricht eintrifft, deren Zeitstempel älter als eine festgelegte Schranke ist, wird diese Nachricht nicht mehr akzeptiert. Ein solches Verfahren ist nur möglich, wenn die Teilnehmer an dem Protokoll über synchronisierte Uhren verfügen. Doch dieses ist kein triviales Unterfangen, besonders bei Kommunikation, die über die Grenzen eines kleineren Netzwerkes hinausgeht.

- *Hochauflösende Zeitstempel*: Die Notwendigkeit für hochauflösende Zeitstempel hängt eng mit dem Punkt der synchronisierten Uhren zusammen. Viele Betriebssysteme liefern nur sehr grob gemessene Zeitangaben. Für manche Protokolle ist es aber nötig, Unterschiede in Zeitstempeln, die in den Bereich von Zeiträumen unter einer Nanosekunde fallen, zu erkennen. Dafür reichen dann oft die vom Betriebssystem zur Verfügung gestellten Hilfsmittel nicht aus.

2.5.8 Kryptographische Protokolle

Ein kryptographisches Protokoll ist ein Kommunikationsprotokoll, das mindestens ein Sicherheitsziel wie beispielsweise Authenzität, Vertraulichkeit oder Integrität der übertragenen Daten gewährleisten kann. Es definiert eindeutig eine festgelegte Abfolge von Handlungen der Kommunikationspartner. Dabei wird, je nach Aufgabe des Protokolls, bestimmt, welches Format die verschickten Nachrichten haben, welche Algorithmen verwendet werden, wie die Schlüssel ausgetauscht werden und ähnliches. Dabei finden üblicherweise mehrere kryptographische Algorithmen Verwendung. Man findet kryptographische Protokolle in allen Schichten der TCP/IP-Familie. Es seien als Beispiele genannt, *TLS* und *SSL* in der Transportschicht und *SSH*, *Kerberos* und *PGP* in der Anwendungsschicht.

3 Konzept und Implementation

Ziel dieser Arbeit ist die Realisierung eines gesicherten Kanals für die Übertragung von biometrischen Daten. Hierbei gehe ich von mehreren dezentralen Erfassungseinheiten und einer zentralen Authentisierungseinheit aus.

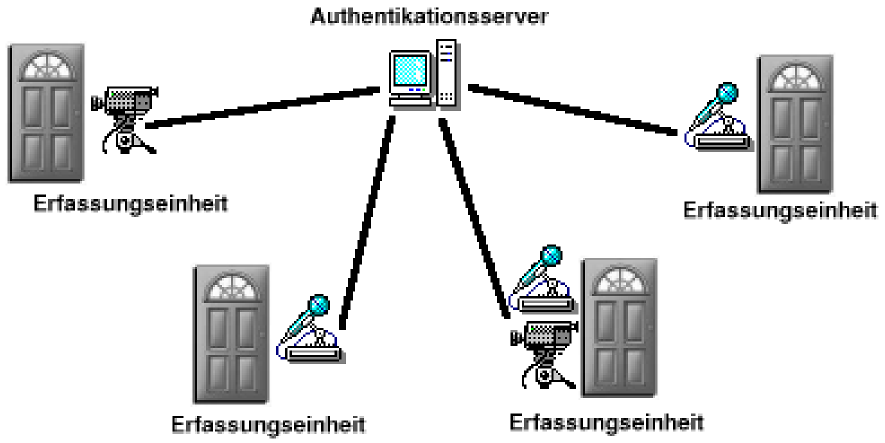

Abbildung 7: Erfassungseinheit und Server

Die Erfassungseinheit zeichnet bestimmte biologische Charakteristika auf und betreibt eventuell eine Vorverarbeitung der zu übertragenden Daten.

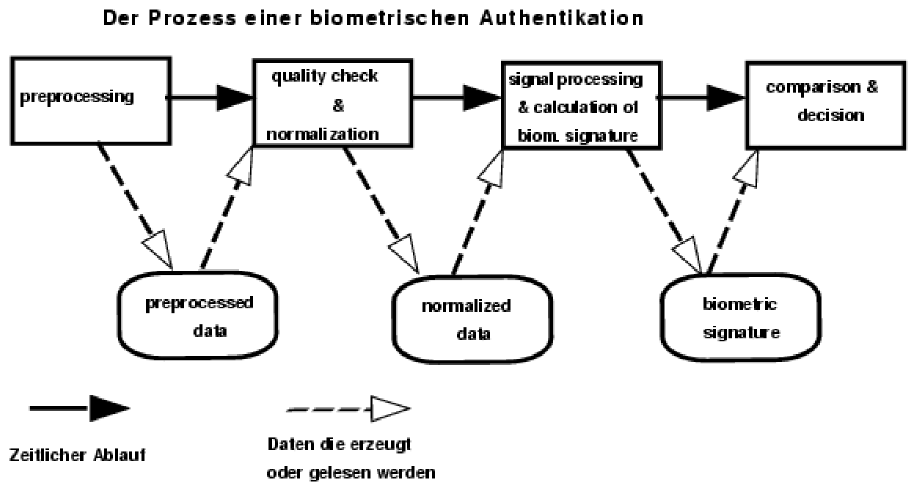

Abbildung 8: Biometrischer Authentikations Prozess (nach [Barg02-ACM])

Der Prozess des Aufzeichnens, sowie eventuelle Vorverarbeitungsschritte betrachte ich nicht tiefergehend, lediglich notwendige Protokollschritte müssen untersucht werden. Diese aufgezeichneten und eventuell weiter aufbereiteten Daten werden anschliessend über ein IP-Netzwerk zu einem Authentisierungsserver übertragen. Der Authetisierungsserver führt einen Abgleich mit seinen vorhanden Referenzdaten durch und gibt ein Feedback zu der Erfassungseinheit zurück. Der Abgleich und die daraus abgeleiteten Folgeschritte, wie Zugang zu Objekten oder Ressourcen, werden nicht in dieser Arbeit behandelt. Auch die Verwaltung der Referenzdaten, sowie der Zugriff auf diese werden als gegeben angesehen. Natürlich wird die Art und Funktionalität der Erfassungseinheit auf das zu erstellende Protokoll Einfluss haben. Das Szenario, bei dem der Abgleich der aufgenommen Daten mit den Referenzdaten auf einer Smartcard stattfindet, wird bei dieser Arbeit nicht betrachtet.

Somit kann der Fokus dieser Arbeit auf die folgenden Punkte begrenzt werden:

- Verbindungsaufbau
- Datenübertragung
- Authentikationsergebnisübertragung
- Verbindungsabbau.

Wobei natürlich Überlegungen zu den oben genannten Punkten erfolgen müssen, damit das zu erstellende Protokoll möglichst für einen breiten Einsatz in der Biometrik eingesetzt werden kann.

Das Protokoll soll zudem noch folgenden Bedingungen genügen:

- Authentikation: Der Prozess für die Verifikation der Identitäten der Kommunikationsstationen. Es soll nicht möglich sein, sich einfach als eine Erfassungseinheit oder als den Authentikationsserver auszugeben.
- Datenintegrität: Nachweis der Authentizität und Unversehrtheit der übertragenen Daten. Es soll gewährleistet werden, dass die übertragenen Daten nicht manipuliert wurden. Ausserdem soll sichergestellt sein, dass die Nachrichten auch vom erwarteten Sender stammen.

- Vertraulichkeit: Gewährleistung des Schutzes vor unauthorisierter Offenlegung eines Objektes. Es soll keine andere Instanz die Kommuntikation zwischen der Erfassungseinheit und dem Authentikationsserver mitlesen können.

3.1 Voraussetzung und geforderte Funktionalität

Auch wenn das Protokoll nicht direkt mit der Erfassung der Daten im Zusammenhang steht, muss man sich dennoch diesen Vorgang genauer anschauen. Da sich dort notwendige Protokollzustände ableiten lassen.

3.1.1 Ressourcen der Erfassung von biologischen Charakteristika

Hier wollen wir untersuchen, welche unterschiedlichen biometrischen Verfahren bekannt sind und welcher Art Daten dabei verarbeitet/übertragen werden müssen. Da der Mensch eine Vielzahl von biologischen Charakteristika besitzt und eine erstaunlich grosse Untermenge von diesen sich für die Biometrik eignen, ist es notwendig, zunächst zu untersuchen, was für Daten bei der Aufzeichnung dieser Merkmale anfallen können. Ich unterscheide dabei zwischen *single valued data* und *stream data*. *Single valued data* kann z.B ein Standbild/Foto oder auch nur ein Scan eines Sensors zu einem ganz kurzen Augenblick sein, also ohne die Veränderung durch die Zeit zu betrachten. *Stream data* anderseits impliziert eine Folge von Aufzeichnungen, welche im Zusammenhang mit der Zeit der Aufnahmen stehen. Gerade bei Bewegungserkennung oder Spracherkennung ist diese Korellation wichtig. Folgende wohlbekannten biometrischen Verfahren lassen sich in nachfolgende Tabelle einordnen, die Daten stammen aus: „A Conceptual Framework for Testing Biometric Algorithms within Operating Systems Authentication“ [Barg02-ACM] von der „Biometric Authentication Research Group“ der Universität Hamburg

body part	biological characterristic	video.image	videostream	audiostream	scansingle	scan timeseries
head	face geometry	X	X	-	-	-
	iris	X	X	-	-	-
	retina (veins)	X	X	-	-	-
	voice	-	-	X	-	-
	lip movement	-	X	-	-	-
	dental	X	-	-	X	-
	ear	X	-	-	X	-
	tongue	X	-	-	X	-
hand	handgeometry	X	X	-	X	-
	fingerprint	X	X	-	X	-
	palm print	X	X	-	X	-
cerebric	brain waves (EEG)	-	-	-	-	X
	intuitional acts	-	X	-	-	X
body	DNA	X	-	-	-	X
	gesture	-	X	-	-	-
	odor	-	-	-	X	X
motoric	movement patterns	-	X	-	-	X
	posture	X	X	-	-	-
	signing	X	X	-	X	-

Tabelle 4: Biometrische Aufzeichnungsressourcen [Barg02-ACM]

Zusätzlich zu diesen zu übertragenden Daten, sowohl als abgeschlossener oder kontinuierlicher Datensatz, muss zusätzlich noch die Funktionalität der Erfassungseinheit betrachtet werden.

3.1.2 Die Erfassungseinheit

Die Erfassungseinheit ist zuständig für die Aufzeichnung der Daten, welche aus einem biologischem Merkmal erstellt wird. Dabei kann die Erfassungseinheit verschiedene Grade an Funktionalität bereitstellen. Wenn man z.B die Iris-Biometrie betrachtet, können folgende Szenarien unterschieden werden. Allen ist das Ziel, eine Aufzeichnung der Iris zu erzeugen, gemeinsam. Die notwendigen Schritte können aber variieren. Deswegen unterscheiden wir zunächst die Art der Vorverarbeitung.

- Voll autonome Erfassungseinheit [VAE]: Bei diesem Modell verfügt die Erfassungseinheit nicht nur über einen Sensor für die Abtastung der biologischen Merkmale, sondern auch über eine Prozessoreinheit. Mit dieser Prozesoreinheit kann die Justierung sowie Kalibrierung der Aufnahmeeinheit selbständig erfolgen. Die Erfassungseinheit findet von sich aus das Auge und stellt das System auf mögliche optimale Parameter für die Aufzeichnung ein. Dies beinhaltet die Schärfe und Grösse des zu erstellenden Bildes, sowie eventuell auch die benötigte Belichtung zu justieren. Natürlich kann dieser Prozess sich über mehrere Schleifen hinziehen, bis eine geforderte Qualität erreicht ist. Nachdem das Bild erstellt wurde, findet schon auf der Erfassungseinheit eine Vorverarbeitung der aufgezeichneten Daten statt. Dies beinhaltet die ganze Prozedur der Signaturberechnung aus den aufgezeichneten Daten. Somit wird anschliessend nur noch die erstellte Signatur an den Authentikationsserver übertragen.

- Halb autonome Erfassungseinheit [HAE]: Bei dieser Erfassungseinheit geschieht die Aufnahme und deren notwendige Parametrisierung auch selbständig in der Erfassungseinheit. Nach einer bestimmten qualitätssichernden Aufzeichnung findet aber anschliessend keine weitere Verarbeitung der erstellten Daten statt. Stattdessen werden die gewonnen Daten an den Authentikationsserver übertragen. Dieser führt jetzt den Prozess der Signaturberechnung durch. Es kann sich herausstellen, dass die Aufnahme bestimmten Vorraussetzungen der Signaturerzeugung nicht genügt und somit eine erneute Aufnahme der biologischen Charakteristika erfoderlich wird. Dies bedeutet, dass der Authentikationsserver der Erfassungseinheit mitteilen muss, dass er eine erneute Aufzeichnung benötigt. Idealerweise mit notwenigen Hinweisen für die Erfassungseinheit versehen, um eine höhere Qualitätsstufe zu erreichen.

- Nicht autonome Erfassungseinheit [NAE]: Diese Erfassungseinheit besitzt nur einen Sensor für die Erfassung von biologischen Merkmalen. Die notwendige Parametrisierung dieses Prozesses kann die Erfassungseinheit nicht von sich aus einstellen. Deswegen ist es notwendig, dass die Erfassungseinheit permanent Daten an den Authentikationsserver überträgt. Der Authentikationsserver kann anhand der empfangenen Daten und deren Verarbeitung, die notwendigen Parameter der Erfassungseinheit einstellen, bis diese die erfoderliche Güte der Daten liefert. Bei dem permanenten Datenstrom der Erfassungseinheit kann es sich um abgeschlossene Datensätze handeln, wie bei einzelnen Bildern oder Fingerabdrücken. Es können aber auch kontinuierliche Daten übertragen werden, z.B Videostrom oder Sprachaufzeichnungen.

Dabei leitet sich die mögliche Verwendung von zwei Kanälen ab. Wobei der eine Kanal für die Kontroll- sowie Metainformationen zuständig ist. Der zweite Kanal ist aussschliesslich für die Aufzeichnungsdaten reseviert. Es wird aber nicht unterschieden, ob die Aufzeichnungsdaten schon durch eine Vorverarbeitung gelaufen sind oder nur die rohen Daten des Sensors übertragen werden.

3.1.3 Aufnahmeparametrisierung des Servers

In dem zu erstellenden Protokoll sollte die Möglichkeit bestehen, dass der Authentikationsserver vor dem Aufzeichnungsprozess der Erfassungeseinheit durch die Übertragung von Parametern beeinflussen kann. Diese Parameterübergabe kann für zwei unterschiedliche Aufgaben benutzt werden.

- Auswahl von zusätzlichen Sicherheitsfeaturen: In der Erfassungseinheit kann die Option vorhanden sein, dass die zu erstellende Aufzeichnung von bio-

logischen Charakteristika durch zusätzliche Features eine Erhöhung der Sicherheit erlauben. Diese besteht hauptsächlich in der Form, der Lebenderkennung, oder besser in dem Gewährleisten, dass die zu übertragende Aufzeichnung auch zu diesem Zeitpunkt erstellt wurde.
Mögliche Implementierungen könnten sein, dass Reflektionen auf dem aufgezeichneteten Bild ausgewertet werden, welche bei der Aufnahme durch Lampen an der Erfassungseinheit entstanden sind, die vom dem Authentikationsserver aktiviert wurden. Der Authentikationsserver teilt der Erfassungseinheit vor der Aufzeichnung oder dem Scan mit, welche Lampen leuchten sollen und kann dann anschliessend anhand des Datensatzes, welches er von der Erfassungseinheit bekommen hat, auswerten, ob dies eine aktuelle Aufzeichnung ist.
Es könnte aber auch sein, dass bei einer Spracherkennung, der zu sprechende Satz vom Server vorgegeben wird.

- Einschränkungen des Erfassungsprozesses: Gerade bei multimodalen Erfassungssystemen, also wenn mehrere biologische Merkmale betrachtet werden, kann es notwendig sein, diese einzuschränken. Aber auch bei einem monomondalen System besteht diese Forderung. Nehmen wir zum Beispiel ein Fingerprint-System. Es kann durchaus vorkommen, dass die zu authentisierende Person nicht den Forderungen der Erfassungseinheit nachkommen kann. Vielleicht fehlt der Person der linke Arm und somit kann sie sich nicht mit irgendeinem Finger der linken Hand ausweisen. Sollte aber das System dieses fordern, wäre es in diesem Fall nicht benutzbar. Auch bei einem multimodalen System kann nicht garantiert werden, dass alle erforderlichen biologischen Charakteristika vorhanden sind. Dies könnte auch nur eine temporäre Einschränkung sein, z.B bei Krankheit oder Unfallfolgen.

Bei beiden eben aufgezählten Forderungen besteht natürlich auch die Möglichkeit, diese Entscheidung direkt in der Erfassungseinheit zu realisieren. Dies hat jedoch Nachtteile. Da die Erfassungseinheit notwendigerweise sich im physischen Handlungsbereich der zu überprüfenden Personen befindet, besteht die Gefahr

der Entwendung, Analysierung und Modifikation derselben. Aus diesem Grund sollte die Erfassungseinheit so wenig sensible Daten wie möglich besitzen. Gerade die Lebenderkennung wäre hinfällig, wenn eine modifizierte Erfassungseinheit dieses vorgeben sollte.
Zudem treibt diese zusätzliche Funktionalität der Erfassungeinheit die Kosten der Herstellung in die Höhe und auch die Komplexität der Erstellung wird höher. Ausserdem steigt auch der Verwaltungsaufwand, da die Speicherung der personenbezogenen Daten in dem System nicht an einer zentralen Stelle erfolgt. Es muss diese Information regelmässig auf allen benutzten Erfassungeinheiten repliziert werden.
Es ist also sinnvoll, die notwendigen Daten für diese Funktionalität beim ohnehin besonders gesicherten Server zu hinterlegen und bei Bedarf der Erfassungseinheit nur die für die Aufzeichnung notwendigen Daten zu übertragen.

3.1.4 Rollenbasierende Authentikation

Die Erfassungseinheit sollte zusätzlich zu dem Übertragen der aufgezeichneten Daten auch eine Benutzerkennung sowie die gewünschte Rolle des Anmelders übertragen. Dies kann vor dem Scanprozess interaktiv mit dem Nutzer an der Erfassungseinheit erfolgen oder je nach Einsatzgebiet auch fest vorgegeben in der Erfassungseinheit eingestellt sein. Anhand dieser Daten hat der Authentikationsserver zusätzliche Hinweise, welche nicht nur die Sicherheit erhöhen, sondern auch in Abhängigkeit der gewünschten Rolle den Berechtigungsgrad beeinflussen. Das Erlangen der Benutzerkennung sowie der gewünschten Rolle ist in dieser Arbeit nicht von belang, sondern nur die Möglichkeit der Übertragung dieser Information an den Authentikationsserver.
Anhand dieser Daten kann auch der Authentikationsserver die notwendigen Parameter auswählen, welche für den obigen erwähnten Einsatz von Einschränkungen des Erfassungsprozesses notwendig sind. Dieser Kommunikationsschritt sollte demnach schon am Anfang der Kommunikation übermittelt werden.

3.1.5 Meta- und Kalibrierungsdaten

Obwohl die Erfassungseinheit so simpel wie möglich sein kann oder auch recht komplex, ist es eventuell durchaus notwendig, vor dem Begin der Aufzeichnung und Übertragung der Daten, dem Authentikationsserver bestimmte Informationen mitzuteilen. Dabei bezieht sich diese Kommunikation auf den Aufnahmeprozess. Dabei kann man folgende Unterscheidungen treffen.

- *voll autonome Erfassungseinheit*: Hier justiert die Erfassungseinheit das Aufzeichnungssystem selbständig und versucht somit ein optimales Abbild der gewünschten Information zu erzeugen. Dennoch kann es notwendig sein, dass der Authentikationsserver zusätzliche Informationen benötigt, um die Daten dann anschliessend auszuwerten oder um bestimmte Einstellungen für einen erneuten Scan zu beeinflussen. Ausserdem kann es hilfreich sein, bestimmte Parameter des Aufzeichnungsprozesses dem Server mitzuteilen. Dabei handelt es sich um Kontextdaten, welche sich auf die Aufzeichnung und der daraus resultierenden Daten beziehen.

- *halb/nicht autonome Erfasungseinheit*: Zusätzlich zu der oben genannten Forderung muss gerade bei minimal funktionalen Erfassungseinheiten vor und während der Aufzeichnung, der Authentikationsserver die Möglichkeit haben, die Parameter der Aufzeichnung zu verändern. Dies muss eventuell parallel während der Übertragung der Aufzeichnungsdaten geschehen, um das Erfassungssystem für ein nutzbares Ergebnis einzustellen.

Es sollte also zumindest vor dem eigentlichen Übertragen der aufgezeichneten Daten ein Protokollschritt existieren, welcher die benutzten Meta- und Kalibrierungsdaten dem Server übermittelt. Abhängig von der verwendeten Erfassungseinheit kann dann noch zusätzlich zum Aufzeichnungsstrom eine Kalibrierungskommunikation zwischen Authentikationsserver und Erfassungseinheit stattfinden. Hierbei sollte es keine Rolle spielen, ob die Aufzeichnungsdaten abgeschlossen (Fotos) oder offen (Videostream) sind.

3.1.6 Authentikationsergebnisübertragung

Zum Abschluss des Anmelde-/Zugangsprozesses meldet der Server der Erfassungseinheit das Ergebnis der Überprüfung zurück. Diese besteht entweder in der positiven Verifikation der sich authentisierenden Person oder in einer Ablehnung des Vorgangs. Egal welche der beiden Möglichkeiten zurückgesendet wird, begeben sich die beiden Kommunikationsstellen wieder in ihren Ausgangszustand und sind somit bereit für eine Wiederholung des gesamten Vorganges oder warten auf einen neuen Vorgang mit einer anderen Person.

3.2 Konzept der Implementation.

Die Implementation erfolgt unter Linux. Für die gesicherte Verbindung wird SSL verwendet. Das zu erstellende Protokoll baut dann auf eine bestehende SSL-Verbindung auf. Somit wird mit der SSL-Protokollschicht ein sicherer Kanal zu den Kommunikationsteilnehmer aufgebaut. Dieser sichere Kanal tunnelt die höheren Protokolle und überträgt diese dann zu dem entfernten Teilnehmer über ein möglicherweise unsicheres Netzwerk. Für die höherliegenden Protokolle ist dieser Tunnel transparent.

3.2.1 SSL/TLS

Die „Secure Sockets Layer" (**SSL**) Protokoll-Familie ist ursprünglich bei Netscape entstanden. Die aktuelle Version 3 des SSL-Protokolls ist auch als „Internet Draft" veröffentlicht [SSLv3 Draft] und die IETF hat daraus den allgemeinen Standard TLS 1.0 [TLS-RFC] entworfen, auch bekannt als „Transport Layer Security" (**TLS**). Diese erste Version von TLS kann als SSLv3.1 angesehen werden und ist sehr eng an SSLv3 angelehnt. Deswegen wird hier im Allgemeinen SSL sowohl für SSLv3, als auch synonym für TLS benutzt. Am Ende von dieses Unterkapitels, werden kurz die Unterschiede zwischen SSLv3 und TLS erläutert.

SSL Architektur SSL ist eine Protokollschicht, welche sich direkt über die Transportschicht im TCP/IP-Protokollstapel setzt und sich somit unter der Applikationsschicht befindet. Dieses Design hat den Vorteil, dass SSL unabhängig von dem verwendeten Applikationsprotokoll eine sichere Punkt-zu-Punkt Verbindung herstellen kann. SSL besteht nicht aus einem einzelnem Protokoll, sondern besitzt zwei Protokollschichten.

SSL Handshake Protocol	SSL Change Cipher Spec Protocol	SSL Alert Protocol	Anwendungs-protokoll z.B HTTP
SSL Record Protocol			
TCP			
IP			

Tabelle 5: SSL Protokoll-Stack

Zwei wichtige Konzepte von SSL sind die SSL-Session und die SSL-Connection.

SSL-Session Eine SSL-Session ist eine Assoziation zwischen einem Client und einem Server. Sessions werden über das Handshake-Protokoll aufgebaut. Eine Session definiert eine Menge von kryptologischen Sicherheitsparametern, welche gemeinsam über mehrere Verbindungen genutzt werden können. Der Sinn hinter Session besteht darin, nicht jedesmal eine neue zeitaufwendige Verhandlung der Sicherheitsparameter auszuführen.

SSL-Connection Eine SSL-Connection ist ein Transportkanal, welcher einrn spezifizierten Service bietet. Bei SSL handelt es sich um Peer-to-Peer Verbindung, welche nur vorübergehend aufgebaut sein kann. Jede Connection muss mit einer Session assoziert sein.

Somit können aus einer Session mehrere Connections aufgebaut werden, welche auch parallel betrieben werden können.

SSL-Sessions besitzen unterschiedliche Zustände. Ein Sessionzustand ist definiert über folgende Parameter [SSLv3 Draft]:

- **Session Identifier**: Eine zufällige Bytesequenz, vom Server erzeugt, um eine aktive oder wiederherstellbare Session zu identifizieren.

- **Peer certificate**: Ein X509.v3 Zertifikat des Clients. Dieses Element kann Null sein.

- **Compression method**: Der Kompressionsalgorithmus, welcher vor der Verschlüsselung benutzt wird.
- **Cipher spec**: Spezifiziert den Verschlüsselungsalgorithmus, sowie den Hashalgorithmus für den MAC. Zusätzlich werden noch Attribute, wie z.B Hashlänge, definiert.
- **Master secret**: 48-Byte Geheimnis, welches vom Client und Server benutzt wird.
- **Is resumable**: Ein Flag, welches anzeigt, ob diese Session für neue Verbindungen genutzt werden kann.

Auch die Connections definieren unterschiedliche Zustände

- **Server and client random**: Byte-Sequenzen, welche vom Server und Client erzeugt werden, für jede Verbindung.
- **Server write MAC secret**: Der geheime Schlüssel, der bei MAC-Operationen von Daten, die zum Server gehen, benutzt wird.
- **Client write MAC secret**: Der geheime Schlüssel, der bei MAC-Operationen von Daten, die zum Client gehen, benutzt wird.
- **Server write key**: Der symmetrische Schlüssel, der zur Verschlüsselung vom Server und bei der Entschlüsselung vom Client benutzt wird.
- **Client write key**: Der symmetrische Schlüssel, der zur Verschlüsselung vom Client und bei der Entschlüsselung vom Server benutzt wird.
- **Initialization vectors**: Wenn ein Blockchiffre in CBC-Mode benutzt wird, so wird ein „initialization vector (IV) " für jeden Schlüssel beibehalten. Dieses Feld wird zum ersten Mal beim SSL Handshake Protokoll initialisiert. Danach wird der letzte Ciphertextblock von jedem Record, als IV benutzt für den nächsten Record.
- **Sequence numbers**: Beide Parteien verwalten eigene Sequenznummern für die gesendeten und empfangenen Nachrichten jeder Verbindung. Falls

einer der Parteien ein „Change cipher spec“ verschickt oder erhält, so wird die Sequenznummer wieder auf Zero gesetzt. Sequenznummern können nicht größer als $2^{64} - 1$ werden.

SSL Record Protokoll Das SSL Record Protokoll stellt zwei Dienste für SSL-Verbindungen zur Verfügung

- **Vertraulichkeit**: Das Handshake Protokoll handelt einen gemeinsamen geheimen Schlüssel aus, welcher für die symmetrische Verschlüsselung der SSL Nutzdaten benutzt wird.
- **Nachrichten Integrität**: Das Handschake Protokoll definiert zudem auch einen gemeinsamen geheimen Schlüssel, welcher für den „Message Authentication Code“ (MAC) benutzt wird.

Das SSL Record Protokoll nimmt eine für die Übertragung vorgesehene Applikationsnachricht an und erstellt daraus ein SSL-Paket. Zunächst wird das Paket fragmentiert und in die notwendige Blöckgröße zerlegt/aufgefüllt. Optional wird dann dieser Block komprimiert. Anschliessend wird ein MAC angehangen. Dieser neue Block wird verschlüsselt und ein SSL-Header vorrangestellt. Anschliessend wird dieser Block dem TCP-Stack übergeben. Der Empfänger entschlüsselt, verifiziert und dekomprimiert diesen Block. Anschliessend setzt er aus den anderen Teilen wieder den ursprünglichen Block zusammen und übergibt ihn dann an die Applikationsebene.

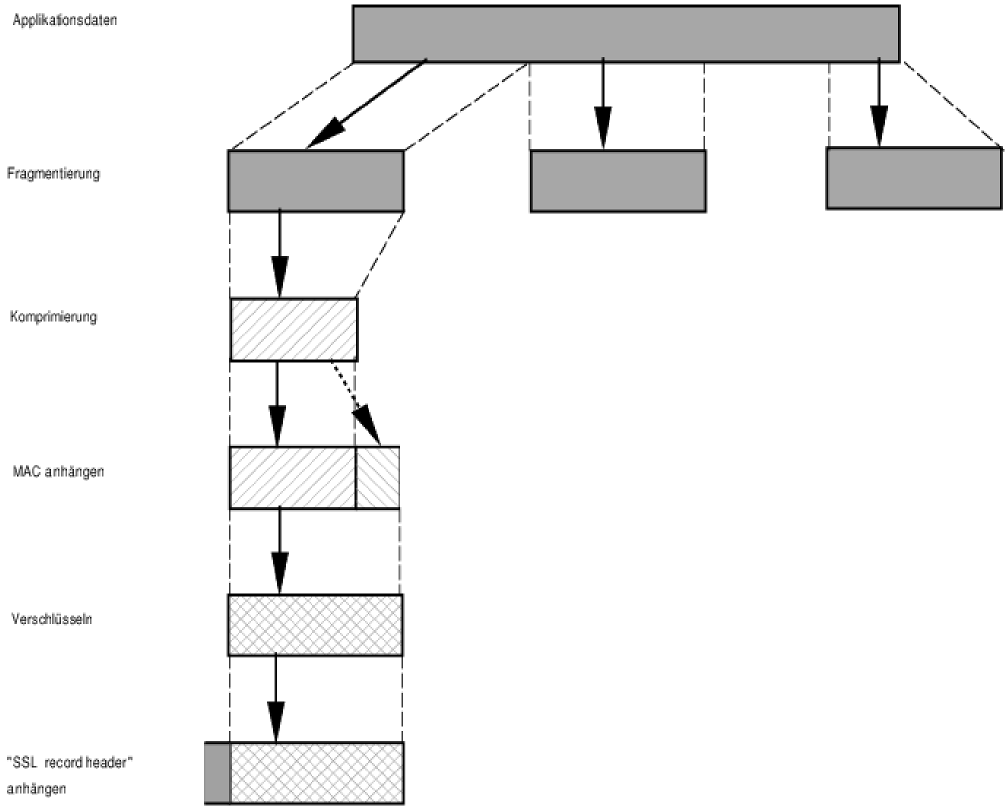

Abbildung 9: SSL-Paket erstellen

Der erste Schritt ist die *Fragmentierung*. Jedes Fragment hat eine Größe von 2^{14} Bytes oder kleiner. Anschliessend wird optional dieser Block *komprimiert*. Der verwendete Kompressionsalgorithmus muss verlustlos sein und darf die Nachricht nicht auf mehr als 1024 Bytes vergrössern[6] Wenn bei SSLv3 (als auch bei TLS) kein Kompressionsalgorithmus gewählt wurde, wird der Block nicht komprimiert. Der nächste Schritt ist das Berechnen des *message authentication code* und anschliessend das Anhängen desselben an den Datenblock. Hierzu wird ein gemeinsamer geheimer Schlüssel benötigt. Der verwendete Algorithmus ist dem des HMAC sehr ähnlich.

$hash(MAC.write.secret||pad2||$
$hash(MAC.write.secret||pad1||seq.num||SSLCompressed.type||$
$SSLCompressed.length||SSLCompressed.fragment))$

wobei
|| = Konkatenation
MAC.write.secret = gemeinsamer geheimer Schlüssel
hash = Hashalgorithms, entweder MD5 oder SHA-1
pad1 = Das Byte 0x36 48 mal wiederholt für MD5 oder 40 mal für SHA-1
pad2 = Das Byte 0x5c 48 mal wiederholt für MD5 oder 40 mal für SHA-1
seq.num = Die Sequenznummer für diese Nachricht
SSLCompressed.type = Das nächst höhere Protokoll,welches benutzt wurde, um dieses Fragment zu bearbeiten
SSLCompressed.length = Die Länge des Fragments
SSLCompressed.fragment = Das komprimierte Fragment, oder falls keine Kompression der Daten gewählt wurde, der Klartext.

Anschliessend wird die komprimierte Nachricht einschliesslich des MACs mit einem symmetrischen Schlüssel verschlüsselt. Wobei die verschlüsselte Nachricht nicht größer als 1024 Bytes werden darf. Somit wird die Gesamtlänge nicht größer als $2^{14} + 1024$ Bytes.

[6] Eigentlich sollte ein Komprimierungsalgorithmus den Block verkleinern. Bei sehr kleinen Blöcken, kann es wegen Formatierungskonventionen vorkommen, dass diese anschliessend grösser sind.

Folgende Verschlüsselungsalgorithmen dürfen hierfür verwendet werden:

	BlockChiffren		Strom Chiffren
Algorithmus	Schlüsselgrösse	Algorithmus	Schlüsselgrösse
IDEA	128	RC4-40	40
RC2-40	40	RC4-128	128
DES-40	40		
DES	56		
3DES	168		
Fortezza	80		

Tabelle 6: SSL Verschlüsselungsalgorithmen

Zum Abschluss wird im SSL Record Protokoll noch ein Header der Nachricht vorangestellt, der folgende Felder hat:

- **Content Type (8 bits)**: Das höhere Protokoll, welches für die Verarbeitung verwendet wird. Hierbei handelt es sich nicht um das Protokoll aus der Applikationsebene sondern um eines aus change_cipher_spec, alert, hand_shake und application_data. Das Anwendungsprotokoll ist transparent für SSL.
- **Major Version (8 bits)**: Kennzeichnet die Version des verwendeten SSL Protokolls. Bei SSLv3 ist der Wert 3.
- **Minor Version (8 bits)**: Kennzeichnet die Unterversion. Bei SSLv3 ist der Wert 0
- **Compressed length (16 bits)**: Die Länge der Klartextnachricht in Bytes. Bei Komprimierung, die Länge der komprimierten Klartextnachricht. Der maximale Wert darf nicht grösser als $2^{14} + 2048$ sein.

Change Cipher Spec Protocol Das *Change Cipher Spec Protocol* ist eines von drei SSL-spezifizierten Protokollen, welches das darunterliegende *SSL Record Protokoll* benutzt. Dieses Protokoll besteht nur aus einer einzigen Nachricht mit nur einem Byte. Dieses Byte hat nur den Wert 1. Der Sinn hinter diesem Protokoll ist es, einen wartenden Zustand in einen aktuellen Zustand zu kopieren. Dieses hat zur Folge, dass die Chiffre-Suite aktualisiert wird.

Content Type	Major Version	Minor Version	Compressed Length
Plaintext (optinal komprimiert) verschlüsselt			
MAC (0,16 oder 20 Bytes) verschlüsselt			

Tabelle 7: SSL Record Format

Alert Protocol Das *Alert Protocol* dient zum Übertragen von SSL Fehlermeldungen. Wie bei Applikationen, die SSL benutzen, werden diese Nachrichten komprimiert und verschlüsselt. Jede Nachricht aus diesem Protokoll besteht aus 2 Bytes. Das erste Byte kann den Wert warning(1) oder fatal(2) haben und gibt den Grad der Fehlermeldung an. Wird der Wert fatal übertragen, beendet SSL sofort die Verbindung. Andere Verbindungen aus der Session bleiben bestehen, aber es kann keine neue Verbindung erzeugt werden. Das zweite Byte beschreibt den Fehler genauer.

Fatale Alertmeldungen

- **unexpected_message**: Ungültige Nachricht erhalten.
- **bad_record_mac**: Ungültige MAC erhalten.
- **decompression_failure**: Die Kompressionsfunktion hat eine ungültige Eingabe erhalten.
- **handshake_failure**: Der Sender konnte keine Vereinbarung über die geforderten Sicherheitsparameter eingehen.
- **illegal_parameter**: Ein Feld in einer *handshake*-Nachricht ist ungültig oder inkonsistent mit anderen Feldern.

Warnende Alertmeldungen

- **close_notify**: Der Sender dieser Nachricht wird keine Nachrichten mehr schicken.
- **no_certificate**: Antwort auf ein certificate_request, falls keines vorhanden ist.
- **bad_certificate**: Ein korruptes Zertifikat erhalten.

- **unsupported_certificate**: Type des erfhaltenen Zertifikats wird nicht unterstützt.
- **certificate_revoked**: Das Zertifikat wurde widerrufen von seinem Signierer.
- **certificate_expired**: Das Zertifikat ist abgelaufen.
- **certificate_unkown**: Sonstiger Fehler ist beim validieren des Zertifikats aufgetreten.

Handshake Protokoll Das komplexeste Protokoll bei SSL ist das Handshake Protokoll. Dieses Protokoll dient zum authentisieren des Servers und des Clients, zum Aushandeln eines Verschlüsselungs- und MAC-algorithmus, sowie zum Austauschen der kryptographischen Schlüssel, welche für die Sicherung der Daten benutzt werden. Dieses Protokoll tauscht unterschiedliche Nachrichten zwischen Client und Server aus, die alle den gleichen Aufbau besitzen.

- Type: (1Byte) Zeigt eine von 10 möglichen Nachrichten an.
- Length: (3 Bytes) Enthält die Länge der Nachricht in Bytes.
- Content: ($\geq$ 1 Byte) Parameter, welche assoziert sind mit dieser Nachricht.

Auf der nächsten Seite folgt eine Tabelle mit den unterschiedlichen Nachrichtentypen.

Nachrichten-Typ	Parameter
hello_request	null
client_hello	version, random, session_id, cipher suite, compression method
server_hello	version, random, session_id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Tabelle 8: SSL Handshake Protokoll Nachrichten Typen

Das Handshake Protokoll kann in 4 Phasen aufgeteilt werden.

Phase 1: Erstellung der Sicherheitsfähigkeiten Diese Phase dient zum Initiieren einer logischen Verbindung und zum Aufbau der Sicherheitmöglichkeiten, die mit der Verbindung assoziert werden sollen. Dieser Austausch wird vom Client gestartet, indem dieser eine **client_hello** Nachricht an den Server schickt mit folgenden Parametern:

- **Version**: Die höchste Version von SSL, die der Client versteht.
- **Random**: Eine auf dem Client generierte Zufallsstruktur, bestehend aus einem 32-Bit Zeitstempel und 28 zufällig generierten Bytes. Diese Werte dienen zum Schlüsselaustausch und verhindern Replay-Attacken.
- **Session ID**: Ein Sitzungsbezeichner von variabler Länge. Wenn dieser Wert nicht 0 ist, dann kennzeichnet der Client hiermit, dass er die Parameter einer bestehende Session verändern möchte, oder eine neue Verbindung in der Session starten will. Ist der Wert 0, so wird eine neue Session gewünscht.
- **CipherSuite**: Eine Liste von vorhandenen kryptographischen Algorithmen, in absteigender favorisierter Ordnung. Jeder Eintrag definiert sowohl ein Chiffre, wie auch ein Schlüsseltauschalgorithmus. Diese werden später genauer erläutert.

- **Compression Method**: Eine Liste von Kompressionsalgorithmen, die der Client unterstützt.

Nachdem der Client ein *client_hello* verschickt hat, wartet er auf die *server_hello* Nachricht. Diese vom Server verschickte Nachricht hat dieselben Felder wie die *client_hello* Nachricht. Das Version Feld enthält die höchste SSL Version, die der Server unterstützt und die auch vom Client verarbeitet werden kann. Der Inhalt des Random Feldes wird vom Server generiert und ist unabhängig vom Inhalt des Random Feldes aus der Client Nachricht. Wenn das Session ID Feld vom Client nicht 0 war, wird vom Server derselbe Wert benutzt. Ansonsten enthält dieses Feld den Wert der neuen Session. Das CipherSuite Feld enthält in der Serverantwort nur einen einzigen Algorithmus mit zugehörigem Schlüsselaustauschalgorithmus. Der Server hat diese aus den angebotenen Möglichkeiten vom Client ausgewählt. Das Compression Method Feld enthält wiederrum nur einen Kompressionalgorithmus aus den Vorschlägen des Clients.
Das erste Element im CipherSuite Feld ist der Schlüsselaustauschalgorithmus. Folgende Schlüsselaustauschalgorithmen werden unterstützt:

- **RSA**: Der geheime symmetrische Schlüssel wird mit dem öffentlichen RSA Schlüssel des Empfängers verschlüsselt. Zu diesem Zweck muss ein Public-Key-zertifikat verfügbar sein.

- **Fixed Diffie-Hellman**: Hier wird das Diffie-Hellman Schlüsseltausch-Protokoll benutzt. Die Diffie-Hellman öffentlichen Parameter müssen dazu im Server Zertifikat stehen und dieses von einer „certificate authority“ (CA) signiert sein. Der Client stellt seine Diffie-Hellman öffentlichen Schlüssel Parameter entweder in einem Zertifikat zur Verfügung, falls Client-Authentisierung erfoderlich ist, oder in einer Schlüsselaustausch-Nachricht .

- **Ephemeral Diffie-Hellman**: Hier werden temporäre Einmalschlüssel erzeugt. Die Diffie-Hellman öffentlichen Schlüssel werden mit dem privaten RSA oder DSS Schlüssel des Senders signiert. Zertifikate werden eingesetzt um die öffentlichen Schlüssel zu verifizieren.

- **Anonymous Diffie-Hellman**: Der Diffie-Hellman Algorithmus wird ohne Authentikation benutzt. Dazu schickt jede Seite ihre Diffie-Hellman Parameter an die andere Seite, ohne eine Authentikation derselben. Hierbei besteht deswegen die Gefahr für ein Man-in-Middle Attacke.

- **Fortezza**: Hier wird das Fortezza Verfahren benutzt.

Nach dem Schlüsselaustauschalgorithmus folgen die Chiffre-Parameter, welche aus folgenden Feldern besteht:

- **Cipher Algorithm**: Ein Algorithmus aus RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza.
- **MAC-Algorithm**: MD5 oder SHA-1.
- **Chipher Type**: Stream oder Block Chiffre.
- **Is Exportable**: true oder false.
- **Hash Size**: 0, 16 (MD5) oder 20 (SHA-1).
- **Key Material**: Eine Sequenz von Bytes, welche für die Generierung von Schreibkeys benutzt wurde.
- **IV Size**: Die Größe des Initialisierungswertes für die „Cipher Block Chaining“ Verschlüsselung.

Phase 2: Serverauthentisierung und Schlüsseltausch Der Server beginnt die Phase mit dem Senden seines Zertifikates in einer **certificate message** , falls er authentisiert werden soll. Dies ist nicht notwendig, falls *Anonymous Diffie-Hellman* benutzt wird. Diese Nachricht enthält ein X.509 Zertifikat oder eine Kette von X.509 Zertifikaten. Falls *Fixed Diffie-Hellman* benutzt wird, steht in diesem Zertifikat die öffentlichen Diffie-Hellman Parameter. Anschließend kann, falls notwendig, eine **server_key_exchange_message** vom Server gesendet werden. Es wird nicht verlangt, falls der Server *Fixed Diffie-Hellman* benutzt oder RSA zum Schlüsseltausch angewendet wird. Nun kann der Server ein Client Zertifikat anfordern, dazu schickt er eine **certificate_request_message** Nachricht. Diese enthält zwei Parameter: certificate_type und certificate_authorities. Der certificate_type bezeichnet den Publickey-Algorithmus des Zertifikates und certificate_authorities enthält eine Liste von ausgezeichneten Namen und akzeptierten CAs. Abschliessend für die 2. Phase wird die **server_done_message** Nachricht geschickt. Diese ist immer erforderlich. Nach dem Senden dieser Nachricht wartet der Server auf die Antwort vom Client. Diese Nachricht besitzt keine Parameter.

Phase 3: Clientauthentisierung und Schlüsseltausch Nachdem der Client die *server_done_message* empfangen hat, prüft dieser , falls notwendig, ob er ein validiertes Zertifikat erhalten hat und checkt ob die *server_hello* Parameter akzeptabel sind. Wenn alles zufriedenstellend ist, schickt der Client folgende Nachrichten an den Server. Falls der Server ein Client- Zertifikat erfordert, so schickt der Client zunächst eine **certificate_message** zum Server. Anschließend muss ein **client_key_exchange_message** zum Server geschickt werden. Der Inhalt dieser Nachricht hängt von dem verwendeten Schlüsseltausch-Protokoll ab.

- **RSA**: Der Client generiert ein 48-Byte *pre-master secret* und verschlüsselt dieses mit dem öffentlichen Schlüssel des Servers. Entweder aus dem Zertifikat oder aus dem temporären RSA-Schlüssel aus der *server_key_message*. Dieses wird später benutzt, um einen *master secret* zu berechnen.

- **Ephemeral oder Anonymous Diffie-Hellman**: Die Client öffentlichen Diffie-Hellman Parameter werden verschickt.

- **Fixed Diffie-Hellman**: Die öffentlichen Diffie-Hellman Parameter wurden schon im *certificate_message* verschickt, daher ist der Inhalt dieses Paketes null.

- **Fortezza**: Die Fortezza Parameter werden verschickt.

Zuletzt in dieser Phase kann der Client eine **certificate_verify_message** verschicken. Diese dient zum expliziten verifizieren des Clients Zertifikates. Diese Nachricht wird nur verschickt, wenn das Client Zertifikat Signierfunktionalität besitzt (alle ausser *Fixed Diffie-Hellman*). Diese Nachricht enthält den signierten

Hashcode aller vorherigen Nachrichten. Wenn jemand das Client-Zertifikat entwendet hat, aber nicht den privaten Schlüssel für dieses Zertifikat besitzt, kann er diese Nachricht nicht erstellen.

Phase 4: Finish Diese Phase beendet den Aufbau einer gesicherten Verbindung. Der Client sendet eine **change_cipher_spec_message** Nachricht und kopiert die ausgehandelten ChipherSpec in die aktuellen CipherSpec. Diese Nachricht ist nicht mehr Teil des *Handshake Protokolls*, sondern nutzt das *Change Cipher Spec Protokoll*. Anschliessend schickt der Client eine **finished_message** Nachricht. Diese Nachricht wird mit dem neuen Algorithmus, Schlüssel und Geheimnis erstellt. Diese Nachricht verifiziert, dass der Schlüsseltausch und Authentisierungsprozess erfolgreich waren. Der Inhalt dieses Paketes besteht aus zwei Hashwerten:
$MD5(master\_secret||pad\_2||$
$MD5(handshake\_messages||Sender||master\_secret||pad\_1))$
$SHA(master\_secret||pad\_2||$
$SHA(handshake\_messages||Sender||master\_secret||pad\_1))$
Wobei $Sender$ ein Code ist, der den Client identifiziert und $handshake\_messages$ alle Nachrichten bis zu dieser enthält.
Der Server antwortet auf diese empfangene Nachricht mit seiner **change_Cipher_Spec_message** und transferiert die ausgehandelte CipherSpec in seinen aktuellen CipherSpec. Anschliessend schickt auch der Server eine **finished_message**.
Nun ist der Verbindungsaufbau vollständig und der Client und Server können die Daten aus dem Applikations-Layer austauschen.

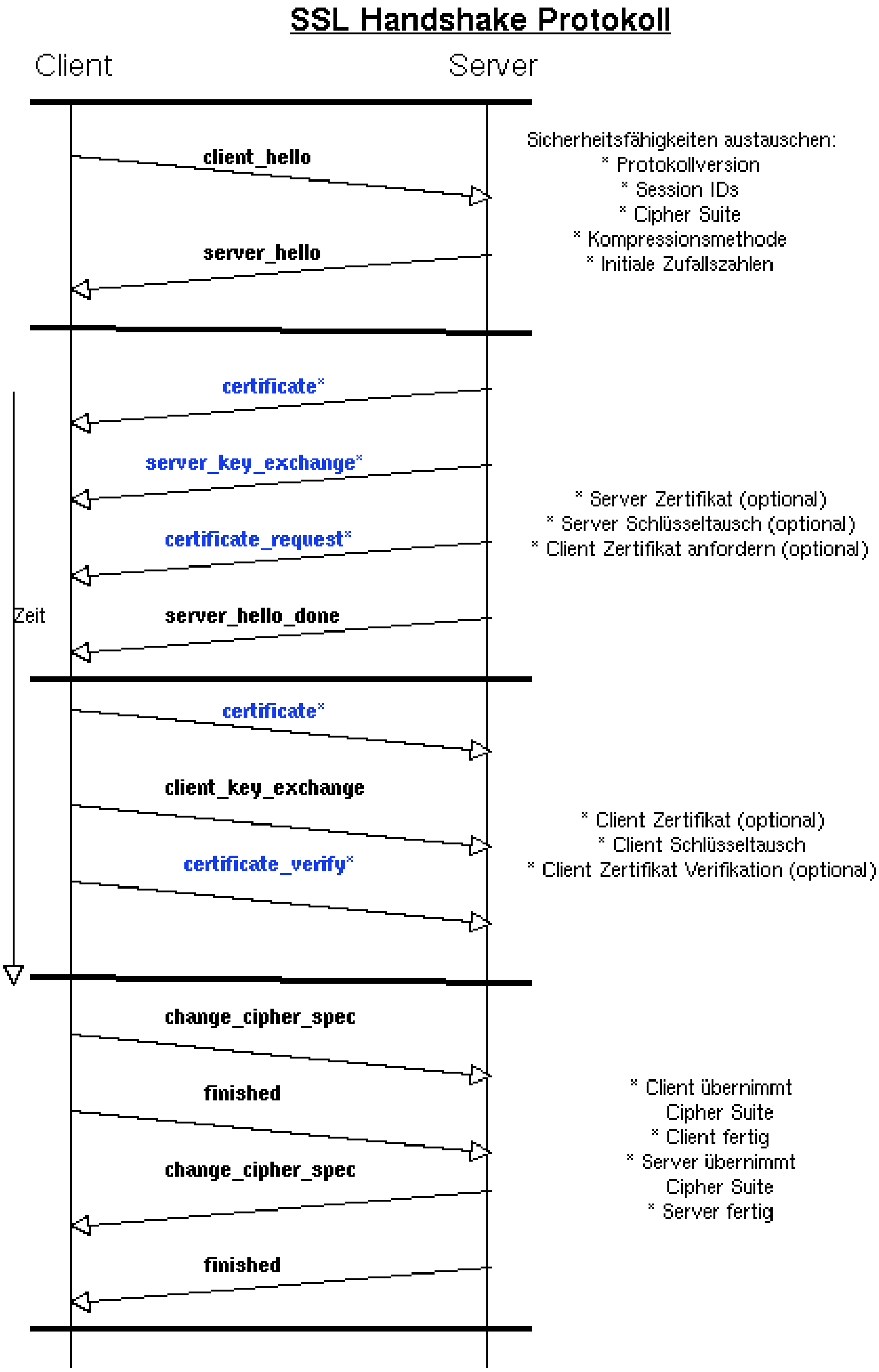

Abbildung 10: SSL Handshake Protokoll

Generierung des Master-Keys Der gemeinsam benutzte *Master-Key* besteht aus 48 Byte und und gilt nur für eine Session. Der Schlüssel wird in 2 Stufen generiert. Zuerst wird im Handshake-Protokoll ein *pre_master_secret* ausgetauscht. Dabei exisitieren zwei Fälle für den Austausch des *pre_master_secret*:

- **RSA**: Der 48 Byte große *pre_master_secret* wird vom Client erzeugt, verschlüsselt mit dem öffentlichen Schlüssel des Servers und anschließend an diesen versandt. Der Server entschlüsselt die erhaltene Nachricht mit seinem privaten Schlüssel um den *pre_master_secret* zu erhalten.
- **Diffie-Hellman**: Sowohl Client und Server generieren einen Diffie-Hellman öffentlichen Schlüssel. Nachdem diese Schlüssel ausgetauscht worden sind, führt jede Seite eine Diffie-Hellman Calculation durch, um den gemeinsamen *pre_master_secret* zu generieren.

In der zweiten Stufe berechnen dann beide Parteien aus diesem *pre_master_secret* den *Master-Key*:

```
master_secret := MD5(pre_master_secret || SHA('A' || pre_master_secret ||
                        ClientHello.random || ServerHello.random)) ||
                 MD5(pre_master_secret || SHA('BB' || pre_master_secret ||
                        ClientHello.random || ServerHello.random)) ||
                 MD5(pre_master_secret || SHA('CCC' || pre_master_secret ||
                        ClientHello.random || ServerHello.random)) ||
```

wobei ClientHello.random und ServerHello.random die beiden Werte aus der ausgetauschten anfänglichen *hello_message* sind.

Erzeugung der CipherSpecs Die benutzten CipherSpecs benötigen ein Client MAC-Geheimnis, einen Server MAC-Geheimnis, einen Client Schlüssel, ein Server Schlüssel, ein Client Initialisierungsvektor und einen Server Initialisierungsvektor. Diese werden aus dem *Master-Key* erzeugt. Dazu werden mehrere Hashes des *Master-Key* für die notwendige Anzahl von Bytes für die einzelnen Paramter errechnet. Man kann somit den *Master-Key* als Seed-Wert für eine pseudozufällige Berechnung ansehen und die ClientHello.random und ServerHello.random als zusätzliche Verschleierung für eine Kryptoanalyse.

Unterschiede zwischen SSLv3 und TLS TLS, welches aufbauend von SSLv3 bei der IETF entstanden ist, liegt zur Zeit als Draft Version vor. Hier werden kurz die Unterschiede zu SSLv3 vorgestellt:

- **Versionsnummer**: Die Versionsnummer im *Recordformat* unterscheidet sich zu SSLv3. Aktuell ist für TLS die Major Version 3 und die Minor Version 1.
- **Message Authentication Code**: Während beim MAC-Algorithmus in SSLv3 die Padding-Daten mit dem geheimen Schlüssel verkettet werden , wird bei TLS die Padding-Daten XORed. Ausserdem wird bei der MAC-Berechnung bei TLS noch ein zusätzlicher Parameter als bei SSLv3 benutzt, der Wert TLSCompression.version.
- **Pseudozufallsfunktion**: TLS benutzt zusätzlich noch eine Pseudozufallsfunktion in der Generierung und Verifikation der Schlüssel. Der Sinn dabei ist, aus relativ kleinen Geheimnissen größere Byteblöcke zu generieren, welche resistenter sind bei Attacken gegen die Hashfunktionen.
- **Alert Codes**: TLS implementiert alle Alert Codes aus SSLv3 und fügt noch zusätzliche hinzu.
- **Cipher Suite**: TLS unterstützt alle Schlüsselaustausch-Protokolle ausser Fortezza. TLS unterstützt alle Symmetrischen Verschlüsselungsalgorithmen von SSLv3 ausser Fortezza.
- **Client Zertifikat Type**: SSLv3 besitzt zusätzlich zu TLS rsa_ephemeral_dh, dss_ephemeral_dh und fortezza_kea als Zertifikat Typen. Ephemeral Diffie-Hellman benutzt entweder RSA oder DSS um die Diffie-Hellman Parameter zu signieren. Bei TLS werden hierfür die vorhanden rsa_sign oder dss_sign benutzt. TLS beinhaltet nicht das Fortezza-Schema.

- **Certificate_Verify and Finished Messages**: Bei TLS wird in der *certificate_verify_message* der Hash nur über die *handshake_messages* erzeugt. Die extra Parameter aus SSLv3 *master_secret* und *pads* würden die Sicherheit nicht erhöhen. Der Hash in der *finished_message* wird bei TLS anders berechnet.

- **Master Secret**: Der *pre_master_secret* wird wie bei SSLv3 erzeugt. Der daraus berechnete *master_secret* wird aber anders erzeugt. Auch die aus dem *master_secret* erzeugten MAC Geheimnisse, Session Schlüssel und Initialisierungs Vektoren werden anders berechnet.

- **Padding**: Bei SSLv3 werden die Applikationsdaten auf das Minimum der benötigten Chiffreblockgröße aufgefüllt. TLS erlaubt auch das Auffüllen in ein vielfaches der benötigten Chiffreblockgröße. Dies soll die Angriffe auf die Länge der übermittelten Nachrichten erschweren.

3.2.2 Das Biometrische Protokoll

Hier wird das Konzept des biometrischen Protokolls erläutert. Es wird versucht ein recht allgemeines Protokoll für die Übertragung von biometrischen Daten zu erstellen. Natürlich unterscheiden sich die Protokollschritte in Abhängigkeit von der verwendeten Erfassungseinheit. Bei einer *Voll autonomen Erfassungseinheit* fallen andere Protokollschritte an, als bei einer *Nicht autonomen Erfassungseinheit*.
Das Protokoll soll aber so erstellt werden, dass diese Fälle mit nur wenigen unterschiedlichen Protokollschritten abgebildet werden können. Hauptsächlich liegt der Unterschied bei der Übertragung der Daten und den eventuell notwendigen Kalibrierungsschritten. Kann die Erfassungseinheit dieses nicht selbständig leisten, so müssen die notwendigen Schritte vom Server aus getriggert werden und fliessen somit in das Protokoll. Auch der Abschluss der zu übertragenden biometrischen Daten kann entweder bei der Erfassungseinheit liegen oder beim Server.
Zunächst wird das Protokoll für eine *Voll autonome Erfassungseinheit* aufgezeigt. Anschliessend werden dann die unterschiedlichen Schritte bei einer *Nicht autonomen Erfassungseinheit* erklärt.

Biometrisches Protokoll für eine „Voll autonome Erfassungseinheit“
Die nachfolgende Abbildung illustriert einen erfolgreichen Authentikationsprozess. Es zeigt also den idealen Ablauf einer Kommunikation zwischen den beiden Teilnehmern. Es erscheinen keine Fehlerereignisse oder eventuelle notwendige Rescans in diesem Ablauf. Das zu erstellende Protokoll hingegen muss solche Situationen natürlich behandeln. Anhand dieser Abbildung werden anschliessend die einzelnen Schritte genauer erläutert.

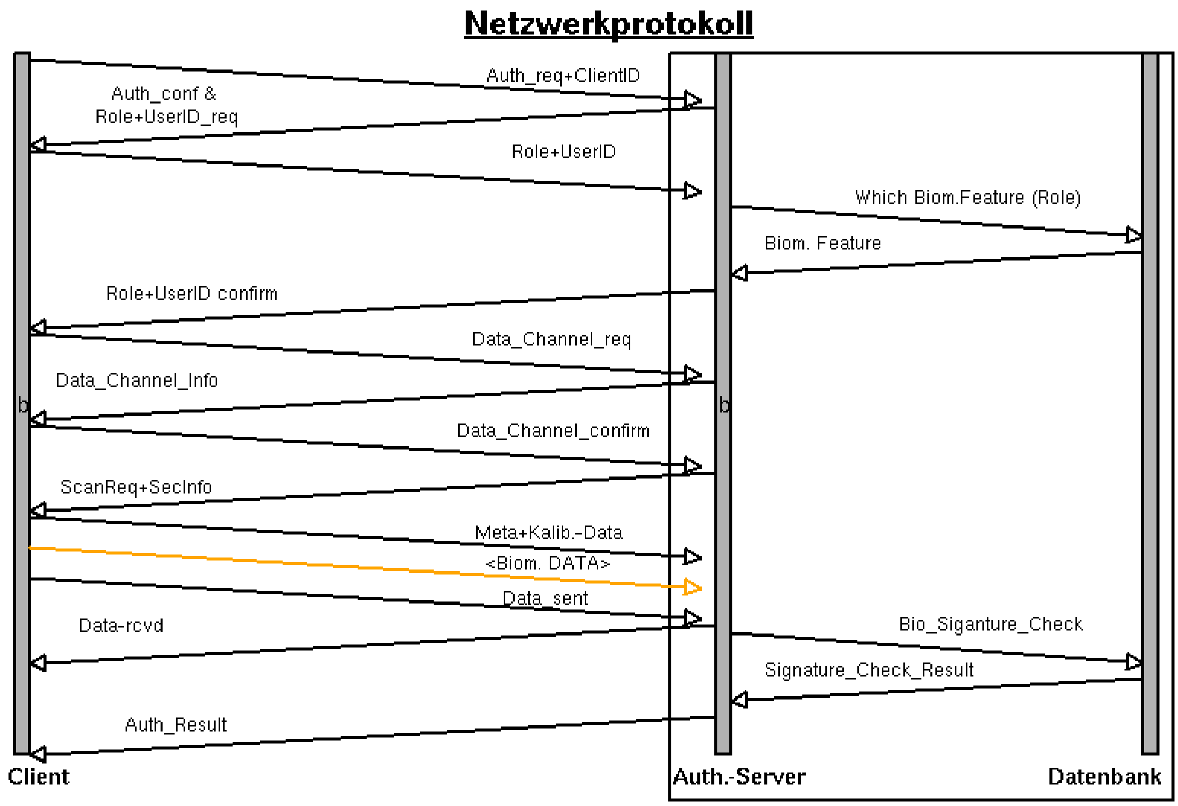

Abbildung 11: Biometrisches Protokoll für VAE

Auth_req+ClientID : Der Client initiiert eine Verbindung mit dem Server zum Zweck einer „Biometrischen Authentikation“. Dazu schickt er ein Paket mit der Aufforderung eines neuen Authentikationsprozesses zum Server. In diesem Paket überträgt der Client auch seine ClientID. Diese ClientID dient dem Server, die unterschiedlichen Clients zu unterscheiden und kann für bestimmte Auswahlprozesse auf dem Server genutzt werden, z.B kann der Server eine Liste mit zeitlich zulässigen Clients führen oder auch anhand der ClientID herausfinden, welche Funktionalität diese Erfassungseinheit bereit hält.

Auth_conf&Role+UserID_req : Der Server antwortet auf ein *Auth_req+ClientID* mit entweder einer Bestätigung *Auth_conf&Role+UserID_req*, so dass der Prozess fortgesetzt werden kann oder lehnt die Verbindung mit *Auth_conf[DENY]* ab. Soll der Prozess fortgeführt werden, fordert der Server den Client auf, ihm den Authentisierungskontext zu nennen. Zu diesem Zweck benötigt der Server sowohl eine Benutzerkennung, als auch die Role für den Nutzer, der sich authentisieren möchte.

Role+UserID : Die Erfassungseinheit muss zunächst die geforderten Daten für die Benutzerkennung und die zugehörige Role beschaffen. Dazu könnte eine Interaktion mit dem Benutzer stattfinden, bei der er diese Daten der Erfassungseinheit mitteilt (Tastatureingabe/SmartCard). Es könnte aber auch sein, dass die Role für diese Erfassungseinheit fest vorgegeben ist. In diesem Fall wird die Role von der Erfassungseinheit erstellt und der Nutzer muss lediglich seine Benutzerkennung mitteilen. Es sind auch Erfassungseinheiten vorstellbar, die überhaupt keine Möglichkeit vorsehen, eine Benutzerkennung und eine Role von dem sich Authentisierenden zu erhalten. In diesem Fall sollte ein zuvor definierter Benutzername mit zugehöriger Role von der Erfassungseinheit übertragen werden, damit der Server weiß, dass er den Nutzer nur anhand seiner biometrischen Daten ermitteln soll. Egal welche Funktionalität die Erfassungseinheit besitzt, lassen sich alle diese Fälle aber mit diesem Protokollschritt abbilden.

Role+UserID_conf : Bei dieser Nachricht bestätigt der Server den Authentisierungsprozess, falls der Benutzer mit der angegebenen Role bekannt ist.

Sollte der gewählte Benutzer oder seine Role nicht mit den Serverdaten übereinstimmen, besteht die Möglichkeit, erneut ein *Role+UserID_req* zu senden oder aber den gesamten Authentisierungsprozess mit einem *Role+UserID_deny* abzubrechen. Das erneute Versenden eines *Role+UserID_req* bietet die Möglichkeit, eine bestimmte Anzahl von Wiederholungen bei der Auswahl der Role und der UserID zu erlauben.

Data_Channel_req : Hat der Client eine positive Rückantwort vom Server auf die UserID und die Role erhalten, so schickt er diese Nachricht an den Server mit der Bitte, einen Datenkanal zu eröffnen. Es besteht auch die Möglichkeit, dass in dieser Nachricht, um mehrere Datenkanäle gebeten wird. Dieses wäre dann der Fall, wenn die Erfassungseinheit ein *Multimodales System* ist und somit mehrere biometrische Charakteristika aufzeichnen kann. Gerade wenn es sich um eine *nicht autonome Erfassungseinheit* handelt, würden hier für die verschiedenen biometrischen Datenströme unterschiedliche Datenkanäle erbeten.

Data_Channel_Info : Der Server bereitet alles für den Empfang der biometrischen Daten vor und schickt anschliessend dem Client die notwendigen Informationen für die Datenkanäle zurück. Er teilt ihm also die IP-Adresse und die Port-Nummer mit. Bei mehreren Datenkanälen efolgt in dieser Nachricht zudem eine Zuordnung der IPs und Ports zu den zu übersendenden biometrischen Daten. Es wäre auch möglich, den Empfang der biometrischen Daten nicht auf dem Authentisierungsserver zu realisieren, sondern explizit ausgewählte Rechner für diese Aufgabe vorzuhalten. Aus diesem Grund wird die IP-Adresse mit übermittelt.

Data_Channel_conf : Der Client baut mit der vom Server erhaltenen Information eine oder mehrere Verbindungen zu den Datenkanälen-Gegenstellen auf und schickt anschliessend die Bestätigung *Data_Channel_conf*. Somit weiß der Server, dass der Aufbau der benötigten Datenkanäle abgeschlossen ist. Sollte es dem Client nicht möglich sein, die Datenkänale aufzubauen, so teilt der Client dem Server dieses mit, indem er erneut ein *Data_Channel_req* an den Server sendet. In der erneut versendeten *Data_Channel_req* brauchen nur die notwendigen Verbindungsdaten für die Datenkanäle zu stehen, die nicht erfolgreich aufgebaut werden konnten. Schon aufgebaute Kanäle bleiben bestehen.

Scan_req+Sec_Info : Der Server löst jetzt einen Scan auf der Erfassungseinheit aus. Zusätzlich besteht die Möglichkeit, bestimmte sicherheitserhöhende Parameter für den Scan der Erfassungseinheit mitzuteilen. Es handelt sich dabei um Parameter, die das Umfeld des Scan beeinflussen. Bei einem Irisscan könnte dies die Auswahl von Lampen an der Erfassungseinheit sein, die sich als Reflektionen auf der Aufzeichnung widerspiegeln. Bei Spracherkennung wäre auch die Vorgabe eines zu sprechenden Satzes möglich. Je nach verwendetem biometrischen Verfahren lassen sich hier also Parameter übertragen, die der Server vorgibt. So wird es möglich, dass der Server eine Lebenderkennung durchführen kann.

Meta+Kalib_Data : Vor der Übertragung der eigentlichen biometrischen Daten über den oder die Datenkanäle teilt der Client dem Server Metadaten bezüglich der Aufzeichnung mit. Der Inhalt dieser Metadaten ist nicht weiter spezifiziert und könnte Zeit, Ort oder auch Umwelteinflüsse bei der Aufzeichnung beinhalten. Dies könnten Temperatur oder Wetterdaten sein oder bei akustischen Aufzeichnungen der Umgebungsgeräuschpegel. Zusätzlich werden Kalibrierungsdaten von der Aufzeichnung dem Server übersandt. Dies dient zum einen für statistische Auswertungen oder ist bei der *nicht autonomen Erfassungseinheit* notwendig, damit der Server die Parameter der Aufzeichnung ändern kann. Es versteht sich von selbst, dass bei der *nicht autonomen Erfassungseinheit* während der Übertragung der biometrischen Daten, diese Kalibrierungsdaten wiederholt an den Server gesendet werden müssen.

<Biometrik_Data> : Jetzt werden die biometrischen Daten über einen oder mehrere Datenkanäle übertragen. Es kann sich um abgeschlossene oder Streamingdaten handeln. Da diese Daten nicht über den Kontrollkanal laufen, bleibt dieser frei für die Kalibrierungskommunikation.

Data_Sent : Dieser Schritt ist eventuell optional und hängt von der Erfassungseinheit ab. Er kommt nur dann nicht zum Einsatz, wenn es sich um eine *nicht autonome Erfassungseinheit* handelt. Die Erfassungseinheit teilt in diesem Protokollschritt dem Server mit, dass sie die biometrischen Daten übermittelt hat. Optional kann in diesem Protokollschritt noch die Anzahl der Bytes mit angegeben werden, welche über den Datenkanal übertragen wurden.

Data_rcvd : Der Server quittiert die empfangenen biometrischen Daten mit dieser Meldung dem Client. Bei einer *nicht autonomen Erfassungseinheit* wird diese Nachricht vom Server zum Client übermittelt, wenn der Server ausreichend Daten empfangen hat. Mit diesem Schritt wird die Übertragung der biometrischen Daten abgeschlossen. Sollte der Server einen erneuten Scan benötigen, so braucht er anschliessend nur erneut eine **Scan_req+Sec_Info** Nachricht an die Erfassungseinheit schicken und der Scanvorgang würde wiederholt.

Auth_Result : Zum Schluss wird jetzt das Ergebnis der Authentisierung übermittelt, nachdem die empfangenen Daten dem biometrischen Authentisierungsprozess übermittelt worden sind, und das Ergebnis bekannt ist. Bei einer erfolgreichen Authentisierung schickt der Server *Auth_allow* an den Client. Bei Misserfolg wird ein *Auth_deny* verschickt. Hiermit endet die Kommunikation der beiden Stationen.

DFA Zustandsdiagramm des Kontrollprotokolls für die Erfassungseinheit und den Server:
Bei den nachfolgenden Diagrammen, handelt es sich um die Darstellung des Kontrollprotokolls für eine Erfassungseinheit und eines für den Server. Diese Erfassungseinheit kann eine *voll autonome* oder auch eine *halb autonome* Erfassungseinheit sein. Das Protokoll wird als endlicher Automat (DFA) dargestellt. Die Kreise zeichnen Zustände aus, in denen sich die Erfassungseinheit befinden kann. Die Kanten werden durch empfangene Nachricht (recv) und zu sendende Nachricht (send) beschriftet. Der Wechsel in einen anderen Zustand ist nur dann möglich, wenn die Kantenbeschriftung erfüllt wird. Der Client muss die Nachricht unter **recv** empfangen und darauf die Nachricht unter **send** verschicken. Ansch-

liessend gelangt der Client in den neuen Zustand, zu dem die Kante zeigt. Die roten Kanten führen zu einem misslungenen Authentikationsprozess und beenden die Kommunikation mit dem Server. Die grünen Kanten weisen auf Wiederholungen im Ablauf hin. Außerdem sind noch an jedem Zustand eine Schlinge[7] und eine Kante, welche nicht mit eingezeichnet wurden. Die Schlinge gilt, wenn der Client in diesem Zustand eine Nachricht erhält, die an keiner seiner anderen abgehenden Kanten bei **recv** auftaucht. Die Kante führt wieder zum Startzustand. Diese Kante tritt ein wenn ein Timeout erfolgt. Ein Timeout findet dann statt, wenn nach einer vorzugebenden Zeit der Zustand nicht gewechselt wurde.

[7]Schlingen sind Kanten, welche wieder auf den selben Zustand zeigen.

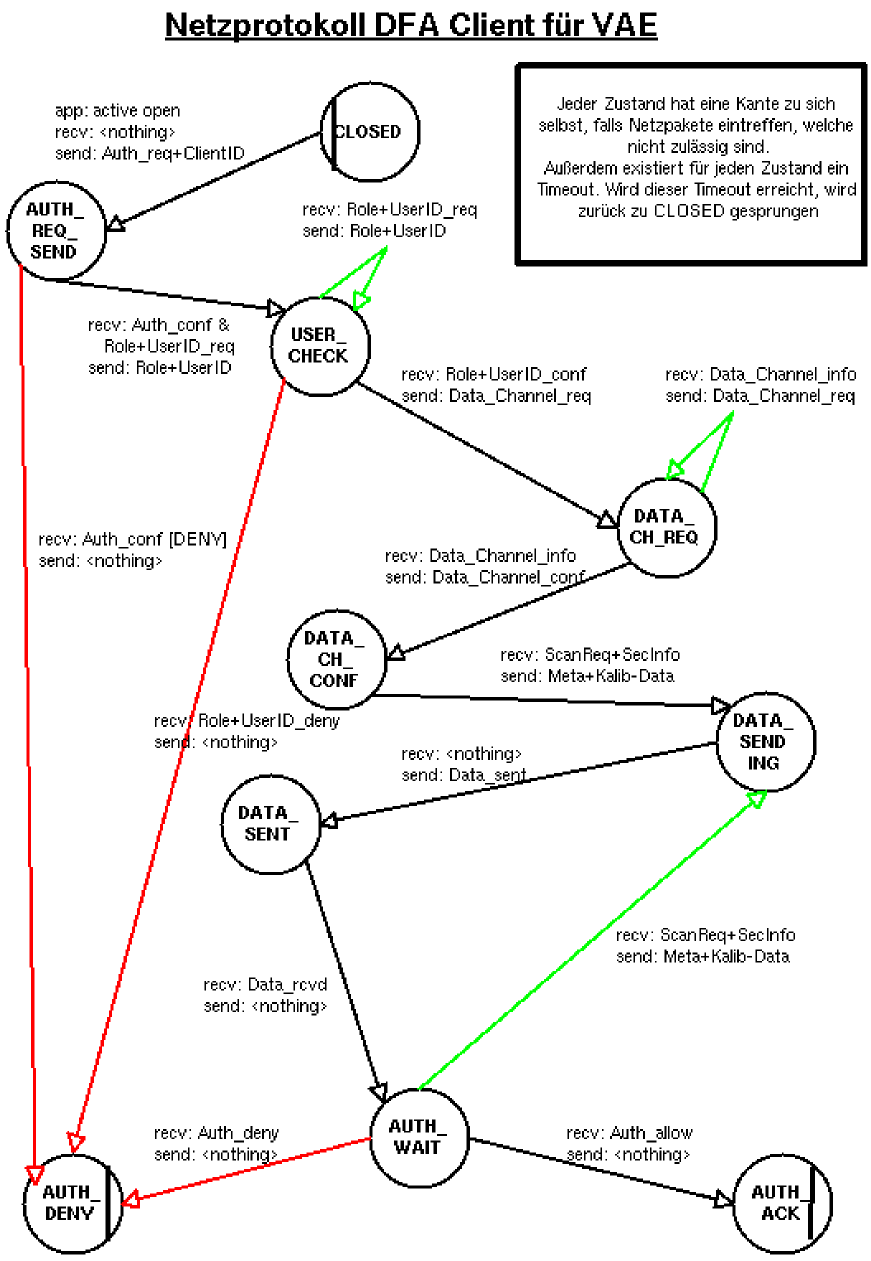

Abbildung 12: Kontrollprotokoll DFA Client

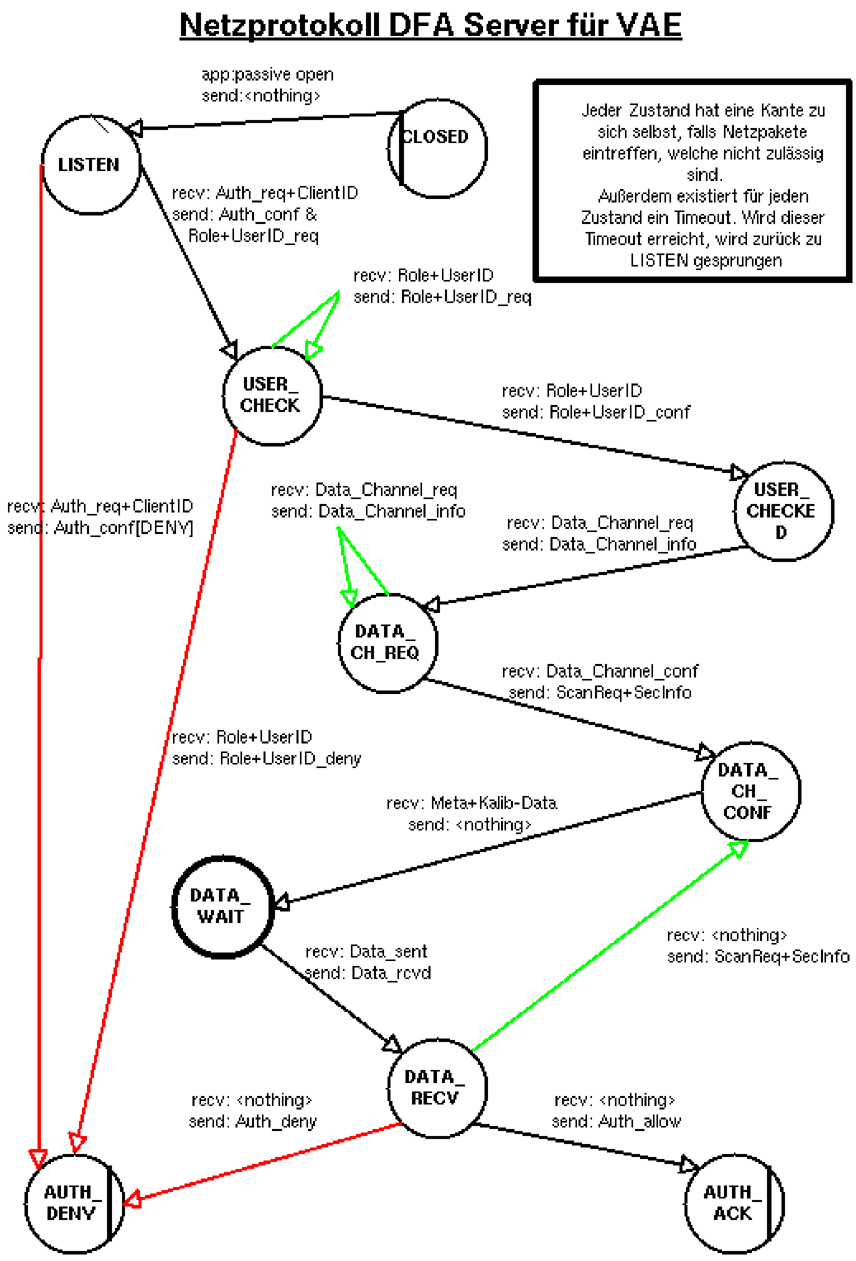

Abbildung 13: Protokoll DFA Server

3.3 Nachweis der Implementation

Um das entworfene Protokoll zu testen, wurde ein kleines Programm geschrieben. Dieses Programm besteht aus einer Serverapplikation und einer Clientapplikation. Es wird mit diesen beiden Programmen ein biometrischer Authentikationsprozess simuliert. Der Server stellt den Authentikationsserver dar, bei denen sich eine Erfassungseinheit anmeldet. Der Client simuliert die Erfassungseinheit. Da der Client keine Aufzeichnungsfunktionalität für biologische Merkmale besitzt, wird dieses durch unterschiedliche zuvor erzeugte Irisbilder bewerkstelligt. Es wird davon ausgegangen, dass die zu simulierende Erfassungseinheit vier unterschiedliche Lampen besitzt, die bei der Aufnahme aktiviert sein können. Welche der Lampen aktiviert sein sollen, gibt der Server dem Client vor. Dies erhöht die Sicherheit gegen Replay-Attacken. Anhand der Reflektionen auf dem emfangenen Irisbild, kann überprüft werden, ob die Lampen, wie gewünscht, angeschaltet waren.

Bei einem neuen Authentisierungsprozess, versucht der Client zunächst eine SSLv3 Verbindung mit dem Server aufzubauen. Dabei durchlaufen Server und Client die unterschiedlichen Kommunikationsschritte, die zuvor im SSL-Kapitel beschrieben wurden. Haben die beiden Parteien einen sicheren Kanal für die restliche Kommunikation aufgebaut, ist davon auszugehen, dass eine Authentisierung der Kommunikationspartner erfolgt ist und dass jeglicher Datenverkehr verschlüsselt und geschützt ist. Erst jetzt kommt das neu erstellte Protokoll zum Zuge. Es werden alle beschriebenen Protokollschritte von den Kommunikationspartnern durchgeführt, die für eine *voll autonome Erfassungseinheit* **(VAE)** beschrieben wurden. Da der Serverprozess keinen biometrischen Algorithmus ausführt oder auch keine Verbindung zu einer Benutzerdatenbank besitzt, können diese Ergebnisse des Authentikationsprozesses in der Server-GUI vorgegeben werden.

3.3.1 Aufbau der Clientapplikation

bioclient.cpp ist für die GUI-Komponenten zuständig. Es kümmert sich somit um die graphische Ausgabe. Wird eine Verbindung zu dem Authentikationsserver gewünscht, instanziert sie die Klasse **protocolmanager.cpp**. Die Klasse **protocolmanager.cpp** baut nun mit der Klasse **controlprotocol.cpp** einen gesicherten Kanal zu dem Server auf. Wird während des Authentisierungsprozesses der Datenkanal benötigt, instanziert die Klasse **protocolmanager.cpp** dynamisch die Klasse **dataprotocol.cpp**. Somit dient **protocolmanager.cpp** als Mittler zwischen den beiden gesicherten Kanälen und sorgt somit für die notwendige Synchronisation des Protokolls. Für die Übertragung des Irisbildes benutzt **dataprotokoll.cpp** die Klasse **fileloader.cpp**, die zuständig ist für die Bereitstellung der notwendigen Daten.

3.3.2 Aufbau der Serverapplikation

Analog zu der Clientapplikation ist die Klasse **bioserver.cpp** für die GUI-Darstellung zuständig und instanziert gleich zu Anfang die Klasse **protocolmanager.cpp**. Nun wird auf eine Clientverbindung gewartet. Der Protokollmanager ist so programmiert, dass mehrere unterschiedliche Clientapplikationen sich parallel mit dem Server verbinden können. Verbindet sich ein Client, instanziert der Protokollmanager die Klasse **controlprotocol.cpp** und übergibt ihr die Verbindung. Anschliessend wird vom Protocolmanager die Klasse dataprotocol.cpp instanziert. Diese Klasse ist für den Datenkanal zuständig. Auch diese Klasse kann mehrere Datenkanäle von einem Client parallel verwalten. Wenn der Server das Irisbild über den Datenkanal erhalten hat, sorgt die Klasse **Image.cpp** für die Darstellung des Bildes auf dem Bildschirm.

3.3.3 SSL-Trace

Um den Nachweis einer gesicherten Verbindung zu liefern, kann der Netzverkehr abgehört werden. Sehr hilfreich bei dieser Aktion ist **ssldump**[8]. Mit **ssldump** wurde der Netzwerkverkehr zwischen dem Server und einem Client beobachtet und ausgewertet. **ssldump** sorgt für die geordnete Darstellung der einzelnen beobachteten SSL-Protokollschritte.

In der Ausgabe von ssldump, kann man sehr schön erkennen, wie zunächst der Kontrollkanal mit Port 4342 und anschliessend der Datenkanal mit Port 4343 aufgebaut werden.

[8] SSL Protokoll Analyzer
http://www.rtfm.com/ssldump/

*** Log von ssldump ***

```
New TCP connection 1: localhost(1892) <-> localhost(4342)
1 1 0.0010 (0.0010) C>S Handshake
      ClientHello
      Version 3.0
1 2 0.0285 (0.0274) S>C Handshake
      ServerHello
      Version 3.0
      cipherSuite SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
1 3 0.0285 (0.0000) S>C Handshake
      Certificate
1 4 0.0285 (0.0000) S>C Handshake
      ServerKeyExchange
1 5 0.0285 (0.0000) S>C Handshake
      ServerHelloDone
1 6 0.1640 (0.1354) C>S Handshake
      ClientKeyExchange
1 7 0.1640 (0.0000) C>S ChangeCipherSpec
1 8 0.1640 (0.0000) C>S Handshake
1 9 0.2419 (0.0778) S>C ChangeCipherSpec
1 10 0.2419 (0.0000) S>C Handshake
1 11 1.6408 (1.3989) C>S application_data
1 12 1.6437 (0.0029) S>C application_data
1 13 1.6474 (0.0036) C>S application_data
1 14 1.6498 (0.0024) S>C application_data
1 15 1.6527 (0.0028) C>S application_data
1 16 1.6674 (0.0146) S>C application_data
New TCP connection 2: localhost(1895) <-> localhost(4343)
2 1 0.0002 (0.0002) C>S Handshake
      ClientHello
      Version 3.0
2 2 0.0350 (0.0348) S>C Handshake
```

ServerHello
Version 3.0
2 3 0.0350 (0.0000) S>C Handshake
Certificate
2 4 0.0350 (0.0000) S>C Handshake
ServerKeyExchange
2 5 0.0350 (0.0000) S>C Handshake
ServerHelloDone
2 6 0.1461 (0.1110) C>S Handshake
ClientKeyExchange
2 7 0.1461 (0.0000) C>S ChangeCipherSpec
2 8 0.1461 (0.0000) C>S Handshake
2 9 0.2153 (0.0692) S>C ChangeCipherSpec
2 10 0.2153 (0.0000) S>C Handshake
2 11 0.3532 (0.2305) C>S application_data
2 12 0.3587 (0.0563) S>C application_data
...
1 17 1.8979 (0.2305) C>S application_data
1 18 1.9542 (0.0563) S>C application_data
1 19 1.9710 (0.0167) C>S application_data
1 20 2.4576 (2.2209) C>S Alert
1 21 2.4577 (0.0004) S>C Alert
1 22 2.4582 (0.0001) C>S TCP FIN
1 23 5.9016 (3.4434) S>C TCP FIN

Beschreibung der Spalten: Verbindungsreferenz / Protokollschritt / Zeit seit Aufbau / Zeit / Kommunikationsrichtung / Paket-Typ

Die exemplarische Implementation erfüllt ihren Zweck. Es werden für den Kontrollkanal und die Datenkanäle gesicherte Verbindungen aufgebaut. Das implementierte Protokoll für den Austausch von Daten zum Zwecke einer biometrischen Authentikation beinhaltet alle zuvor beschriebenen Protokollschritte.

4 Zusammenfassung und Ausblick

Diese Studienarbeit legt die Entwicklung eines neuen Netz-Protokolls vor, das für einen biometrischen Authentikationsprozess genutzt werden kann. Die ersten Kapitel behandeln die dafür notwendigen Netzwerkgrundlagen und erläutern verschiedene biometrische Verfahren. Es wurde auch eine Einführung in die Kryptographie gegeben. Es folgt die Erarbeitung der erforderlichen Eigenschaften eines biometrischen Authentikationsprotokolles. Mit Hilfe der Protokollschicht SSL/TLS kann ein gesicherter Kanal zwischen den Kommunikationspartnern aufgebaut werden, was in Kapitel 3 näher beschrieben wird. Das neu entwickelte biometrische Authentikationsprotokoll wird spezifiziert und anschliessend daran folgt dessen exemplarische Implementation.

Im Rahmen dieser Studienarbeit, die zunächst die Grundlagen für einen verteilten biometrischen Authentikationsprozess erarbeitet hat, konnten weiterführende Fragen nicht behandelt werden. Über das Thema dieser Arbeit hinausgehend bieten sich jedoch Folgeprojekte an, die hier kurz angesprochen werden sollen.

Bei dem Entwurf des Protokolls wurde das Einsatzgebiet sehr allgemein gehalten, die Umsetzung bezieht sich aber auf ein bestimmtes Szenario, so dass hier nicht alle theoretischen Forderungen erfüllt werden. Auch der Aufbau der einzelnen Netzpakete für den Kontrollkanal wurde nicht genauer spezifiziert. Hier würde sich eine Beschreibung in der „eXtensible Markup Language“ XML[9] anbieten, die die einzelnen Felder in den Protokollpaketen genauer definiert. Der Datenkanal überträgt z.Z. nur die reinen Bilddaten. Zusätzliche Informationen werden nicht über den Datenkanal gesendet. Es ist also zu erarbeiten, welche weiteren Daten und Protokollschritte in das Datenprotokoll einfliessen sollten. Auch das Datenprotokoll selbst ist in seinem Paketaufbau nicht spezifiziert. Die exemplarische Implementation soll die wichtigsten Elemente nachweisen und arbeitet daher mit abgeschlossenen Datensätzen. Streamingdaten werden nicht über den Datenkanal übertragen. Auch das Szenario mit einer „Nicht Autonomen Erfassungseinheit“ ist nicht implementiert. Zudem wären wesentlich mehr Einstellungsmöglichkeiten,

[9]http://www.w3.org/XML/

welche die einzelnen Protokollschritte betreffen, in der realen Implementation wünschenswert. Dies beinhaltet Schleifen und Verzweigungen in dem Protokoll (z.B Rescan).

Ausserdem ist zu fragen, wie dieses Protokoll in einem noch verteilteren System aussehen müsste. Wenn z.B. die Verarbeitung der biometrischen Daten nicht auf dem Authentikationsserver stattfindet, sondern auf einem dezidierten Rechner, wäre die Entwicklung eines zusätzlichen Protokolls notwendig, das für die Synchronisation zwischen dem Authentisierungsrechner und dem biometrischen Datenverarbeitungsrechner verantwortlich ist. Zudem könnten für die Zertifikatsverteilung Lösungen erarbeitet werden, sowie für die Zertifikatsverifikation in einem hierarchischen System.

Wegen der Komplexität des Themas kann diese Arbeit nur einen Teil dessen behandeln. Sie sollte Grundlagen für einen verteilten biometrischen Authentikationsprozess entwickeln und schon die reduzierten Testszenarios zeigen nützliche Ergebnisse, die somit eine solide Basis für weiterführende Untersuchungen und Problemlösungen bilden.

Tabellenverzeichnis

1 ISO/OSI Schichtenmodell . . . 19
2 IP Schichtenmodell . . . 22
3 TCP/IP-ISO . . . 24
4 Biometrische Aufzeichnungsressourcen [Barg02-ACM] . . . 50
5 SSL Protokoll-Stack . . . 57
6 SSL Verschlüsselungsalgorithmen . . . 62
7 SSL Record Format . . . 63
8 SSL Handshake Protokoll Nachrichten Typen . . . 65

Abbildungsverzeichnis

1 FAR/FRR . . . 11
2 ARE . . . 12
3 Kommunikation . . . 14
4 Datenfluss . . . 15
5 IP Paketschachtelung . . . 25
6 Synchrone und selbstsynchronisierende Stromchiffren . . . 34
7 Erfassungseinheit und Server . . . 47
8 Biometrischer Authentikations Prozess (nach [Barg02-ACM]) . . . 47
9 SSL-Paket erstellen . . . 60
10 SSL Handshake Protokoll . . . 70
11 Biometrisches Protokoll für VAE . . . 74
12 Kontrollprotokoll DFA Client . . . 80
13 Protokoll DFA Server . . . 81

Literatur

[GK98] Lukas Gundermann und Marit Köhntoppaus, Juristische Aspekte biometrischer Verfahren, Oktober 1998, http://www.datenschutzzentrum.de/projekte/biometri/biometkk.htm

[LaFr] Langenscheidts Fremdwörterbuch Onlineausgabe http://www.langenscheidt.aol.de/

[Teu96] „Teubner - Taschenbuch der Mathematik ", B.G Teubner Verlaggesellschaft Leipzig 1996

[Barg02] Handout „Biometrik in der Gesellschaft ", Biometric Authentikation Research Group, Universität Hamburg, 2002

[Barg02-ACM] A. Brömme, M. Kronberg, O. Ellenbeck, O. Kasch: „A Conceptual Framework for Testing Biometric Algorithms within Operating Systems' Authentication ", ACM Symposium on Applied Computing SAC 2002, Madrid, Spain, March 10-14.

[DuInf93] Duden „Informatik" 2.Auflage 1993

[Lie00] Lienemann, Gerhard: TCP/IP-Grundlagen: Protokolle und Routing , 2.Aufl., Heise, Hannover, 2000

[MBK00] Mück, Dr. Hans-Joachim, Benecke, Carsten & Kelm, Stefan (Hrsg.):„Sicherheit im vernetzten Systemen", Bericht Nr. 224/00, Universität Hamburg, 2000

[SS00] M. Schumacher und R. Steinmetz (Hrsg.):„Sicherheit in Netzen und Medienströmen";, Informatik aktuell, Springer Verlag, 2000

[MOV96] A. Menezes, P. van Oorschot und S. Vanstone, „Handbook of Applied Cryptography", CRC Press, 1996

[Sch96] B. Schneier, „Angewandte Kryptographie", Addison-Wesley, 1996

[SSLv3 Draft] Alan O. Freier, Philip Karlton,Paul C. Kocher, „The SSL Protocol", 1996, http://wp.netscape.com/eng/ssl3/ssl-toc.html

[TLS-RFC] T. Dierks, C. Allen, „The TLS Protocol“, 1999, ftp://ftp.rfc-editor.org/in-notes/rfc2246.txt

[CSPK00] N. Courtois, N. Shamir, A. Patarin, J. und A. Klimov, „Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations“, Eurocrypt 2000

[CP02] N. Courtois, J. Pieprzyk, „Cryptanalysis of Block Ciphers with Overdefined Systems of Equations“, Asiacrypt 2002, http://eprint.iacr.org/2002/044/

[FSW01] N. Ferguson, R. Schroeppel und D. Whiting, „A simple algebraic representation of Rijndael“, 2001, http://www.macfergus.com/niels/pubs/rdalgeq.html