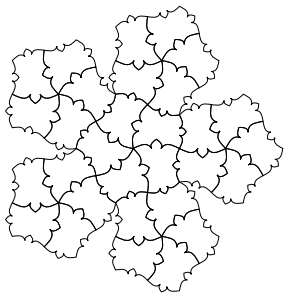


Olaf Gellert

Sicherheitsdienste im TCP/IP Protokollstapel

Diplomarbeit

Universität Hamburg
Fachbereich Informatik
Anwendungen in Geistes- und Naturwissenschaften (AGN)

Betreuer: Prof. Dr. Klaus Brunnstein
Dr. Hans-Joachim Mück

Juni 2000

Erklärung:
Ich versichere, daß ich die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt habe und keine außer den angegebenen Quellen und Hilfsmitteln benutzt habe.

Hamburg, den 29.06.2000

Danksagung:
Ich möchte mich an dieser Stelle bei meinem Erstbetreuer, Herrn Professor Dr. Klaus Brunnstein, und meinem Zweitbetreuer, Herrn Dr. Hans-Joachim Mück, für die hervorragende Betreuung bedanken. Konstruktive und motivierende Gespräche waren eine wertvolle Hilfe ebenso wie das Bereitstellen der benötigten Arbeitsumgebung und die Integration in ein produktives Arbeitsumfeld.

Besonderen Dank schulde ich Herrn Carsten Benecke, der über den gesamten Zeitraum der Erstellung dieser Arbeit stets ansprechbar war und wertvolle Hinweise und Anregungen gegeben hat.

Zusammenfassung

Mit dem Wachstum des Internets und seiner zunehmenden Kommerzialisierung nimmt auch die Anzahl der Computer und Netzwerke zu, die sensitive Daten verarbeiten, auf einen Internetzugang jedoch nicht verzichten können. Hieraus entsteht die Notwendigkeit, beim Datentransfer und Nachrichtenaustausch über Netzwerke u.a. Vertraulichkeit, Authentizität und Integrität der Daten zu gewährleisten. Die heutigen Internetprotokolle weisen jedoch gerade in dieser Hinsicht erhebliche Lücken auf, so daß mit ihnen die kommerzielle Nutzung des Internets nur bedingt möglich ist. Diese Diplomarbeit gibt einen Überblick über die bekannten Methoden, die geforderten Eigenschaften zu garantieren. Weiterhin werden die bestehenden Lösungsansätze und -möglichkeiten vorgestellt und deren Stärken und Schwächen aufgezeigt. Den Schwerpunkt der Arbeit bildet die Integration geeigneter Methoden in die TCP-Protokollschicht, um eine sichere Kommunikation zwischen Anwendungsprozessen zu ermöglichen. Ein entsprechender Prototyp wurde im Rahmen der Diplomarbeit implementiert und getestet.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Zunächst folgt eine kurze Einführung in die Problematik der Sicherheit vernetzter Systeme. Darauf aufbauend wird die Zielsetzung dieser Arbeit erläutert und ein Überblick über die folgenden Kapitel gegeben.

1.1 Sicherheit vernetzter Systeme

Das exponentielle Wachstum des Internets macht eine Anbindung von lokalen Netzwerken zunehmend attraktiver, da somit weitreichende Informationsdienste und entfernte Anwendungen auch von Rechnern des internen Netzes genutzt werden können. Mit einer Internetanbindung wachsen jedoch nicht nur die Möglichkeiten, sondern auch die Gefahren. Mit der zunehmenden Verbreitung des Internets in Wirtschaft und Privathaushalten seit Anfang der neunziger Jahre werden auch vermehrt Angriffe aus dem Internet beobachtet. Lag die Zahl der vom "Computer Emergency Response Team" (CERT) gemeldeten Sicherheitsvorfälle im Jahre 1989 nur bei 132, so waren es im Jahre 1996 bereits 2573, wobei die Zahl der betroffenen Netzwerke bei mehr als 40.000 lag [Kyas 1998, S. 23].

Läßt sich das Risiko eines Angriffs aus dem Internet z.B. durch Einsatz einer Firewall drastisch reduzieren [Chapman 1995], so bleibt die Möglichkeit eines Angriffs aus dem internen Netz. Studien belegen, daß mehr als zwei Drittel der Fälle von mißbräuchlicher Nutzung von Computersystemen durch Mitarbeiter des eigenen Unternehmens begangen werden [Kyas 1998, S. 19].

Die meisten der Angriffe nutzen Schwächen der Betriebssysteme, Netzwerkprotokolle und Netzwerkhardware aus. Insbesondere im Bereich der Netzwerkprotokolle und ihrer Implementationen sind seit Mitte der achtziger Jahre viele Schwachstellen aufgezeigt geworden [Bellovin 1989]. Diese ermöglichen das Fälschen von Absenderadressen von Datagrammen (IP-Spoofing) und darauf aufbauend das unautorisierte Aufbauen, Unterbrechen und sogar Übernehmen von Verbindungen [Morris 1985, Joncheray 1995].

Obwohl viele dieser Schwächen schon lange bekannt sind, wurde erst spät ein Handlungsbedarf erkannt, diese Probleme zu beseitigen. Mit wachsender Kommerzialisierung des Internets und zunehmender Verarbeitung sensitiver Daten auf Computern

sind die Folgekosten von Angriffen jedoch deutlich gestiegen [Kyas 1998, S. 55]. Zudem steigt der Bedarf an sicherer Kommunikation über unsichere Verbindungen. Dies gilt sowohl für weltweit operierende Unternehmen, die zwischen ihren Filialen über das Internet Daten austauschen müssen, als auch für Einzelpersonen, die Emails versenden oder Waren über das Internet bestellen.

Durch den Einsatz von kryptographischen Techniken wird versucht, Angriffe auf Kommunikationsbeziehungen zu verhindern bzw. zu erkennen. Zum einen läßt sich durch Verschlüsselung der zu übertragenden Daten die Vertraulichkeit bewahren, zum anderen werden Verschlüsselungssysteme verwendet, um die Authentizität des Absenders zu gewährleisten. Weiterhin werden mit sog. Einweg-Hashfunktionen Prüfsummen über Daten berechnet, um Integritätsprüfungen durchführen zu können, die Manipulationen an den übertragenen Dateneinheiten aufdecken.

Obwohl diese Verfahren schon seit längerer Zeit bekannt sind und bereits viel Erfahrung bezüglich ihrer Anwendung gesammelt wurde [Diffie 1992], besteht ein Mangel an plattform- und anwendungsunabhängigen Implementationen. Viele Bemühungen der letzten Jahre galten dem gezielten Einsatz von kryptographischen Algorithmen in bestimmten Anwendungen [Borman 1993, Lunt 1991, Ylönen 1996]. Erhöht der Einsatz dieser neuen oder veränderten Anwendungen auch die Sicherheit, so bedingt er gleichzeitig ein Umgewöhnen des Anwenders an die neuen Programme (neue Benutzerschnittstellen, Befehle, Fehlermeldungen etc.). Weitaus gravierender ist jedoch die Notwendigkeit, diese Anwendungen jeweils einzeln abzusichern, ihre Sicherheitsmechanismen zu überprüfen und zu konfigurieren und neu entdeckte Sicherheitslücken jeder einzelnen Anwendung zu beseitigen.

Wünschenswert ist hingegen eine Basis von Sicherheitsdiensten, die transparent für den Anwender arbeiten. Solche Dienste lassen sich innerhalb des Protokollstapels des Kommunikationssystems integrieren. Das Bereitstellen der kryptographischen Verfahren erfolgt dann entweder durch die Einführung eines neuen, sicheren Protokolls oder durch die Modifikation eines bestehenden Protokolls [O'Malley 1991].

In den letzten Jahren wurde oftmals der erstere Weg gewählt. Auf diese Weise sind zunehmend proprietäre Protokolle entstanden, die eine sichere Übertragung im Sinne von Vertraulichkeit, Authentizität und Integrität gewährleisten.[1] Diese werden i.d.R. in der Anwendungsschicht implementiert, d.h. sie bilden eine neue Schicht zwischen der Transportschicht und der Anwendung selbst [Dierks 1999, Ramaswamy 1989].[2] Dies bedeutet, daß bestehende Anwendungen, die diese Protokolle verwenden sollen, an die Protokolle angepaßt werden müssen. Auch hier ist für jede Anwendung eine Prüfung notwendig, ob die Teile, die das kryptographische Protokoll verwenden, korrekt implementiert wurden.

Ein Beispiel für die Modifikation eines bestehenden Protokolls sind die Sicherheits-

[1] Die Begriffe werden im Abschnitt 2.1 erläutert.

[2] In dieser Arbeit wird im wesentlichen auf die TCP/IP-Protokolle Bezug genommen, bei denen sich direkt oberhalb der Transportschicht bereits die Anwendungsschicht befindet. Im OSI Referenzmodell befinden sich zwischen Anwendung und Transportschicht noch Sitzungsschicht, Darstellungsschicht und Anwendungsschicht. Implementationen dieses Modells sind nicht sehr weit verbreitet, so daß die TCP/IP-Protokolle auch in der nächsten Zukunft vorrangig bleiben werden. Das OSI Modell liefert jedoch eine brauchbare Terminologie zur Beschreibung von Kommunikationssystemen, so daß diese für TCP/IP adaptiert wurde [Hunt 1992].

mechanismen des IP Protokolls, die unter dem Namen *IPsec* zusammengefaßt werden [Kent 1998a]. Diese Mechanismen beinhalten sowohl die Identitätsprüfung des Absenders eines Datagramms als auch die Gewähr von Integrität und Vertraulichkeit der übertragenen Daten [Huitema 1996, S. 97ff]. Die Verbreitung dieser Mechanismen schreitet jedoch schleppend voran, da sich die Integration in die bestehenden Implementationen von Version 4 des Internetprotokolls recht schwierig gestaltet. Die neue Version 6 des Internetprotokolls, die diese Mechanismen bereits enthält, setzt sich hingegen nur langsam durch, da zusätzliche Änderungen des Protokolls (z.B. Änderung des Adressierungsschemas) inkompatibel zur Version 4 sind. Zudem können auf diese Weise keine anwendungsabhängigen Sicherheitsdienste (im Sinne von Sicherheit als "Quality of Service" für Anwendungen) ermöglicht werden, da IP keine Ende-zu-Ende Kommunikation zwischen Prozessen bietet, also über keine Informationen über die Art der zugehörigen Anwendungen verfügt.[3]

Nach wie vor besteht also ein Mangel an sicheren Kommunikationsmöglichkeiten zwischen Anwendungen, die sowohl auf die Applikationen abgestimmte Sicherheitsdienstgüte als auch Transparenz bieten.

1.2 Zielsetzung

Ziel dieser Arbeit ist der Entwurf und die Implementation einer Protokollschicht, die die Anforderungen der Transparenz und der anwendungsabhängigen Sicherheitsdienstgüte erfüllt.

Dazu werden bestehende kryptographische Verfahren auf ihre Anwendbarkeit überprüft, um Vertraulichkeit, Integrität und Authentizität von Kommunikationsbeziehungen zu gewährleisten. Weiterhin wird untersucht, wie diese Verfahren in die bestehenden Kommunikationsprotokolle integriert werden können und in wie weit die Protokolle an diese Verfahren angepaßt werden müssen.

Auf diesen Analysen aufbauend wird eine Implementation realisiert, die die folgenden Dienstleistungen erbringen kann:

- Authentisierung von Anwendungen auf entfernten Rechnern
- Verschlüsselung der zu übertragenden Anwendungsdaten
- Integritätsprüfung der übertragenen Daten
- Generierung und sicherer Austausch neuer Schlüssel (z.B. bei lang andauernden Verbindungen sollte eine Änderung der Schlüssel erfolgen)
- anwendungsseitige und systemweite Konfiguration der Sicherheitsgüte

Anhand der Implementation soll der Nutzen der Verwendung dieser Sicherheitsdienste den entsprechenden Kosten gegenübergestellt werden (z.B. Grad von erreichter Sicherheit kontra Performanz).

[3] IPsec realisiert trotzdem anwendungsabhängige Sicherheitseinstellungen. Hierzu wird jedoch die Schichtstruktur des Protokollstapels durchbrochen.

1.3 Aufbau der Arbeit

Im zweiten Kapitel sollen zunächst einige einführende Erläuterungen der Problematik der sicheren Kommunikation gegeben und die möglichen Lösungsansätze aufgezeigt werden. Die grundlegenden Eigenschaften von Sicherheitsdiensten innerhalb eines Kommunikationssystems werden anhand des ISO Referenzmodells für offene Netze erläutert. Es folgt die Beschreibung der möglichen Sicherheitsdienste der einzelnen Protokollschichten, wobei auch die bestehenden Implementierungen vorgestellt werden.

In Kapitel 3 wird eine Grundstruktur eines sicheren Kommunikationssystems hergeleitet. Die wesentlichen Anforderungen an das zu erstellende System werden bewertet und ein entsprechender Systementwurf vorgestellt.

Das vierte Kapitel zeigt die wesentlichen Schritte der Umsetzung des Entwurfs in eine Implementation.

Im fünften Kapitel wird das Arbeiten mit dem erstellten Prototyp erläutert. Ebenso erfolgt eine Demonstration der Sicherheitsdienste im praktischen Einsatz.

Es folgt eine Bewertung der erstellten Implementatierung in Kapitel sechs, die auch Leistungsmessungen des erstellten Prototyps einschließt. Ein Ausblick auf zukünftige Entwicklungen bildet den Abschluß der Arbeit.

Kapitel 2

Sicherung von Kommunikationsbeziehungen

Dieses Kapitel gibt einen Überblick über grundlegende Konzepte sicherer Kommunikation. Zunächst wird erläutert, wie der Begriff "Sicherheit" in dieser Arbeit, speziell bezogen auf Kommunikation in Computernetzen, verwendet wird. Darauf folgt eine Übersicht über die bestehenden Verfahren, welche die Vertraulichkeit, Integrität und Authentizität von Daten gewährleisten. Der dritte Abschnitt beschreibt die bestehenden Möglichkeiten, diese Verfahren in ein Kommunikationssystem zu integrieren und betrachtet jeweils Beispiele für entsprechende Implementationen.

2.1 Begriffsbestimmung "Sicherheit"

Die Anzahl von lokalen und globalen Computernetzwerken wächst beständig. Die Nutzung von Netzwerken ist immer mehr eine unerläßliche Grundlage für Firmen und Institutionen. Daher muß ein *Netzwerkmanagement* dafür Sorge tragen, daß ein Netzwerk seine Aufgaben erfüllt. Das Netzwerkmanagement teilt sich dabei in die folgenden, fünf wichtigen Teilbereiche [Terplan 1992]:

Konfigurationsmanagement: Die Verwaltung der zum Netzwerk gehörigen Komponenten (Rechner, Leitungen, Router) und der installierten Software. Das Konfigurationsmanagement ist der zentrale Vorgang des Netzwerkmanagements. Die hierbei festgelegte Konfiguration aller Netzwerkkomponenten bildet die Informationsgrundlage für alle weiteren Teilbereiche des Netzwerkmanagements.

Fehlermanagement: Maßnahmen zur Erhaltung der Funktionsfähigkeit des Netzwerkes. Hierzu gehören z.B. Sicherungskopien von Daten und der Einsatz von redundanten Servern zum Erhöhen der Ausfallsicherheit.

Leistungsmanagement: Die ständige Evaluation der Netzwerkleistung und das Beseitigen von diesbezüglichen Schwachstellen (z.B. Engpässen).

Abrechnungsmanagement: *Accounting* bezeichnet den Vorgang des Sammelns und Auswertens von Informationen, die eine Abrechnung der genutzten Ressourcen

ermöglichen. Zum Accounting- oder Abrechnungsmanagement gehört die Identifizierung entsprechender Kostenfaktoren und die Bestimmung von entstandenen Kosten. Hierzu sind Abrechnungsverfahren nötig, die bestimmen, welche Dienstleistung wie bemessen wird (z.B. übertragene Pakete, Anzahl von Verbindungen).

Sicherheitsmanagement: Maßnahmen, die die Sicherheit des Netzwerkes und seiner Komponenten gewährleisten. Hierzu gehören Zugangskontrollen, Risikoanalysen, das Erstellen von Verhaltensmaßregeln ("policies") und Alarmmeldungen bei deren Verletzung. Der Einsatz von Verschlüsselungs- und Authentisierungsverfahren und die dafür notwendige Schlüsselverwaltung sind wichtige Teilbereiche des Sicherheitsmanagements.

Diese Bereiche sind keine disjunkten Teilmengen. So kann z.B. eine Redundanz von Servern sowohl die Ausfallsicherheit als auch die Leistung eines Systems verbessern. Maßnahmen des Sicherheitsmanagements haben Einfluß auf die Ausfallsicherheit des Netzwerkes. Weiterhin sollte das Sicherheitsmanagement die Richtigkeit der Accountingdaten gewährleisten. In jedem dieser Bereiche hat der Begriff "Sicherheit" einen anderen Bedeutungsumfang. Sicherheit im Sinne des Fehlermanagements beinhaltet z.B. die Ausfallsicherheit eines Systems, die Zuverlässigkeit einer Datenübertragung, die Fehlerbehandlung und Wartbarkeit [Schneider 1991, S. 725]. Der Begriff "Sicherheit" bezieht sich in diesem Dokument auf den Bereich des Sicherheitsmanagement. Mit dem Begriff "Sicherheit" ist demnach die folgende Eigenschaft gemeint:

Sicherheit: Die durch Schutzmechanismen erreichte Resistenz gegen unerwünschte Zugriffe.

Zunächst muß also definiert werden, welche Aktivitäten im Umgang mit den Systemen gewollt und nötig sind und welche Handlungen diesen Zielen entgegenstehen. Dies geschieht ganz allgemein in einer sog. Policy und detaillierter in Standards und Handbüchern [Großklaus 1999]. In Bezug auf die Sicherheit von Daten gibt es einige wesentliche Eigenschaften, die gewährleistet werden sollen [Stallings 1999]:[1]

Vertraulichkeit: Die übertragenen Daten sind nur von den rechtmäßigen Kommunikationspartnern einsehbar.

Integrität: Die übertragenen Daten erreichen den Empfänger in unverändertem Zustand.

Authentizität: Die übertragenen Daten stammen nachweislich vom angegebenen Absender.

Verfügbarkeit: Die Kommunikation läßt sich nicht durch fremde Dritte verhindern oder unterbrechen.

[1] Die Beschreibungen sind hier bereits auf Kommunikationsbeziehungen bezogen. Natürlich gelten diese Eigenschaften auch für lokale Systeme. Authentizität eines Benutzers wird z.B. durch einen Paßwortmechanismus sichergestellt, Vertraulichkeit von Dateien wird durch Zugriffsrechte ermöglicht.

Zusätzlich gibt es die Eigenschaft der Nichtleugbarkeit. Sie beinhaltet die Nachweisbarkeit der Übertragung von Nachrichten, so daß keiner der Kommunikationspartner später das Versenden, den Empfang und den Zeitpunkt der Datenübertragung bestreiten kann. Da sich Nichtleugbarkeit durch Kombination von Authentizität und Integrität ergibt, evtl. ergänzt um einen Zeitstempel, wird auf diese Eigenschaft hier nicht speziell eingegangen. Entsprechend der vier angeführten, grundlegenden Eigenschaften lassen sich unerwünschte Zugriffe wie folgt klassifizieren:

Sniffing: Ein Angreifer liest übertragene Nachrichten mit. Dies ist ein Angriff auf die Vertraulichkeit der Daten.

Man in the Middle: Ein Angreifer verändert die übertragenen Daten auf ihrem Weg zum Empfänger. Dies stellt einen Angriff auf die Integrität der Daten dar.[2]

Spoofing: Ein Angreifer schleust Nachrichten in das System ein, die mit einem falschen Absender versehen wurden. Hierbei handelt es sich um einen Angriff auf die Authentizität der Daten.

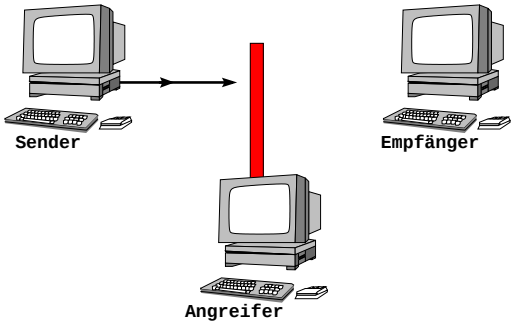
Denial of Service: Ein Bestandteil des Kommunikationssystems wird durch einen Angreifer zum erliegen gebracht, so daß dieser seine Aufgabe nicht mehr erfüllen kann. Dieser Angriff zielt auf die Verfügbarkeit der Kommunikationsdienste.

Abbildung 2.1 zeigt noch einmal die Angriffsmöglichkeiten.

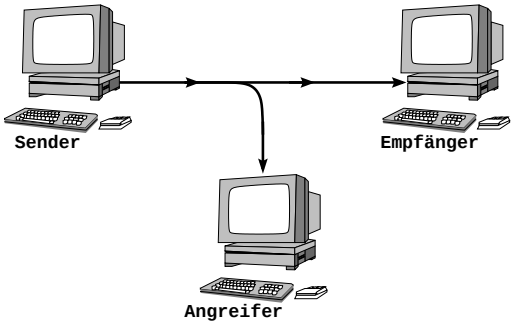
Vertraulichkeit, Authentizität, Integrität und Verfügbarkeit sind also keine selbstverständlichen Eigenschaften eines jeden Systems. Sie lassen sich nur durch den Einsatz von gezielten Maßnahmen erreichen. Somit sind sie das Ziel von Sicherheitsmanagement. Ein sicheres Kommunikationssystem muß für jede der gewünschten Eigenschaften der Datenübertragung Mechanismen bereitstellen, die entsprechende Angriffe verhindern. Diese Mechanismen werden als Sicherheitsdienste bezeichnet. Verfügbarkeit läßt sich nicht durch einen einzelnen Mechanismus gewährleisten, für alle Komponenten des Gesamtsystems (Anwendungen, Protokolle, Hardware) muß einzeln geprüft werden, ob sie in undefinierte oder ungewollte Zustände geraten können.

Die übrigen Dienste sind orthogonal zueinander. Durch Auswahl einiger dieser Dienste lassen sich die Sicherheitseigenschaften der Datenübertragung an die Anforderungen einer Anwendung anpassen. So ist z.B. für die Administration eines Routers über das Netzwerk die Integrität und die Authentizität der übermittelten Konfigurationsdaten essentiell, i.d.R. sind diese Daten jedoch nicht vertraulich. Bei der Übertragung einer Video- oder Audiosequenz hingegen ist die Integrität nur von untergeordneter Bedeutung, je nach Inhalt könnten jedoch Vertraulichkeit oder Authentizität erforderlich sein.

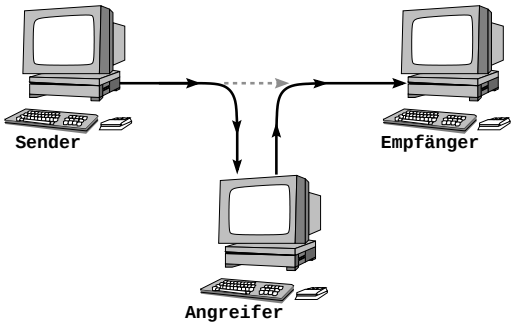
[2]Natürlich befindet sich der Angreifende, der eine Kommunikationsbeziehung beeinträchtigt, immer zwischen den beiden Endpunkten der Kommunikation. Der Terminus “Man in the Middle” wird jedoch i.d.R. nur auf diese Art von Angriff bezogen, wobei dieser oftmals mit einem Mitlesen der übertragenen Nachrichten verbunden ist.



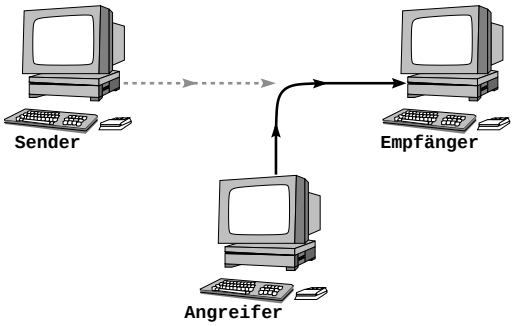
Denial of Service: Angriff auf die Verfügbarkeit



Sniffing: Angriff auf die Vertraulichkeit



Man in the Middle: Angriff auf die Integrität



Spoofing: Angriff auf die Authentizität

Abbildung 2.1: Verletzungen gewünschter Systemeigenschaften

2.2 Verfahren zur Gewährleistung von Sicherheit

Es existiert bereits eine Vielzahl von Verfahren, die jeweils eines oder mehrere der oben genannten Sicherungsziele realisieren. Diese sollen in den folgenden Abschnitten jeweils kurz vorgestellt werden. Die Aufzählung erfolgt nach den Diensten, die bereitgestellt werden.

2.2.1 Vertraulichkeit

Verschlüsselungsalgorithmen bieten die Möglichkeit, auch bei einer Übertragung über ein unsicheres Netzwerk die Vertraulichkeit von Daten sicherzustellen. Die Daten werden vom Sender verschlüsselt, in verschlüsselter Form übertragen und vom Empfänger wieder entschlüsselt. Ein unberechtigter Dritter, der die Datenübertragung abhört, erhält zwar die verschlüsselten Daten, kann sie jedoch ohne den entsprechenden Schlüssel nicht lesbar machen.

Die Verschlüsselung von Daten besteht aus einer mathematischen Verknüpfung des Klartextes M (*Message*) mit einem Schlüssel K (*Key*), wodurch der Chiffretext C entsteht. Die Verschlüsselung E (*Encryption*) stellt sich als $E_K(M) = C$ dar. Die Entschlüsselung D (*Decryption*) setzt C wieder um in M: $D_K(C) = M$. Diese Form

der Ver- und Entschlüsselung wird aufgrund der identischen Schlüssel bei beiden Vorgängen "symmetrische Kryptographie" genannt. Andere Verschlüsselungsalgorithmen, die sog. asymmetrischen Verfahren, benutzen verschiedene Schlüssel zum Ver- bzw. Entschlüsseln, so daß die Verschlüsselung $E_{K_1}(M) = C$ durch die Entschlüsselung $E_{K_2}(C) = M$ wieder aufgelöst wird. Diese Algorithmen werden auch als *Public-Key-Verfahren* bezeichnet. Auf beide Arten von Algorithmen wird in späteren Abschnitten eingegangen.

Ein Problem bei der Dechiffrierung einer Nachricht besteht, wenn der Ausgangstext aus beliebigen Bitfolgen gebildet werden kann, wie es z.B. bei Binärdateien der Fall ist. Es läßt sich für einen Chiffretext C nicht entscheiden, ob eine Dechiffrierung der Form $D_{K_1}(C)$ den korrekten Ausgangstext liefert. Schließlich liefert $D_{K_2}(C)$ für einen beliebigen Schlüssel K_2 ebenfalls eine Bitfolge. Welche der so entstehenden Bitfolgen ist der korrekte Ausgangstext? Hier müssen also Daten in den Ausgangstext eingebracht werden, die beim Entschlüsseln identifizierbar sind und daher eine maschinelle Erkennung einer inkorrekten Entschlüsselung ermöglichen [Stallings 1995, S. 207]. Diese Problematik des Erkennens der korrekten Entschlüsselung tritt auch bei der Anwendung kryptographischer Verfahren zur Sicherung von Integrität und Authentizität in Erscheinung.

Strom- und Blockchiffrierungen

Weiterhin gibt es Unterschiede in der Anwendung der Algorithmen auf die Daten. Die sog. Stromchiffrierungen konvertieren den Klartext jeweils bitweise zu Chiffretext. Dazu wird i.d.R. ein Pseudozufallszahlengenerator (auch Schlüsselstromgenerator genannt) verwendet, der in Abhängigkeit von einem Initialisierungswert in deterministischer Weise einen Strom von Nullen und Einsen erzeugt. Diese Folge von Bits sollte möglichst den Ergebnissen eines wiederholten Münzenwerfens ähnlich sehen [Rueppel 1992]. Das gerade zu verschlüsselnde Bit m_i des Klartextes wird durch ein *XOR* (exklusives Oder) mit dem entsprechenden Bit k_i des Schlüsselstromgenerators verknüpft; das Ergebnis ist das zugehörige Chiffretextbit c_i. Die ausgeführte Verknüpfungsfunktion ist also variabel über die Zeit. Der Schlüssel ist in diesem Fall der Initialisierungswert des Schlüsselstromgenerators. Zum Entschlüsseln dient derselbe Zufallszahlengenerator, der mit dem Schlüssel initialisiert wird und somit denselben Schlüsselstrom liefert. Somit wird ein Bit c_i des Chiffretextes mit demselben Bit-Wert k_i verknüpft, der bei der Verschlüsselung bereits auf dieses Bit angewendet wurde. Es entsteht wieder der Ausgangstext (Verschlüsselung: $m_i \oplus k_i = c_i$; Entschlüsselung: $c_i \oplus k_i = m_i$). Auf Grund der bitweisen Verarbeitung der Nachrichten sind Stromchiffrierungen besonders gut für Hardware-Implementationen von Verschlüsselungsverfahren geeignet, da die Bitoperationen sehr effizient von einer entsprechenden Hardware durchgeführt werden können [Schneier 1996, S. 211f].

Eine Alternative zu den Stromchiffrierungen stellen die Blockchiffrierungen dar. Sie arbeiten mit einer festen Transformation, die große Blöcke von Klartext jeweils in Chiffretext wandelt [Rueppel 1992]. Die Anwendung der Verschlüsselungsfunktion erfolgt dabei i.d.R. in einem von vier Standardmodi, die für den Algorithmus DES standardisiert wurden, jedoch auf jede Blockchiffrierung anwendbar sind [Kaufman 1995]. Dies sind im einzelnen:

Electronic-Codebook-Modus (ECB): Jeder Block wird einzeln für sich verschlüsselt. Dies bedeutet, daß zwei identische Klartextblöcke bei Verwendung desselben Schlüssels immer in identischen Chiffretext transformiert werden. Im ECB-Modus verhält sich ein Algorithmus also wie ein Substitutionsalgorithmus, der nach einem Lexikon die Klartextblöcke durch ihre zugeordneten Chiffretextblöcke ersetzt, daher der Name des Modus. Das Lexikon müßte jedoch sehr groß sein, bei 64 Bit Blöcken müßte es 2^{64} Einträge enthalten. Der ECB-Modus ist sehr effizient und einfach zu implementieren, er beinhaltet jedoch auch zwei wesentliche Schwächen. Zum einen ermöglicht das Substitutionsverhalten einem Angreifer, der mehrere Nachrichten im Klartext und im Chiffretext kennt, ein Lexikon von bereits bekannten Blöcken zu erstellen, um weitere Nachrichten zu entschlüsseln. Zum anderen können Nachrichten verändert werden, indem ganze Blöcke ausgetauscht werden. Man denke an eine elektronische Überweisung, die in einem 64-Bit-Block die Konto-Nummer des Empfängers enthält. Dieser entscheidende Block kann durch einen anderen ersetzt werden, so daß ein anderer Kunde das Geld erhält.

Cipher-Block-Chaining (CBC): Chaining (engl. *chain*: Kette) bezieht eine Rückkopplung in den Verschlüsselungsvorgang mit ein. Dabei wird i.d.R. ein Klartextblock M_i vor der Verschlüsselung jeweils mit dem Chiffretext des vorigen Blocks C_{i-1} XOR-verknüpft. Die Verschlüsselung ist dann durch $C_i = E_K(M_i \oplus C_{i-1})$ gegeben, die Entschlüsselung durch $M_i = C_{i-1} \oplus D_k(C_i)$. Es wird also ein Rückkopplungsregister benutzt, in dem der vorherige Block abgelegt wird, um beim nächsten Block verwendet zu werden. Der Chiffretext zweier identischer Blöcke ist nur dann gleich, wenn ihre Vorgängerblöcke identisch waren. Um zu verhindern, daß Nachrichten mit gleichem Anfang bis zur ersten Differenz den gleichen Chiffretext erhalten, wird als erster Block eine Zufallszahl verschlüsselt, die nicht zur Nachricht gehört. Diese Zufallszahl dient nur zur Initialisierung des Rückkopplungsregisters, sie wird Initialisierungsvektor genannt. Der CBC-Modus verhindert also offensichtlich einen Lexikonangriff. Allerdings wirkt sich ein Bitfehler im Chiffretext eines Blockes (z.B. durch fehlerhafte Übertragung der Nachricht) nicht nur auf den Block selbst, sondern auch auf das entsprechende Bit des folgenden Blocks aus. Ein Angreifer könnte auf diese Weise evtl. gezielt Bits im Klartext verändern. Das Anhängen von Blöcken an eine Nachricht durch einen Angreifer und das Auftreten identischer Blöcke bei sehr langen Nachrichten sind eher vernachlässigbare Probleme [Schneier 1996, S. 196].

Cipher Feedback Modus (CFB): Häufig wird beim Einsatz von Blockchiffrierungen die Möglichkeit benötigt, auch kleinere Dateneinheiten als ganze Blöcke zu verschlüsseln. So werden von manchen Anwendungen einzelne Bytes versendet (z.B. Telnet). Dies geschieht im Cipher Feedback Modus, indem ein Schieberegister, das einen ganzen Block aufnehmen kann, mit einem Initialisierungsvektor geladen wird. Der Inhalt I des Schieberegisters wird verschlüsselt und das oberste Byte k_j wird mit dem zu verschlüsselnden Klartextbyte m_i XOR-verknüpft. Das Ergebnis ist das Chiffrezeichen c_i. Der Inhalt des Schieberegisters wird um 8 Bits nach oben verschoben und das soeben generierte Chiffretextbyte c_i gelangt in die freigewordenen unteren 8 Bits. Somit werden verschlüsselte Bytes verwendet, um den Initialisierungsvektor zu mutieren. Bei der Entschlüsselung läuft der Vor-

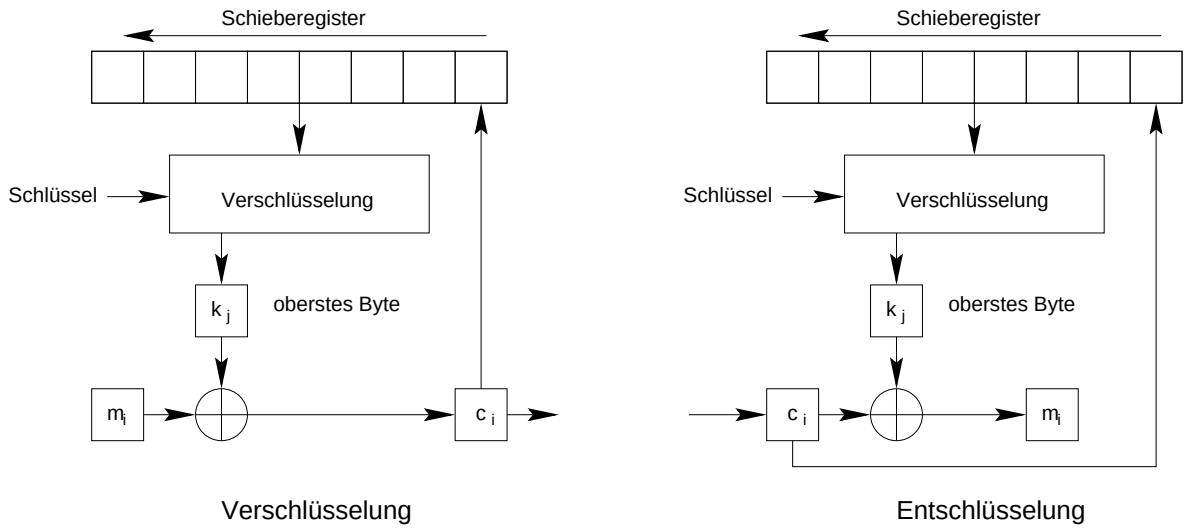


Abbildung 2.2: Cipher Feedback Modus (nach [Schneier 1996, S. 200])

gang exakt umgekehrt ab. Abbildung 2.2 verdeutlicht den Ablauf von Ver- und Entschlüsselung.

Bei einem Bitfehler im Chiffretext wirkt sich dieser nicht nur auf das entsprechende Zeichen aus, sondern auch auf den gesamten folgenden Block, da das falsche Bit noch solange im Schieberegister vorhanden ist. Durch die einfache XOR-Verknüpfung kann ein Angreifer bei bekanntem Klartext einige Bits gezielt verändern, um einen anderen Klartext zu erzeugen. Eine solche Änderung kann am Ende einer Nachricht nicht bemerkt werden, innerhalb einer Nachricht wird der folgende Block dadurch zerstört [Schneier 1996, S. 201].

Output Feedback Modus (OFB): Der OFB-Modus ähnelt dem CFB-Modus, es wird jedoch nicht der Chiffretext als Rückkopplung in das Schieberegister übertragen, sondern eine entsprechende Anzahl Bits aus dem Ausgabeblock der Verschlüsselung. Daher ist die Rückkopplung unabhängig vom zu verschlüsselnden Klartext. Dies bietet den Vorteil, für einen Initialisierungsvektor IV und einen Schlüssel K bereits Verschlüsselungsoperationen durchführen zu können, bevor der Klartext existiert. Die gewonnenen Werte müssen nur noch mit dem Klartext entsprechend XOR-verknüpft werden. Dies ist besonders günstig, da heutige Verschlüsselungsverfahren sehr rechenzeitintensiv sind. Weiterhin wirkt sich ein Ein-Bit-Fehler im Chiffretext auch nur auf ein Bit des entschlüsselten Klartextes aus. Dies ist bei Übertragung von digitalisierten Analogdaten (Audio, Video) von Vorteil, da Ein-Bit-Fehler oftmals nicht korrigiert werden müssen. Der Datenstrom muß jedoch synchronisiert werden, da ein verlorenes oder hinzugefügtes Bit den folgenden Teil der Nachricht unleserlich macht [Jueneman 1982]. Der generierte Schlüsselstrom, der mit dem Klartext durch XOR-Verknüpft wird, sollte bei Verwendung desselben Schlüssels im optimalen Fall keine Zyklen enthalten. Enthält er große Zyklen, so ist die Wahrscheinlichkeit groß, von einem beliebigen Initialisierungsvektor in den Zyklus zu geraten. In kleine Zyklen zu geraten ist zwar nicht so wahrscheinlich,

bedeutet jedoch, daß der Schlüsselstrom nur wenige verschiedene Werte annimmt. Erwünscht ist eine durchschnittliche Zyklenlänge, die möglichst dicht an die Anzahl der möglichen Werte heranreicht, die der Schlüsselstrom annehmen kann (bei einem 64 Bit Algorithmus also dicht an 2^{64}). Dies kann dadurch sichergestellt werden, daß die Anzahl der Rückkopplungsbits gleich der Länge des Schieberegisters ist. Für geringere Rückkopplungsbreiten sinkt die durchschnittliche Länge der Zyklen im Schlüsselstrom drastisch [Davies 1982].

Neben den vier genannten Betriebsarten von Blockchiffrierungen gibt es noch andere, die jedoch selten Verwendung finden. Weiterhin gibt es auch Kombinationen mehrerer solcher Betriebsarten. Exotische und komplexere Betriebsarten erhöhen die Sicherheit i.d.R. jedoch nicht und sind den aufgeführten Standardmodi weder in Bezug auf Robustheit noch auf Fehlerfortpflanzung überlegen [Schneier 1996, S. 208].

Symmetrische Kryptographie

Die symmetrischen kryptographischen Verfahren nutzen denselben Schlüssel zur Ver- und Entschlüsselung von Nachrichten. Dies bedeutet, daß jeweils ein Schlüssel zwischen den Kommunikationspartnern vereinbart werden muß. Die Sicherheit dieser Verfahren hängt im wesentlichen davon ab, daß dieser Schlüssel nur den rechtmäßigen Kommunikationspartnern bekannt sein darf. Das Problem des vertraulichen Nachrichtenaustauschs wird somit reduziert auf das Problem des vertraulichen Austauschs von Schlüsseln.

Für die symmetrischen Verfahren werden i.d.R. Substitutions- und Transpositionsoperationen verwendet. In Abhängigkeit vom Schlüssel werden Bits oder Bitfolgen des Ausgangstextes durch andere Bits ersetzt oder ihre Reihenfolge permutiert[3]. Die Sicherheit einer durchgeführten Chiffrierung ist sowohl abhängig von der Länge des verwendeten Schlüssels als auch von der Anzahl der durchgeführten Verschlüsselungsvorgänge. Das sog. “One Time Pad” stellt dabei das eine Extrem dar, bei dem nur eine Ersetzungsoperation (XOR-Verknüpfung) benutzt wird. Zum Verschlüsseln einer m Bits langen Nachricht wird ein völlig zufälliger Schlüssel der Länge m benutzt, der nur einmal Verwendung findet. Somit ergibt sich keine Möglichkeit, eine derart verschlüsselte Nachricht ohne Kenntnis des Schlüssels zu dechiffrieren, da der chiffrierte Text keinerlei statistische Beziehung zum Ausgangstext aufweist [Stallings 1999, S. 40f]. Das andere Extrem stellen die Rotormaschinen dar, die mehrere Ersetzungsoperationen verwenden. Ein Zylinder, der 26 Eingänge mit 26 Ausgängen verbindet, liefert die Ersetzung des Eingabealphabetes. Mit jeder erfolgten Ersetzung wird der Zylinder um eine Position gedreht, so daß eine neue Ersetzungschiffre definiert wird [Schuchmann 1982]. Der Einsatz von mehreren Zylindern hintereinander führt zu mehreren Ersetzungsoperationen für jedes einzelne Zeichen. Den Schlüssel stellen hierbei die durch die Zylinder definierten Zuordnungen von Eingängen zu Ausgängen und die Ausgangsposition der Zylinder dar. Durch Kombination von Ersetzungs- und Transpositionsoperationen in mehreren Verschlüsselungsstufen (sog. Runden) läßt sich eine Chiffrierung erzielen, die nur durch Ausprobieren aller möglichen Schlüssel angreifbar ist, so daß der Aufwand zu groß ist, um noch praktikabel zu sein.

[3] Für eine besonders anschauliche Einführung siehe [Stallings 1999, S. 28ff].

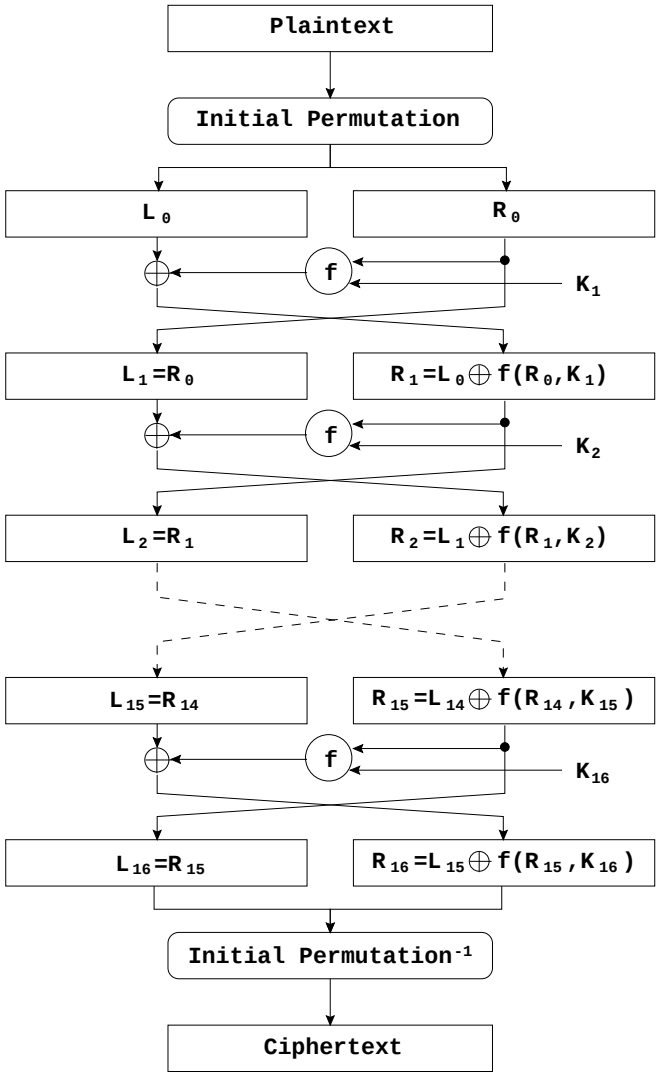


Abbildung 2.3: Die 16 Runden des DES Algorithmus [Schneier 1996].

Das bekannteste Verfahren dieser Art ist der *Data Encryption Standard* (DES) [Schneier 1996, S. 265], bei dem jeweils Blöcke von 64 Bit Länge mit einem 56-Bit-Schlüssel chiffriert werden. Hierbei wird nach einer Anfangspermutation der Eingabeblock in zwei 32-Bit-Blöcke geteilt. In sechzehn Runden wird jeweils der rechte Block auf 48 Bits expandiert und mit einem auf 48 Bits reduzierten Rundenschlüssel (gewonnen aus dem 56-Bit-Schlüssel) XOR verknüpft. In Abhängigkeit von der Eingabe erfolgen dann eine Ersetzung einzelner Bits, die eine 32 Bit Ausgabe erzeugt, und eine Bitvertauschung. Nun wird der rechte Block mit dem linken Block XOR verknüpft und das Ergebnis mit dem linken Block vertauscht, so daß in der nächsten Runde der vormalige linke Block die Ersetzungs- und Vertauschungsoperationen durchläuft. Nach sechzehn Runden hat jedes Bit des Schlüssels und jedes Bit des Ausgangstextblocks alle Bits des Chiffretextblocks beeinflußt. Abbildung 2.3 veranschaulicht die Verknüpfungsoperationen der einzelnen Runden. DES ist jedoch nicht mehr ausreichend, um sensitive Daten

zu schützen.[4]

Die auszuführenden Operationen eines symmetrischen Verfahrens sind i.d.R. XOR-Verknüpfungen, Bitshifts, Bitrotationen, Bitvertauschungen, Multiplikationen und Additionen. Bis auf die Bitvertauschungen sind diese bereits in heutigen Prozessoren verankert, so daß die Ver- und Entschlüsselung auch in Software-Implementationen effizient sind. Durch spezielle Hardware-Implementationen lassen sich noch deutliche Steigerungen erzielen [Schneier 1996, S. 279].

Symmetrische Verfahren sind weiterhin geeignet, auch die Authentizität von Daten zu garantieren, da nur der rechtmäßige Kommunikationspartner den Schlüssel kennt und somit zur korrekten Verschlüsselung fähig ist.

Schwierigkeiten bereitet bei den symmetrischen Verfahren jedoch die Schlüsselverteilung. Bei "Ende-zu-Ende Verschlüsselung" (siehe Abschnitt 2.3.3) muß für jede mögliche Gruppe von Kommunikationspartnern ein spezieller Schlüssel vereinbart werden, der nur für diese gilt. Für n Kommunikationspartner wären bis zu n^n Schlüssel nötig. Diese müßten vor Beginn der Kommunikation sicher an die Teilnehmer verteilt werden [Smith 1997, S. 197f]. Übernimmt diese Aufgabe ein sog. Keyserver, so müssen zunächst n Schlüssel (*Key Encryption Keys*) für die vertrauliche Kommunikation mit dem Server sicher auf die n Partner verteilt werden. Der Keyserver erhält alle Schlüssel. Zum Aufbau einer sicheren Verbindung kontaktieren die Kommunikationspartner jeweils den Server, wobei sie ihren eigenen Schlüssel benutzen. Sie erhalten einen gemeinsamen Schlüssel (*Data Key*) und können damit direkt miteinander vertraulich kommunizieren. Diese Serverlösung skaliert nicht in größeren Netzwerken, zudem bietet der einzelne Keyserver einen zentralen Angriffspunkt [Kyas 1998, S. 287]. Diese Schwierigkeiten führten zur Entwicklung von sog. asymmetrischen Verfahren.

Asymmetrische Kryptographie

Asymmetrische Kryptographie verwendet ein Paar von verschiedenen Schlüsseln K_{public} und K_{secret} zum Ver- bzw. Entschlüsseln. K_{public} ist der sog. öffentliche Schlüssel (engl. *public key*), der der Allgemeinheit bekannt gemacht werden kann. Ein Ausgangstext, der mit dem Schlüssel K_{public} chiffriert wurde, kann nur mit dem zugehörigen geheimen Schlüssel (engl. *secret key*) K_{secret} entschlüsselt werden. Der Schlüssel K_{secret} steht zwar in einem Zusammenhang mit K_{public}, läßt sich jedoch aus diesem nicht mit einem

[4] Zum einen gibt es einige Schlüssel, die zu einer sehr einfachen Chiffrierung führen. Dies sind z.B. Schlüssel, bei denen jede Hälfte des Schlüssels jeweils nur aus Einsen oder aus Nullen besteht. Die Rundenschlüssel werden nur durch Bitrotationen innerhalb der beiden Hälften aus dem Hauptschlüssel gewonnen, so daß diese Hauptschlüssel für jede Runde identische Rundenschlüssel generieren. Es gibt einige weitere Schlüssel mit Schwächen, jedoch sind es nicht sehr viele, so daß diese leicht ausgeschlossen werden können [Schneier 1996, S. 280]. Die Verwendung eines 56-Bit-Schlüssels führt jedoch zu maximal 2^56 verschiedenen möglichen Schlüsseln. Ein Ausprobieren aller Schlüssel läßt sich durch Aufteilung des gesamten Schlüsselraumes in Abschnitte, die dann von einzelnen Prozessoren durchforstet werden, sehr gut parallelisieren. Ein gemeinsamer Einsatz von über das Internet verbundenen Rechnern erreichte dies bei der zweiten 'DES Challenge' (ein Wettbewerb, der von der Firma RSA ausgerichtet wird) im Januar 1998 nach 40 Tagen. Der Einsatz von stark parallelen Rechnern mit speziellen Prozessoren kann diesen Zeitraum bis auf wenige Stunden verkürzen [Wiener 1994]. In der Praxis wurde ein mit DES chiffrierter Text auf diese Weise in unter 23 Stunden entziffert (Dritte 'DES Challenge').

praktikablen Aufwand herleiten. Die Verschlüsselungsfunktion ist eine sog. Einwegfunktion mit Hintertür, d.h. ihre Umkehrung läßt sich nur mit einer Zusatzinformation (Hintertür) bilden. Diese Zusatzinformation ist der geheime Schlüssel. Die asymmetrische Kryptographie beruht also im wesentlichen auf der (nicht bewiesenen) Existenz von Einwegfunktionen. Der Entwurf eines asymmetrischen Verfahrens geht i.d.R. von einem schwierigen Problem aus[5] und beinhaltet das Lösen zweier Fragestellungen [Stallings 1995, S. 156]:

- Wie kann das Problem in ein Verfahren zur Ver- und Entschlüsselung verwandelt werden?
- Wie kann eine Hintertürinformation aufgenommen werden, die eine einfache Lösbarkeit des Problems ermöglicht?

Das Aufnehmen einer Hintertürinformation gestaltet sich dabei als besonders schwierig, da hierbei dafür Sorge getragen werden muß, daß die Komplexität des Problems nicht verringert wird, wie dies z.B. bei den Knappsack Verschlüsselungsverfahren der Fall war [Adleman 1982], [Brickell 1984].

Das einem Algorithmus zugrundeliegende Problem bestimmt die Rechenoperationen, die beim Ver- und Entschlüsseln jeweils Verwendung finden. Dies sind bei den heute bekannten Verfahren gewöhnlich Exponentiation bzw. Multiplikation. Asymmetrische Verfahren sind daher bei weitem nicht so effizient wie symmetrische. Bei Hardware-Implementationen ist z.B. der meistverwendete, asymmetrische Algorithmus, RSA, tausendmal langsamer als der symmetrische DES [Schneier 1996, S. 469]. Zudem sind die Schlüssel bei vergleichbarer Sicherheit wesentlich länger als bei den symmetrischen Verschlüsselungsalgorithmen. Manche der asymmetrischen Verfahren erzeugen auch Chiffretext, der wesentlich länger ist als der Ausgangstext. Die Effizienz der asymmetrischen, kryptographischen Algorithmen ist also offensichtlich ungeeignet, um Datenblöcke in Echtzeit zu ver- und entschlüsseln, die über eine Datenleitung mit hoher Bandbreite übertragen werden [Nechvatal 1992, S. 186].

Eine weitere inhärente Schwäche von asymmetrischen Verfahren besteht darin, daß sog. “Known Plaintext” Angriffe möglich sind. Da der öffentliche Schlüssel für jedermann zugänglich ist, kann ein Angreifer für jeden beliebigen Ausgangstext den zugehörigen Chiffretext erzeugen. Dies ermöglicht die Analyse von Zusammenhängen zwischen beiden Texten, was einen Angriff erheblich erleichtert [Smith 1997, S. 69].

Auf der anderen Seite bieten asymmetrische Verfahren die Möglichkeit, eine vertrauliche Kommunikationsbeziehung aufzubauen, ohne vorher einen vertraulichen Schlüsselaustausch durchzuführen. Jeder Benutzer benötigt nur noch ein Schlüsselpaar, von dem der geheime Teil nie übertragen werden muß. Das Problem der Schlüsselverteilung, wie es bei symmetrischer Kryptographie auftritt, gibt es hier nicht.

Aus diesen gegensätzlichen Eigenschaften beider Verschlüsselungsverfahren ergibt sich ein vorteilhafter gemeinsamer Einsatz innerhalb eines Systems. Dabei werden asymmetrische Verfahren benutzt, um für eine Kommunikationsbeziehung einen gemeinsamen,

[5] Schwierig ist in diesem Fall auf den Berechnungsaufwand der Lösung bezogen. Gewöhnlich werden Probleme gewählt, deren Berechnung mit heutigen Verfahren einen exponentiellen Aufwand erfordert, die also in den Komplexitätsklassen NP oder besser NP-vollständig liegen.

geheimen Schlüssel vertraulich auszutauschen. Hierbei spielt die geringe Effizienz der asymmetrischen Verfahren eine untergeordnete Rolle, da nur eine sehr geringe Datenmenge ausgetauscht wird. Der auf diese Weise übertragene Schlüssel wird dann für die Ver- und Entschlüsselung mit einem symmetrischen Verfahren benutzt. Dies gewährt die für die Datenübertragung benötigte, höhere Performanz [Nechvatal 1992, S. 187].

2.2.2 Integrität

Im Falle der Gefahr einer zufälligen Veränderung von Bits bei der Übertragung einer Nachricht werden oftmals Prüfsummen benutzt, um diese Fehler festzustellen. Eine solche Prüfsumme (auch "Message Digest") ist eine Funktion H aller Bits einer Nachricht M. Sie bildet eine Eingabe M beliebiger Länge auf eine Ausgabe h ab, die eine feste, wesentlich kürzere Länge m hat: $h = H(M)$. Der Sender generiert die Prüfsumme h und überträgt diese gemeinsam mit der Nachricht M an den Empfänger. Dieser berechnet die Prüfsumme erneut auf Grund der empfangenen Nachricht und vergleicht das Ergebnis mit der übertragenen Prüfsumme. Sind diese identisch, so ist die Nachricht mit großer Wahrscheinlichkeit unverändert empfangen worden. Im Falle der häufigen Anwendung dieser Funktionen ist auch ihre Performanz ein wichtiges Kriterium. Zu einem gegebenen M sollte $H(M)$ leicht und effizient berechenbar sein. Bekannte Beispiele für solche Prüfsummen sind Paritätsbits und sog. *Cyclic Redundancy Checks* (CRC).

Diese Verfahren sind jedoch nur geeignet, Integrität gegenüber zufälligen Veränderungen, wie z.B. einer Leitungsstörung bei der Datenübertragung, zu gewährleisten. Gegenüber gezielten Manipulationen bieten sie jedoch keinen Schutz. Ein Angreifer könnte nach wie vor eine Nachricht abfangen, diese verändern, die Prüfsumme erneut berechnen und die modifizierte Nachricht weiterleiten.

Wenn die Kommunikationspartner die Möglichkeit der vertraulichen Kommunikation besitzen, kann der Ausgangstext verschlüsselt übertragen werden. Dies sichert die Integrität der Daten jedoch nur dann, wenn ein korrekter Ausgangstext von einem inkorrekten unterscheidbar ist. Dies ist dann der Fall, wenn die übertragenen Daten feststehende Strukturen besitzen, bei beliebigen Binärdaten ist dies jedoch nicht möglich (siehe Abschnitt über Verschlüsselungsverfahren, S. 9). Das Anhängen oder Ändern von Daten am Ende der Nachricht oder der Austausch von einzelnen Bits oder ganzen Blöcken kann so allerdings auch nicht vollständig verhindert werden (siehe vorherige Ausführungen zu den Standardmodi der Blockchiffrierungen, S. 9ff).

Zudem sollte die Eigenschaft der Integrität auch ohne Verschlüsselung der gesamten Daten gewährleistet werden können, um den nötigen Aufwand zu minimieren. Im Falle eines vorhandenen symmetrischen Verschlüsselungsverfahrens kann eine gewöhnliche CRC Prüfsumme in verschlüsselter Form übertragen werden. Ein Angreifer kann zwar die korrekte Prüfsumme berechnen, sie jedoch nicht entsprechend verschlüsseln. Bei asymmetrischen Verfahren geht dies jedoch i.d.R. nicht, da der öffentliche Schlüssel zum Verschlüsseln auch dem Angreifer bekannt ist. Einzelne Verfahren wie RSA, bei denen ein Chiffretext, der mit dem geheimen Schlüssel generiert wurde, mit dem öffentlichen Schlüssel dechiffriert werden kann, können auf diese Weise auch zum Verschlüsseln der Prüfsumme verwendet werden.

Die bisher genannten Prüfsummen bieten zwar einen Schutz gegen zufällige Verände-

rungen der zu übertragenden Nachricht, es ist hingegen eine weitere Eigenschaft der Prüfsummenfunktion erforderlich, um einem gezielten Angriff entgegenzuwirken. Sog. Einweg-Hashfunktionen bilden ebenfalls Nachrichten beliebiger Länge auf eine Prüfsumme fester Länge ab, besitzen jedoch (unter anderem) die folgende zusätzliche Eigenschaft, die im Zusammenhang mit der Integrität von Nachrichten wichtig ist:

- Zu einer gegebenen Nachricht M ist es schwer, eine andere Nachricht M' zu finden, so daß $H(M) = H(M')$.

Diese Eigenschaft stellt sicher, daß eine mit einem verschlüsselten Hashwert versehene Nachricht M nicht durch eine andere Nachricht M' ausgetauscht werden kann, ohne daß der Hashwert sich verändert. Eine Hashfunktion, die diese Eigenschaft erfüllt, wird schwache Hashfunktion genannt [Stallings 1995, S. 224].

Im Zusammenhang mit digitalen Unterschriften wird noch eine weitere Anforderung an die Hashfunktion gestellt:

- Es ist schwer, ein beliebiges Paar (M, M') von Nachrichten zu finden, für das $H(M) = H(M')$ gilt.

Einweg-Hashfunktionen mit dieser Eigenschaft werden als "starke" Hashfunktionen bezeichnet, eine Funktion, bei der ein solches Paar nicht existiert, nennt man "kollisionsfrei".[6] oder auch kollisionsfreie Hashfunktionen bezeichnet. Sie schützen auch vor sog. "Birthday-Angriffen". Hierbei werden für eine m-Bit-Prüfsumme zunächst $2^{m/2}$ Nachrichten mit harmlosem Inhalt generiert, z.B. durch jeweiliges Einsetzen unterschiedlicher Wörter von gleicher Bedeutung. Weiterhin wird auf diese Art dieselbe Anzahl an brisanten Nachrichten erzeugt. Es ergibt sich eine Wahrscheinlichkeit von über 50%, hierbei ein Paar aus einer harmlosen und einer brisanten Nachricht zu finden, die beide denselben Hashwert haben.[7] Die harmlose Nachricht des gefundenen Paares kann einer Person zur Unterschrift vorgelegt werden. Die mit der Einweg-Hashfunktion berechnete elektronische Unterschrift kann dann ebenfalls an die brisante Nachricht angehängt werden, da der Hashwert übereinstimmt. Die Länge eines Hashwertes ist also offensichtlich wichtig, um den Aufwand solcher Angriffe so zu vergrößern, daß sie unpraktikabel sind.

Um Hashfunktionen mit den genannten Eigenschaften zu konstruieren wird eine Kompressionsfunktion benutzt, die eine Eingabe der Länge m in eine Ausgabe der Länge n umwandelt, wobei $n < m$ ist. Diese Einwegfunktion erhält als fortlaufende Eingaben jeweils den nächsten Block der Nachricht und den Ausgabewert des vorigen Blocks; der Hashwert h_i des Nachrichtenblocks M_i ist also gegeben durch $h_i = f(M_i, h_{i-1})$. Für die erste Ausgabe wird ein Startblock generiert. Um zu verhindern, daß Nachrichten unterschiedlicher Länge möglicherweise auf denselben Hashwert abgebildet werden, wird in den Startblock die Länge der Gesamtnachricht aufgenommen. Diese Technik wird als "MD-strengthening" (Verstärkung des "Message Digest") bezeichnet [Schneier 1996, S. 431].

[6] Wenn die Ausgabe der Hashfunktion kürzer ist als die Eingabe, ist Kollisionsfreiheit unmöglich.

[7] Eine Erklärung des "Birthday-Paradoxons" findet sich in [Stallings 1995]

Bekannte Funktionen sind MD4, MD5 und SHA ("Secure Hash Algorithm"), die nach demselben Schema arbeiten. Sie erweitern die Nachricht um einige Bits, bis die Länge ein Vielfaches von 512 Bits beträgt. Daraufhin werden jeweils 512-Bit-Blöcke der Nachricht in kleinere Blöcke zerlegt. Diese Blöcke werden sowohl miteinander als auch mit vier (MD4, MD5) oder 5 (SHA) Startvariablen durch Funktionen verknüpft, die sich aus Additionen, den logischen Operationen UND, ODER, XOR, NICHT und Bitrotationen zusammensetzen. Diese Verknüpfungen werden in mehreren Runden ausgeführt. Auf diese Weise werden vier bzw. fünf Werte erzeugt, die jeweils zu den entsprechenden Startvariablen hinzuaddiert werden. Mit diesen neuen Startwerten wird der nächste 512-Bit-Block verarbeitet. Die Werte, die nach Bearbeitung des letzten Blocks der Nachricht in den Startvariablen stehen, werden zum Gesamtergebnis des Hash-Algorithmus konkateniert. Somit berechnen MD4 und MD5 128-Bit-Hashwerte (4 Startvariablen mit 32-Bit Länge), SHA erzeugt ein 160-Bit-Ergebnis (5 Variablen).[8]

Ein weiterer Hash-Algorithmus ist MD2 [Kaliski 1992], der auf einer Permutation von Bytes beruht, die von den Stellen der Zahl π abhängt. Er arbeitet jedoch wesentlich langsamer als die oben genannten Verfahren [Schneier 1996, S. 435].

Auch symmetrische Verschlüsselungsverfahren werden benutzt, um Einweg-Hashfunktionen zu realisieren. Hierbei wird ein Blockverfahren im CBC oder CFB Modus mit feststehendem Schlüssel und Initialisierungsvektor benutzt. Der Hashwert ist der letzte verschlüsselte Ausgabeblock des Algorithmus. Ein besseres Verfahren benutzt jeweils den Nachrichtenblock als Schlüssel und den vorherigen Hashwert als Eingabe, um den entsprechenden Ausgabewert zu erzeugen. Ein generelles Konzept dieser Verfahren ist gegeben durch folgende Beschreibung [Schneier 1996, S. 447]:

- $H_0 = I_H$, wobei I_H der Initialisierungsvektor ist.
- $H_i = E_A(B) \oplus C$, wobei A, B und C jeweils der Nachrichtenblock M_i, der Vorgängerwert H_{i-1}, $(M_i \oplus H_{i-1})$ oder eine Konstante sein können.

Eine weitere Möglichkeit ist der Einsatz eines asymmetrischen Verschlüsselungsverfahrens in einem verkettenden Modus (z.B. CBC). Auch hier bildet der letzte Ausgabeblock den erzeugten Hashwert. Wird nun der geheime Teil des Schlüsselpaares verworfen, so ist eine Umkehrung der Funktion genauso schwierig wie das Entschlüsseln eines Chiffretextes ohne geheimen Schlüssel. Nachteil dieses Verfahrens ist die geringe Performanz, die aus der Verwendung eines asymmetrischen Verfahrens beruht.

Zur Sicherung der Integrität existiert also eine Reihe von Verfahren, die jeweils eine Prüfsumme der zu versendenden Nachricht berechnen. Diese Algorithmen sind i.d.R. wesentlich effizienter als eine Verschlüsselung der gesamten Nachricht. Lediglich die Prüfsumme muß verschlüsselt übertragen werden, um eine Veränderung durch einen Angreifer zu verhindern. Dazu kann ein symmetrisches Verschlüsselungsverfahren benutzt werden. Die asymmetrischen Verfahren, bei denen die Reihenfolge der verwendeten Schlüssel vertauschbar ist ($D_{K_{public}}(E_{K_{secret}}(M)) = M$), können hierfür auch Verwendung finden.

[8] Die exakten Beschreibungen der Algorithmen finden sich in [Rivest 1992a] (MD4), [Rivest 1992b] (MD5) und [NIST 1995] (SHA).

2.2.3 Authentizität

Kryptographische Verfahren sind auch geeignet, die Authentizität von Kommunikationspartnern sicherzustellen. Symmetrische Verschlüsselungsverfahren gewährleisten per se eine beschränkte Form von Authentizität, da nur rechtmäßige Kommunikationspartner in der Lage sind, Ausgangstext mit dem benötigten Schlüssel zu verschlüsseln bzw. den Chiffretext zu entschlüsseln. Somit sind zwar die beteiligten Kommunikationspartner von der Authentizität ihres Gegenparts überzeugt, es läßt sich jedoch im Streitfalle weder die Integrität noch die Authentizität der Nachrichten beweisen; jeder Partner kann den anderen imitieren, da beide den Schlüssel besitzen [Mitchell 1992, S. 329]. Lösungen dieses Problems ziehen i.d.R. einen dritten, vertrauenswürdigen Rechner hinzu, so daß z.B. die Kommunikation zwischen A und B über diesen Vermittler C abgewickelt wird. A und B haben jeweils eigene Schlüssel, um mit C zu kommunizieren, C entschlüsselt die Nachrichten von A und verschlüsselt sie mit dem Schlüssel von B, bevor er sie weiterleitet. Eine weitere Möglichkeit wäre das Verteilen von Einmalschlüsseln durch den Vermittler, die Kommunikation kann dann wieder direkt zwischen A und B erfolgen. Diese Vermittler ("Authentication Server") sind wie alle zentralen Dienste eine mögliche Schwachstelle sowohl in Bezug auf die Performanz als auch in Bezug auf die Sicherheit, da sie ein lohnendes Ziel für einen Angriff darstellen.

Asymmetrische Verfahren benötigen in jedem Fall ein getrenntes Verfahren, das Authentisierung ermöglicht. Dies hat zwei Ursachen:

1. Die Verschlüsselung ergibt keine Verifizierung des Absenders, da jeder Angreifer eine beliebige Nachricht erzeugen kann; schließlich ist der öffentliche Schlüssel frei verfügbar.
2. Der Sender von vertraulichen Informationen muß sich zusätzlich vergewissern, daß ein erhaltener, öffentlicher Schlüssel auch wirklich zum gewünschten Empfänger gehört und nicht von einem Angreifer erstellt wurde.

Das Problem 2 kann durch sog. Zertifikate gelöst werden, wobei eine vertrauenswürdige Instanz sich von der Zugehörigkeit eines Schlüssels zu einem Benutzer überzeugt und den Dienst anbietet, diese Zugehörigkeit über das Netzwerk zu bestätigen. So wird das Problem verlagert auf die Vertrauenswürdigkeit der Zertifizierungsinstanz [Liedtke 2000].

Die asymmetrischen Algorithmen, bei denen die Reihenfolge der verwendeten Schlüssel umkehrbar ist ($D_{K_{public}}(E_{K_{secret}}(M)) = D_{K_{secret}}(E_{K_{public}}(M))$), ermöglichen eine einfache Lösung des ersten Problems. Einen mit K_{public} dechiffrierbaren Text kann nur der Besitzer des zugehörigen geheimen Schlüssels durch $E_{K_{secret}}(M)$ erzeugt haben.

Sowohl für symmetrische als auch für asymmetrische Verfahren existiert jedoch auch hier das Problem, daß sich im Falle von beliebigen Binärdaten das Ergebnis einer korrekten Entschlüsselung D_{K_1} nicht von einer Entschlüsselung mit einem anderen, falschen Schlüssel K_2 unterscheiden läßt (siehe S. 9).

Ein allgemeines Verfahren, daß sich für alle asymmetrischen Algorithmen eignet, bietet das folgende Protokoll [Schneier 1996, S. 54]:

1. Kommunikationspartner A sendet an B eine zufällige Zeichenkette s.

2. B bildet die Verschlüsselung $E_{K_{secret}}(s)$ von s mit seinem geheimen Schlüssel K_{secret} und sendet das Ergebnis c zurück an A.

3. A ermittelt den öffentlichen Schlüssel K_{public} von B aus seiner Datenbank und bildet die Entschlüsselung $D_{K_{public}}(c)$.

4. Wenn $D_{K_{public}}(c) = s$ ist, kann A sicher sein, daß er mit B kommuniziert, da nur B die Verschlüsselung von s durchführen kann.

Dieses Protokoll läßt sich so erweitern, daß B nicht beliebige Zeichenketten von unbekannten Rechnern mit seinem geheimen Schlüssel verschlüsseln muß. Dies geschieht, indem B zuerst selbst generierte Zufallsdaten verschlüsselt und an A sendet. Diese Daten werden in den Schritten 2 und 3 jeweils mit in die Berechnungen einbezogen [Schneier 1996, S. 54].

Auch für die Gewährleistung der Authentizität ist es ausreichend, nur einen kleinen verschlüsselten Datenblock zu übertragen, um den Verschlüsselungsaufwand zu reduzieren. Der Datenblock besteht ebenfalls aus einer Prüfsumme, die über die gesamte Nachricht gebildet wurde, so daß diese zum einen die Integrität sicherstellt, zum anderen durch ihre Verschlüsselung die Authentizität gewährleistet. Diese verschlüsselte Prüfsumme wird als “Message Authentication Code” (MAC) oder auch “Authenticator” bezeichnet [Simmons 1992, S. 385].

Es bestehen also Verfahren zur Sicherung der Authentizität auf der Basis von symmetrischen und asymmetrischen Verschlüsselungsalgorithmen. Die asymmetrischen weisen dabei eine breitere Anwendungsvielfalt auf, da sie auch geeignet sind, Nachweise gegenüber dritten zu führen, ohne auf zentralistische Lösungen zurückzugreifen. Mit Hilfe von verschlüsselten Prüfsummen kann der Verschlüsselungsaufwand minimiert werden, um eine hohe Performanz zu erreichen.

2.2.4 Sichere Protokolle

Nicht nur die kryptographischen Algorithmen selbst bestimmen die erreichte Sicherheit eines Systems. Sind bei der Verschlüsselung an sich die Schlüssellänge und die Art und Anzahl der Verschlüsselungsoperationen die wesentlichen Kriterien für die Sicherheitsdienstgüte, so ergeben sich in der Anwendung der Verfahren häufig gravierende Mängel, die ihren Ursprung im Zusammenspiel der Komponenten auf beiden Seiten der Kommunikation haben. Ein einfaches Beispiel soll diesen Zusammenhang illustrieren:

Alice und Bob wollen über ein ungesicherte Verbindung vertrauliche Nachrichten austauschen. Sie benutzen hierfür ein asymmetrisches Verfahren. Dies ist der gewünschte Ablauf ihrer Kommunikation:

1. Alice sendet ihren öffentlichen Schlüssel an Bob.

2. Bob sendet seinen öffentlichen Schlüssel an Alice.

3. Beide verschlüsseln ihre Nachrichten mit dem öffentlichen Schlüssel des anderen.

Ein “Man-in-the-Middle” Angriff könnte jedoch folgendes bewirken [Schneier 1996, S. 48]:

1. Alice sendet ihren öffentlichen Schlüssel an Bob. Der Angreifer Mallory fängt diese Nachricht ab und sendet statt dessen einen eigenen Schlüssel an Bob.
2. Bob sendet seinen öffentlichen Schlüssel an Alice. Mallory fängt auch diesen ab und sendet einen eigenen Schlüssel an Alice.
3. Jede folgende Nachricht wird von Mallory abgefangen, entschlüsselt, mit dem richtigen öffentlichen Schlüssel des Empfängers wieder verschlüsselt und dann an den Empfänger weitergeleitet.

Mallory kann also unbemerkt jede Nachricht mitlesen, obwohl der Verschlüsselungsalgorithmus selbst völlig sicher arbeitet. Mallory's Angriff wird jedoch vereitelt, wenn die die Schlüssel zusätzlich signiert werden. Beispielsweise kann ein vertrauenswürdiger Rechner als Keyserver eingesetzt werden. Jeder Schlüssel eines Benutzers wird bei der Vergabe vom Server signiert. Zum Aufbau der Kommunikationsbeziehung werden nun die signierten Schlüssel ausgetauscht, so daß jeder Kommunikationspartner die Zugehörigkeit des empfangenen Schlüssels zum gewünschten Gegenüber überprüfen kann. Mallory kann jedoch keine Schlüssel erzeugen, die die Signatur des Keyservers tragen.

Eine weitere Art von Angriffen besteht im erneuten Senden von abgefangenen Nachrichten zu einem späteren Zeitpunkt. So könnten z.B. korrekte Überweisungsaufträge mehrmals abgeschickt und dementsprechend mehrmals ausgeführt werden. Diesen Angriffen kann man z.B. durch die Übertragung von verschlüsselten Zeitstempeln begegnen. Eine wiederholte Nachricht wird dann auf Grund ihres nicht mehr aktuellen Zeitstempels abgelehnt.

Diese Beispiele demonstrieren, daß zu jeder Anwendung eines Algorithmus, der Sicherheit gewährleisten soll, auch ein entsprechendes Kommunikationsprotokoll gehört, das ebenfalls gegen Angriffe geschützt ist. Gerade bei der Absicherung von Kommunikationsbeziehungen, die über ein unsicheres Netzwerk führen, ist davon auszugehen, daß ein Angreifer jede Nachricht abfangen und beliebige neue Nachrichten generieren kann. Die verwendeten Protokolle müssen gegen diese weitreichenden Möglichkeiten von Angreifern immun sein. Zeitstempel und Signaturen sind wesentliche Bausteine sicherer Protokolle.

2.3 Integration der Verfahren in das Kommunikationssystem

Die im vorigen Abschnitt vorgestellten Verfahren bieten die Möglichkeit, Dienste zur Sicherung von Kommunikationsbeziehungen zu realisieren. Dazu müssen entsprechende Algorithmen und Protokolle bei der Kommunikation verwendet werden. Zur Absicherung einzelner Anwendungen liegt es zunächst nahe, die Sicherheitsmechanismen direkt in der Anwendung zu implementieren, die sichere Kommunikation benötigt. Dies hat den Vorteil, daß nur auf eine einzelne Anwendung abgestimmte Algorithmen

berücksichtigt werden müssen. Anwendungen wie `elnet`, die einzelne Bytes als Nutzdaten über das Netzwerk übertragen, werden z.B. nur den CFB- oder OFB-Modus benutzen. Der einfachen Implementierbarkeit stehen jedoch gravierende Nachteile gegenüber. Da i.d.R. eine Vielzahl an Kommunikationsdiensten eingesetzt wird, muß jeder einzelne davon gesichert werden. Dies beinhaltet die Implementierung der entsprechenden Algorithmen in der Anwendung selbst, womit für unterschiedliche Anwendungen auch wieder verschiedene Algorithmen und Protokolle zum Einsatz kommen. Eine universelle Bibliothek, die eine Vielzahl von Algorithmen anbietet, ist hierbei hilfreich, löst das Problem jedoch nicht. Denn trotzdem müssen für jede dieser Anwendungen die Sicherheitsmechanismen entsprechend den Erfordernissen konfiguriert werden. Dies erhöht die Wahrscheinlichkeit, daß Administrationsfehler zu Sicherheitslücken führen. Verfügt jede Anwendung über eigene Schlüssel, erhöht sich der Aufwand der Schlüsselverwaltung beträchtlich [Stallings 1995, S. 113]. Wenn jedoch mehrere Anwendungen dieselben Schlüssel benutzen, können Protokollinteraktionen zwischen zwei Anwendungen einen Angriff ermöglichen [Schneier 1998]. Solche Interaktionen sind z.B. Replay-Angriffe (i.d.R. eine Interaktion zwischen zwei Instanzen desselben Protokolls) oder viele “Man-in-the-Middle”-Angriffe [Kelsey 1997]. Der Einsatz kryptographischer Protokolle auf Anwendungsebene ist also besonders gefährlich, da jeder rechtmäßige Nutzer des Systems geänderte Versionen der Anwendungen aufrufen kann, welche die Protokolle gezielt angreifen.

Wünschenswert ist also eine Integration der Sicherheitsdienste in das Kommunikationssystem selbst. Dies ermöglicht, die Sicherheitsmechanismen einmal für alle Anwendungen zu implementieren und sie zentral zu konfigurieren. Weiterhin werden die Algorithmen und Protokolle dem direkten Zugriff und der Sichtbarkeit durch Nutzer des Systems weitgehend entzogen, so daß die Sicherheit erhöht wird und die Komplexität der Sicherheitsmechanismen für den Benutzer transparent ist.

2.3.1 Das ISO Referenzmodell für offene Netze

Heutige Kommunikationssysteme bestehen aus einem Stapel von Protokollschichten, wobei eine höhere Schicht jeweils die angebotenen Dienstleistungen der nächst tieferen nutzt, um weitergehende Dienste zu realisieren. Ist die Aufgabe der unteren Schichten nur, die korrekte Übertragung von Bits sicherzustellen, so bewältigen übergeordnete Schichten z.B. die Wegewahl von Datagrammen durch das Netzwerk, die erneute Übertragung im Fehlerfall und das Einhalten der Reihenfolge der Datagramme. Die oberste Schicht stellt den Anwendungen jeweils eine einfache Schnittstelle zur Nutzung von Netzwerkdiensten zur Verfügung. Die Einteilung in Schichten ermöglicht, beim Entwurf und der Implementierung einer Schicht von den benötigten, tieferliegenden Mechanismen zu abstrahieren. Das bekannteste Modell eines Schichtnetzwerkes ist das OSI-Referenzmodell. Es besteht aus einem Protokollstapel von sieben Schichten, die aufbauend auf der ungesicherten Übertragung von Bits in Schicht 1 auch sehr komplexe Aufgaben erfüllen wie z.B. das Konvertieren von rechnerspezifischen Datenformaten. Im folgenden sollen grundlegende Prinzipien und Bezeichnungen des OSI-Modells kurz eingeführt werden.

Von einer Anwendung ausgehend werden Nutzdaten jeweils bis zur Schicht 1 nach unten weitergereicht, um übertragen zu werden. Die Nutzdaten, die die Schicht n einer

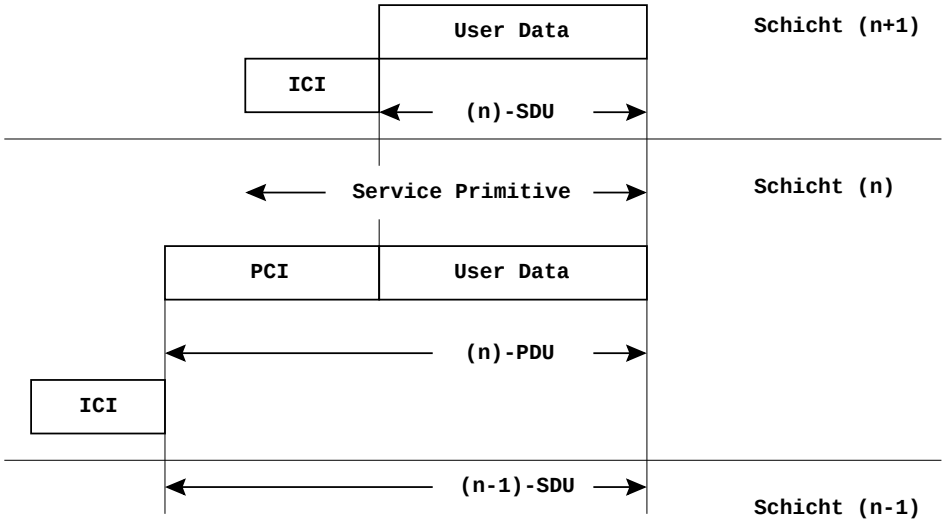


Abbildung 2.4: Das Weiterreichen von Dateneinheiten zwischen Folgeinstanzen [Kerner 1992].

Partnerinstanz eines entfernten Rechners senden will, werden um einen Steuerungsteil ergänzt, der Anweisungen für die empfangende Partnerinstanz enthält, die sog. PCI (*Protocol Control Information*). Ein Beispiel für die Protokollinformationen der Transportschicht sind Steuerinformationen, die eine Flußkontrolle realisieren. Somit kann eine Partnerinstanz angewiesen werden, langsamer zu senden, wenn eine Instanz mit dem Empfang von Paketen überlastet ist. Das so gebildete Datenpaket wird PDU (*Protocol Data Unit*) genannt. Da es i.d.R. keine direkte Verbindung zur Partnerinstanz gibt (nur Schicht 1 kann auf ein Übertragungsmedium zugreifen), wird die PDU um einen Kommandoteil, die sog. ICI (Interface Control Information) ergänzt und an die nächst tiefere Schicht $n-1$ weitergereicht. Der ICI-Teil eines Dienstelements beinhaltet zusätzlich zu Kommandos an die Folgeinstanz auch Informationen, die zur Bearbeitung des Auftrags nötig sind, z.B. die Anzahl der Bytes der übergebenen PDU [Tanenbaum 1990, S. 26]. Das gesamte weitergeleitete Paket aus SDU, PCI und ICI wird Dienstelement (*service primitive*) genannt. Die Schicht $n-1$ entfernt den ICI-Teil, wertet diesen aus und behandelt die PDU der Schicht n ((n)-PDU) als zu übertragende Nutzdaten. Die PDU der Schicht n wird also als SDU (Service Data Unit) der Schicht $n-1$ behandelt. Man bezeichnet die Übertragung der (n)-PDU durch die Schicht $n-1$ als transparent, da keine der darin enthaltenen Informationen von der $(n-1)$-Instanz ausgewertet werden [Kerner 1992, S. 47]. Abbildung 2.4 veranschaulicht den Vorgang des Weiterreichens zwischen Folgeinstanzen.

Die Übergabe von Dienstelementen zwischen Folgeschichten erfolgt über einen Dienstzugangspunkt, den SAP (*Service Access Point*). Dies ist die Schnittstelle zwischen zwei Instanzen von Folgeschichten. Eine mögliche Implementierung eines SAP wäre ein Speicherbereich, auf den beide Folgeinstanzen zugreifen können. In den oberen Schichten des OSI-Modells gibt es i.d.R. mehrere Instanzen einer Schicht (z.B. eine für jede kommunizierende Anwendung), so daß es zu Verzweigungen von einer Schicht n zur übergeordneten Schicht $n+1$ kommt. Auch in die andere Richtung kann es Verzweigungen geben, wenn eine Schicht $n+1$ die Dienste verschiedener (n)-Instanzen

beansprucht. Ein Beispiel wäre eine Anwendung, die mehrere Darstellungsinstanzen benötigt [Kerner 1992, S. 50]. SAPs müssen folglich eine eindeutige Kennzeichnung besitzen, damit eine Folgeschicht in der Lage ist, Dateneinheiten an die richtige Instanz weiterzureichen. SAPs ermöglichen somit die Adressierung verschiedener Instanzen einzelner Schichten.

Ist eine Dateneinheit beim sendenden System auf diese Weise bis zu Schicht 1 heruntergereicht worden, wird sie über das Medium übertragen. Auf dem entfernten Rechner wird die Dateneinheit unter Umkehrung des beschriebenen Vorganges wieder nach oben durch die Schichten durchgereicht. Die Schicht n erhält also eine (n)-PDU. Sie entfernt den PCI-Teil, wertet diesen aus und reicht die verbleibende (n)-SDU weiter an die Schicht $n+1$.

Auf dem Weg nach unten durch den Protokollstapel werden den Nutzdaten also Protokollinformationen jeder einzelnen Schicht zugefügt. Dies geschieht durch Voranstellen (Header) oder Anhängen (Trailer) der PCI-Informationen an die Nutzdaten (SDU). Auf dem Weg nach oben entfernt jede Schicht die entsprechenden zugefügten Protokollinformationen; am Ende bleiben nur noch die Nutzdaten der Anwendung übrig.

Über das Medium empfangene Dateneinheiten werden zunächst so lange nach oben weitergereicht, bis eine Wegewahlentscheidung durchgeführt werden kann. Dies geschieht im OSI-Modell in der Vermittlungsschicht. Ist das Paket an das empfangende System adressiert, wird es an die Transportschicht weitergereicht. Wenn es jedoch an einen anderen Rechner weitergeleitet werden muß, wird es wieder an die unteren Schichten übergeben, bis es auf dem benötigten physikalischen Medium weitergesendet werden kann. Dieser Vorgang wird in Abbildung 2.5 gezeigt.

Da das PCI-Feld einer PDU der Schicht n nur von der entsprechenden Schicht n eines entfernten Rechners interpretiert wird, kommunizieren diese Schichten über ein festgelegtes Protokoll direkt miteinander. Alle tieferliegenden Schichten sind also bis auf die Schnittstelle zur Folgeinstanz $n-1$ von Schicht n aus überhaupt nicht sichtbar, da sie die Inhalte der (n)-PDU nicht verändern.[9] Somit können einzelne Protokolle unabhängig von den verwendeten Protokollen der anderen Schichten entwickelt werden, solange die Schnittstellen zur jeweils über- und untergeordneten Schicht feststehen. Auch kann innerhalb eines Protokollstapels ein einzelnes Protokoll gegen ein anderes ausgetauscht werden, wenn es eine gleiche Funktionalität bietet und die Schnittstellen identisch sind. Auf diese Weise läßt sich z.B. das Übertragungsmedium auswechseln, indem die untersten Protokollschichten durch an das neue Medium angepaßte Pendants ersetzt werden. Die höheren Schichten bleiben von dieser Veränderung unberührt.

2.3.2 Probleme der Integration von Sicherheitsdiensten in das OSI-Modell

Den Vorteilen des Schichtenentwurfs von Netzwerken stehen normalerweise nur geringe Nachteile gegenüber, wie z.B. der Overhead, der durch jede Schicht entsteht. Bei der Integration von kryptographischen Verfahren in ein solches System entstehen jedoch

[9] Dies gilt natürlich nicht im Fehlerfall. Kann eine tieferliegende Schicht ihren Dienst nicht erbringen, so werden die darauf aufsetzenden Dienste höherer Schichten i.d.R. auch nicht mehr erbracht werden können.

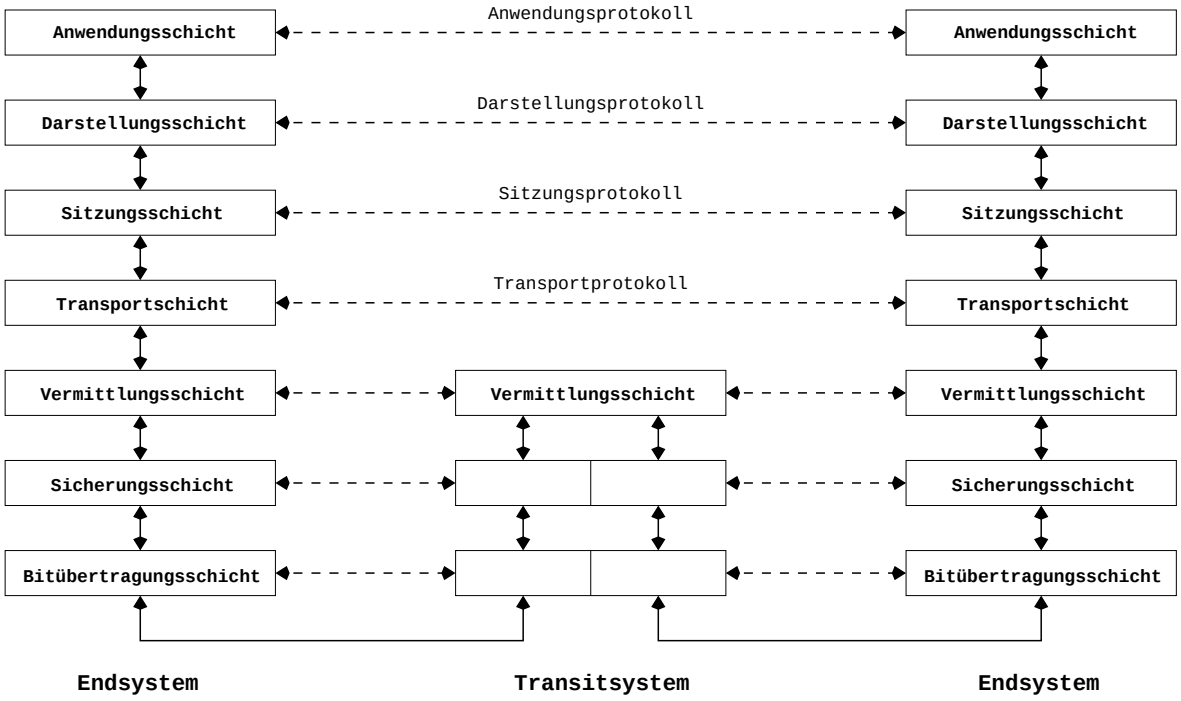


Abbildung 2.5: Kommunikation zwischen OSI Systemen [Kerner 1992].

durch die Einteilung in Schichten einige Schwierigkeiten. Um Sicherheit in Anwendungen zu integrieren, die auf einem Schichtnetzwerk basieren, muß die richtige Schicht gefunden werden, in der kryptographische Verfahren eingesetzt werden [McCurley 1996]. Soll z.B. eine Anwendung Banktransaktionen über ein unsicheres Medium durchführen, so müssen alle Datagramme auf dem Weg vom Anwendungssystem bis zum Bankserver gesichert übertragen werden. In einem OSI-Netzwerk muß also die Verschlüsselung der Daten in einer Schicht durchgeführt werden, die oberhalb der Vermittlungsschicht liegt, da nur dort eine direkte Kommunikation zwischen den Endsystemen vorliegt. Diese Problematik wird im Abschnitt 2.3.3 behandelt. In die Auswahl der Schicht fließt auch die Art der zu gewährleistenden Sicherheit mit ein. In der Vermittlungsschicht werden Datagramme an Endsysteme weitergeleitet. Eine Authentisierung kann sich in dieser Schicht also nur auf einzelne Rechner beziehen, da die Vermittlungsschicht über keine Informationen über Benutzer und Prozesse verfügt. Dies kann jedoch in Umgebungen, in denen sich mehrere Anwender einen Rechner teilen, nicht genügen. Hierfür muß offensichtlich eine höhere Schicht gewählt werden. Das Verwenden von Informationen aus höheren Schichten stellt eine Verletzung der Schichtstruktur dar [Caronni 1996], wodurch die Vorteile der Unabhängigkeit der Protokollschichten aufgegeben werden.

Weiterhin beinhalten die verwendeten Protokolle Annahmen darüber, in welcher Weise verschiedene Systeme interagieren. Solche Interaktionen können dem Einsatz kryptographischer Algorithmen entgegenstehen. Ein Beispiel sind Routingprotokolle, die auf sog. Distanzvektoren beruhen. Hierbei werden Distanzinformationen von einem Router zum nächsten gesendet, von diesem um neue Informationen ergänzt und erneut weitergereicht. Da die Informationen jedesmal verändert werden, sind elektronische

Unterschriften von begrenztem Nutzen. Die Router sind auf die von ihren Nachbarn gelieferten Daten angewiesen und verfügen nicht über die Ursprungsdaten von entfernten Geräten [McCurley 1996].

Zur Integration von Sicherheitsdiensten in ein Kommunikationssystem ist also zunächst einmal die Auswahl der Schicht nötig, in der die gewünschten Dienste erbracht werden können. Zusätzlich müssen die eingesetzten Verfahren in die Protokolle dieser Schicht integrierbar sein. Im folgenden Abschnitt werden diese beiden Aspekte für jede der in Frage kommenden Schichten diskutiert und ein Überblick über bereits bestehende Lösungen gegeben. Vorher soll jedoch zunächst eine wesentliche Kategorisierung der möglichen Lösungen gegeben werden.

2.3.3 "End to End"- und "Link by Link"-orientierte Verfahren

Es gibt zwei grundsätzliche Herangehensweisen, um sichere Kommunikation zu gewährleisten. Da die Kommunikation zwischen zwei Endsystemen i.d.R. über mehrere Transitsysteme (Netzwerkknoten) geschieht, kann versucht werden, jeden Abschnitt der Gesamtstrecke einzeln abzusichern. Dies wird als "Link by Link"-Sicherheit bezeichnet. Ein anderer Weg ist, Sicherheitsmechanismen nur auf Sender- und Empfängerseite zu integrieren, so daß die Transitsysteme nur für das Weiterleiten der Dateneinheiten benötigt werden, an der Sicherung der Kommunikation jedoch nicht beteiligt sind. Dieser Ansatz wird häufig "End to End"-Sicherheit genannt. Den Eigenschaften der Schichten des OSI Modells entsprechend sind Implementierungen von "Link by Link"-Verfahren in den untersten beiden Schichten angesiedelt, "End to End"-Verfahren können frühestens ab Schicht 3 eingesetzt werden, da erst ab Schicht 3 eine allgemeine Adressierung von Endsystemen möglich ist.

Die Entscheidung zwischen beiden Herangehensweisen ist ein generelles Problem beim Entwurf von Software. Das wesentliche Argument für eine Implementierung in einer möglichst hohen Schicht bzw. in der Anwendung selbst ist, daß nur die Anwendung selbst ermessen kann, welcher Dienstgüte die von ihr benutzten Dienste genügen müssen [Saltzer 1984]. Die tieferliegenden Schichten des Kommunikationssystems haben hingegen keine direkte Kommunikationsmöglichkeit mit der Anwendung, so daß sie nur für alle Anwendungen identische Dienste erbringen können. Für eine Implementierung von Diensten in unteren Schichten spricht jedoch oftmals die zu erreichende Performanz und die Einfachheit der Implementierung. Die unterschiedlichen Eigenschaften beider Verfahren in Bezug auf Sicherheitsdienste sollen im folgenden vorgestellt werden.

Link by Link

Verfahren, die die Kommunikation zwischen jeweils benachbarten Netzwerkknoten sichern, werden als "Link by Link"-Verfahren bezeichnet. Das Netzwerk wird hier also als eine Menge von Knoten betrachtet, die durch Kommunikationskanäle verbunden sind, von denen jeder einzelne unabhängig von den anderen abgesichert werden kann. Jedes Paar von direkten Nachbarknoten verfügt dabei über einen gemeinsamen Schlüssel, mit dem die Daten des entsprechenden Kommunikationskanals ver- und entschlüsselt

werden können. Die insgesamt erreichte Sicherheit ist damit bestimmt durch die Sicherheit jedes einzelnen Knotens. Implementierungen von "Link by Link"-Mechanismen sind i.d.R. in Schicht 1 oder Schicht 2 angesiedelt. Insbesondere die Bitübertragungsschicht bietet sich an, um Ver- und Entschlüsselung von Daten durchzuführen, da die Schnittstelle zu Schicht 1 standardisiert ist.[10] Es ist leicht, an dieser Stelle Geräte (sog. "Krypto-Boxen") anzuschließen, die die Ver- und Entschlüsselung durchgehender Daten in Hardware vornehmen und dementsprechend einen hohen Datendurchsatz erreichen [McCurley 1996]. Die Ver- und Entschlüsselung ist für alle höheren Schichten transparent, so daß keine weiteren Änderungen notwendig sind.

Ein weiterer Vorteil ist die Möglichkeit, die gesamten Dateneinheiten einschließlich der Header und Trailer zu verschlüsseln. Da verschlüsselte Dateneinheiten bei Erreichen des nächsten Knotens zunächst wieder entschlüsselt werden, bevor sie von höheren Schichten bearbeitet werden, können auch die Header und Trailer mit den Quell- und Zieladressen, Routinginformationen etc. komplett verschlüsselt werden. Somit lassen sich durch Lauschen an der Kommunikationsleitung nicht einmal Informationen über die Struktur oder den Zielort von Dateneinheiten gewinnen. Selbst im Falle von verschlüsselten Headern ist der Einsatz einer Firewall unproblematisch, da diese nur bereits entschlüsselte Dateneinheiten zur Bearbeitung erhält. Sie hat also vollständige Einsicht in die übertragenen Nutzdaten und Kontrollinformationen.

Es ist sogar möglich, über die Datenleitung einen zufällig generierten Bitstrom zu senden, auch wenn gerade keine Nutzdaten übertragen werden. Das Entschlüsselungsgerät auf der Empfängerseite muß nach der Entschlüsselung erkennen können, ob es sich um Nutzdaten oder um zufällige Dateneinheiten handelt, die nur generiert wurden, um eine Verkehrsflußanalyse zu verhindern. In diesem Fall kann ein Mitschneiden der Kommunikation nicht einmal Informationen über Paketlängen oder Übertragungshäufigkeiten erbringen. Diese hohe Sicherheit der Kommunikationskanäle gegenüber Angriffen wird jedoch durch die erreichte Gesamtsicherheit bei "Link by Link"-Verfahren relativiert. Die Dateneinheiten werden von allen höheren Schichten in unverschlüsselter Form verarbeitet, folglich muß jeder durchlaufene Netzwerkknoten sicher sein. Es gilt auch zu gewährleisten, daß verschiedene, einen Knoten durchlaufende Assoziationen auch im Knoten voneinander isoliert bearbeitet werden. Insbesondere in Umgebungen, die ein dynamisches Routing ermöglichen, wie z.B. im Internet, kann ein einziger kompromittierter Netzwerkknoten weitere Verbindungen zu sich umleiten, so daß die Sicherheit des gesamten Netzes beeinträchtigt wird [Voydock 1983].

Ein Vorteil von "Link by Link"-Verfahren ist die Flexibilität der Verschlüsselung. Geht man von einem sicheren internen Netzwerk aus, dann ist es möglich, Verschlüsselung nur auf externen Übertragungsstrecken einzusetzen. Es können auch unterschiedliche Verschlüsselungsverfahren für die verschiedenen Medien benutzt werden. Ein Satellitenkanal kann z.B. mit einem Algorithmus verschlüsselt werden, der höheren Sicherheitsanforderungen genügt, während bei einer Kommunikation über eine Glasfaserleitung ein niedrigeres Sicherheitsniveau ausreichend sein kann.

[10] Die ICI-Felder des OSI-Modells sind nicht genormt; jeder Hersteller kann seine eigenen Schnittstellen zwischen einzelnen Schichten entwickeln [Kerner 1992, S. 48]. Somit gibt es keine standardisierten Dienstschnittstellen zwischen Folgeschichten. Da i.d.R. die physikalischen Medien und die entsprechenden Schnittstellenkarten weltweiten Standards entsprechen, bietet sich hier die Möglichkeit, herstellerunabhängige Sicherheitslösungen einzusetzen.

Des weiteren führt die einfache Umsetzbarkeit in einer Hardware-Implementation zu einer Entlastung der Endsysteme, was insbesondere in Anbetracht des hohen Rechenzeitbedarfs von kryptographischen Verfahren von Vorteil ist. Auch die Verwaltung der Schlüssel der jeweiligen Netzwerkknoten ist relativ einfach, da zwei benachbarte Knoten jeweils ihren Schlüssel unabhängig vom Rest des Netzwerkes ändern können [Schneier 1996, S. 217].

Problematisch sind bei den "Link by Link"-Verfahren die Kosten, um die Sicherheit zu gewährleisten. Jeder Netzwerkknoten muß vor physikalischem Zugriff geschützt sein und korrekt administriert werden. Die Schlüssel müssen relativ häufig gewechselt werden, was in einem großen Netzwerk aufwendig sein kann. Ein sehr gewichtiges Argument gegen den Einsatz von "Link by Link"-Verfahren ist, daß die Benutzer der Endsysteme sich darauf verlassen müssen, daß alle Netzwerkknoten sicher sind, sie selbst haben keinen Einfluß und keine Kontrolle in Bezug auf die Sicherheit [Voydock 1983]. Weitere Vor- und Nachteile sind in den folgenden Abschnitten zu den entsprechenden Schichten aufgeführt.

End to End

Eine andere Sichtweise liegt der "End to End"-Sicherheit zu Grunde. Hier wird ein Netzwerk als Medium zum Transport von Dateneinheiten zwischen Sender und Empfänger betrachtet. Sicherheitsmechanismen werden dazu eingesetzt, PDU's auf ihrem Weg durch das Netz zu schützen [Voydock 1983]. Somit wird z.B. Verschlüsselung bereits beim Sender durchgeführt und die Dateneinheit erst beim Empfänger wieder dechiffriert. Eine Implementierung von "End to End"-Verfahren kann also nur in Schichten ≥ 3 erfolgen, da erst die Vermittlungsschicht eine virtuelle Verbindung zwischen Sender und Empfänger realisiert [Stallings 1995, S. 112]. Von der Auswahl der Schicht hängt auch ab, was die Endpunkte einer Datenübertragung sind. Sender und Empfänger können je nach Schicht z.B. Rechner oder Prozesse sein. Die gewährte Sicherheit kann also je nach Implementierung für PDU's auf ihrem Weg zwischen Prozessen oder nur zwischen Rechnern gelten. Werden als Endpunkte Prozesse gewählt, so lassen sich benutzer- und anwendungsabhängige Sicherheitsmechanismen konfigurieren.

Der wesentliche Vorteil der "End to End"-Sicherheit besteht darin, daß die Benutzer sich nicht "blind" auf die Schutzmechanismen der Netzwerke verlassen müssen. Dies trägt der Tatsache Rechnung, daß insbesondere bei Datentransporten in weltweiten Datennetzen oftmals die Betreiber der Transitsysteme unbekannt und deshalb nicht vertrauenswürdig sind. Der erreichte Grad an Sicherheit ist deshalb wesentlich höher, da ein kompromittiertes Transitsystem zwar die Dateneinheiten einbehalten (Denial of Service), sie jedoch nicht entschlüsseln oder unerkannt verändern kann.[11] Eigenschaften wie z.B. Vertraulichkeit oder Authentizität bleiben somit gewahrt. Zudem entspricht diese Art von Sicherheitsmechanismen eher den Erwartungen eines Benutzers bezüglich seiner Sicherheitsanforderungen [Voydock 1983]. Besteht z.B. Bedarf bezüglich der Vertraulichkeit einer Datenübertragung, so ist es eher beruhigend, wenn das eigene System für die geforderte Verschlüsselung zuständig ist.

[11] Dies setzt natürlich sichere Algorithmen und Protokolle voraus.

Nachteilig ist der Rechenaufwand, der bei den “End to End”-Verfahren den Endsystemen aufgebürdet wird. I.d.R. werden die Verschlüsselungsverfahren eher in Software implementiert, so daß es zu starken Performanzeinbußen kommt. Eine Hardware-Lösung benötigt für jedes Endsystem ein entsprechendes Gerät, so daß diese Lösung noch verhältnismäßig teuer ist. Insbesondere erweist sich hier die fehlende Standardisierung der Schnittstellen innerhalb der Kommunikationssysteme als Problem, da sie keine hersteller- und betriebssystemunabhängigen Lösungen gestattet.

Auch das Verschlüsseln der Header ist hier nur bedingt möglich. Werden die Sicherheitsdienste in Schicht n implementiert, kann nur die n-SDU verschlüsselt werden [Parker 1990]. Der PCI-Teil wird von der Partnerinstanz benötigt, um z.B. die Herkunft der Dateneinheit festzustellen (und dementsprechend den richtigen Schlüssel zum Dechiffrieren auszuwählen). Die weiteren Kontrollinformationen, die von tieferliegenden Schichten angefügt werden, entziehen sich ohnehin dem Zugriff der Schicht n. Entsprechend schwierig ist es auch, eine Verkehrsflußanalyse zu verhindern [Schneier 1996, S. 219], da die hierfür notwendigen Informationen im unverschlüsselten PCI-Teil der PDU vorliegen.

Eine weitere Schwierigkeit bereiten Firewalls, die für die Absicherung interner Netzwerke unverzichtbar geworden sind. Im Falle der “End to End”-Sicherheit erhält eine zu durchquerende Firewall verschlüsselte Dateneinheiten, deren Inhalt sie nicht überprüfen kann. Dies gilt insbesondere dann, wenn die Sicherheitsdienste in Schicht n implementiert wurden, die Firewall jedoch Protokollinformationen einer höheren Schicht überprüfen soll. Hier sind die entsprechenden PCIs nicht verfügbar, da auch sie in verschlüsselter Form übertragen werden [Benecke 1998, S.81ff].

Auf Grund der höheren, erreichbaren Sicherheit und der Möglichkeit, für Quellen und Ziele (z.B. Rechner oder Prozesse) unterschiedliche Sicherheitsanforderungen zu konfigurieren, ist “End to End”-Sicherheit oft den “Link by Link”-Verfahren vorzuziehen. Der “End to End”-Ansatz bietet Sicherheit auch bei der Verwendung unsicherer Netzwerke, so daß er die heutigen Gegebenheiten eher berücksichtigt. Eine Standardisierung entsprechender Protokolle und Schnittstellen könnte auch die Kosten (sowohl in Hinsicht auf Anschaffung als auch auf Datendurchsatz) reduzieren. Ist das Verhindern von Verkehrsflußanalyse ein Schwerpunkt der eigenen Sicherheitsanforderungen, bietet sich der Einsatz von “Link by Link”-Verfahren an. Der höchste Grad an Sicherheit läßt sich durch eine Kombination beider Verfahren erzielen, allerdings entstehen dabei auch die höchsten Kosten. Tabelle 2.6 stellt die Eigenschaften beider Ansätze noch einmal gegenüber.

2.4 Sicherheitsdienste in den einzelnen Schichten

Die Realisierung von Sicherheitsdiensten in einem Kommunikationssystem erfordert die Auswahl der Schicht, in der diese Dienste implementiert werden sollen. Die grundlegende Entscheidung zwischen “End to End”- oder “Link by Link”-Verfahren gibt bereits vor, ob die Mechanismen in die unteren beiden Schichten oder eher in höhere Schichten integriert werden müssen. Für die konkrete Auswahl einer Schicht ist jedoch eine eingehende Betrachtung der mit ihr verbundenen Eigenschaften und Schwierigkeiten der

Eigenschaft:	Link by Link	End to End
Aufwand der Endsysteme	gering	hoch
Performanz	hoch (Hardware)	gering
Aufwand bei sicheren Kanälen	null	gleich hoch
Sichere Transitsysteme	notwendig	nicht notwendig
Firewalls	unproblematisch	schwierig
Transparenz	hoch	gering
Schutz gegen Verkehrsanalyse	möglich	schwierig
Verschlüsselung von Headern	ja	nein

Abbildung 2.6: Gegenüberstellung der Eigenschaften von “End to End”- und “Link by Link”-Sicherheit

Implementierung nötig. Die folgenden Abschnitte stellen für einzelne Schichten jeweils deren Besonderheiten in Bezug auf die Sicherheitsdienste dar.

2.4.1 Bitübertragungsschicht

Der von der Bitübertragungsschicht angebotene Dienst besteht nur darin, ungesicherte Verbindungen zwischen Systemen für die Übertragung von Bits zur Verfügung zu stellen [Kerner 1992, S. 33]. Auf dieser niedrigen semantischen Ebene können nur wenige Sicherheitsmechanismen realisiert werden. Eine Authentisierung ist z.B. nicht möglich, da in Schicht 1 keine Absender- und Empfängerinformationen bestehen. In der CCITT Sicherheitsarchitektur für OSI Netzwerke sind hier nur Dienste vorgesehen, die eine Vertraulichkeit der Daten und des Datenflusses sicherstellen [CCITT 1991]. Es kann auf diese Weise nur erreicht werden, daß ein unberechtigtes Lauschen auf der Leitung keine Informationen preisgibt.

Eine direkte Kommunikationsleitung zwischen zwei einzelnen Netzwerkknoten läßt sich sehr einfach durch Mechanismen in der Bitübertragungsschicht sichern. Hierbei kann jedes einzelne zu übertragende Bit vom Sender durch eine Stromchiffre verschlüsselt werden. Die Verschlüsselung wird also durch eine XOR-Verknüpfung der zu übertragenden Bits mit der Ausgabe eines Pseudozufallszahlengenerators durchgeführt, auf der Empfängerseite werden die Bits wieder entschlüsselt. Die Schlüsselstromgeneratoren auf beiden Seiten müssen hierzu synchronisiert werden, damit ein einzelnes Bit auf beiden Seiten jeweils mit demselben Wert verknüpft wird. Stehen keine Daten zur Übertragung an, so können beliebige Bits verschlüsselt übertragen werden, wenn sie auf der Empfängerseite nach der Entschlüsselung von Nutzdaten unterschieden werden können. Dies verhindert eine Verkehrsflußanalyse durch Abhören der Leitung vollständig.

Lösungen, die sich nur auf die Bitübertragungsschicht beziehen, sind in der Praxis nicht immer einsetzbar, da ihre einfache Implementierung stark vom benutzten Medium abhängt. Beispielsweise finden diese Verfahren auf Medien mit wahlfreiem Zugriff kaum Verwendung, da die Synchronisation mehrerer Schlüsselstromgeneratoren spätestens bei Kollisionen auf dem Medium zu aufwendig wäre.

Präambel	Zieladresse	Quelladresse	Daten-länge	Daten	Prüfsumme
8 Byte	6 Byte	6 Byte	2 B.	46 - 1500 Byte	4 Byte

Abbildung 2.7: Struktur eines Ethernet Rahmens

2.4.2 Sicherungsschicht

Die Sicherungsschicht realisiert die korrekte Übertragung von Bits über eine physikalische Verbindung. Hierzu werden Teile der zu übertragenden Bitfolgen i.d.R. zu Datenblöcken, sog. Rahmen (engl. *frames*) zusammengefaßt und mit entsprechenden Prüfsummen zur Fehlerkorrektur versehen [Kerner 1992, S. 116]. Im Falle der lokalen Netzwerke beinhaltet die Sicherungsschicht zwei Teilschichten, zuunterst die *MAC*-, darüber die *LLC*-Schicht [Kerner 1992, S. 386]. Die MAC-Teilschicht (*Medium Access Control*) regelt den Zugriff der einzelnen Rechner auf ein gemeinsam genutztes Medium. Die LLC-Teilschicht (*Logical Link Control*) stellt auf dem gemeinsam genutzten Medium logische Verbindungen zwischen einzelnen Rechnern oder Rechnergruppen her, indem in den Rahmen auch Quell- und Zieladressen eingefügt werden. Ethernet, Fast Ethernet und Gigabit Ethernet benutzen beispielsweise das in Abbildung 2.7 dargestellte Rahmenformat.

Da die Dienstschnittstellen der Schicht 2 bzw. der LLC- und MAC-Teilschicht vollständig standardisiert sind, ergibt sich hier die Möglichkeit, Sicherheitsmechanismen zu realisieren, die unabhängig von den bereits eingesetzten Geräten und ihrer Software ist. Aus diesem Grund gibt es bereits einige Lösungen, die jedoch nur proprietär sind [King 1990]. Oft bestehen diese aus Verschlüsselungsgeräten, die zwischen die Schnittstellenkarte und den "Transceiver"[12] in die physikalische Leitung eingeschleift werden. Eine weitere Möglichkeit wäre die Integration der Verschlüsselungsgeräte in die Netzwerkkarten selbst. Auf diese Weise könnten die Geräte zusätzlich über den entsprechenden Rechner konfiguriert werden (z.B. Laden neuer Schlüssel).

Einige der wichtigsten Entwurfskriterien für eine Realisierung von Sicherheitsdiensten für lokale Netzwerke sind im folgenden aufgeführt [Poon 1990]:

- Die verwendeten Sicherheitsdienste sollten möglichst für unterschiedliche physikalische Medien verwendbar sein (z.B. COAX-Kabel und Glasfaser).
- Die Rundsendeeigenschaft des Netzwerkes ("Broadcasts") sollte erhalten bleiben. Auch die Erkennung von Kollisionen sollte weiterhin funktionieren.
- Die existierende Hardware sollte weiterhin verwendbar sein (z.B. Transceiver, Hubs oder Switche).

[12] Transceiver verbinden Datenleitungen von einzelnen Rechnern mit einem gemeinsam genutzten Medium. Sie enthalten die Elektronik zur Träger- und Kollisionserkennung auf diesem Medium und senden im Falle einer Kollision ein zusätzliches Ungültigkeitssignal, so daß alle anderen Transceiver die Kollision sicher erkennen [Tanenbaum 1990, S. 170].

- Es sollte möglich sein, sowohl verschlüsselte als auch unverschlüsselte Rahmen zu empfangen und zu senden.

Bei der Übertragung darf die Präambel eines Rahmens nicht verschlüsselt werden, da ihre Bitfolge für die Synchronisation des Senders mit dem Empfänger benötigt wird.[13] Um die Kompatibilität mit vorhandenen Netzwerkkomponenten sicherzustellen, sollten auch die Ziel- und Quelladressen der Rahmen unverschlüsselt übertragen werden, wenn das lokales Netzwerk aus verschiedenen, unabhängigen Segmenten besteht. Hier werden die Empfängeradressen benötigt, um das Zielsegment eines Rahmens zu identifizieren. Die Zieladressen sollten nicht verändert werden, da viele Switche diese Adressen dynamisch ihren Schnittstellen zuordnen und durch zu viele "Adressen" unnötig belastet werden. Die Längenangabe könnten Switche für ihre Speicherverwaltung benutzen, so daß auch diese unverschlüsselt übertragen werden sollte [Benecke 1998]. Die unverschlüsselten Einträge ermöglichen einem Angreifer natürlich eine Verkehrsflußanalyse. Die vollständige Vertraulichkeit des Verkehrsflusses läßt sich also nur in der Bitübertragungsschicht realisieren [Ramaswamy 1990b].

Zusätzlich darf die Verwendung der Sicherheitsdienste nicht zu einer Überschreitung der maximalen Länge eines Rahmens führen. Die in Schicht 2 eingesetzten Verschlüsselungsalgorithmen sollten also längenerhaltend sein. Eine Erweiterung des Rahmenformats um sicherheitsspezifische Einträge ist ebenfalls problematisch. Ein Beispiel für einen solchen Eintrag ergibt sich, wenn sowohl verschlüsselte als auch unverschlüsselte Rahmen übertragen werden sollen. Hier ist ein weiteres Feld im Header nötig, dessen Wert eine entsprechende Unterscheidung für einen empfangenen Rahmen ermöglicht [Poon 1992].

Eine häufige Anwendung von Verschlüsselung in der Sicherungsschicht ist die Verbindung zweier Subnetze über ein ungesichertes, abhörbares Netzwerk. Hier werden die Verschlüsselungsgeräte als "Bridges" an den Übergangsstellen zum unsicheren Netz eingesetzt. Über das unsichere Netz werden die Daten also nur in verschlüsselter Form übertragen. Eine Verschlüsselung in Abhängigkeit von Quell- und Zieladresse erlaubt hierbei, mehreren Subnetzen jeweils unterschiedliche Schlüssel zuzuweisen. Die Kommunikation zwischen zwei Subnetzen kann dann mit individuellen Schlüsseln stattfinden. Zusätzlich bietet sich die Möglichkeit, bei speziellen Empfängeradressen keine Verschlüsselung durchzuführen, z.B. bei Adressen von Rechnern im unsicheren Netz [Benecke 1998].

Die ersten Standards zur Integration von Sicherheitsmechanismen in die Sicherungsschicht des OSI Modells beinhalteten nur Dienste zur Sicherung der Vertraulichkeit der Nutzdaten [CCITT 1991]. Da lokale Netzwerktechnologien jedoch die Verwaltung von Subnetzen und entsprechendes Routing schon innerhalb dieser Schicht implementieren, wurden weitere Dienste für lokale Netze vorgeschlagen [Parker 1990], die in die Standards aufgenommen wurden [CCITT 1996]. Hierzu gehört die Authentisierung von Sender und Empfänger und die Sicherung der Datenintegrität. Eine Realisierung dieser Dienste erzwingt jedoch Änderungen an den Rahmenformaten, was eine schnelle Verbreitung entsprechender Verfahren behindert.

[13] Die Präambel besitzt also eine Funktion für die Bitübertragungsschicht. Eine strikte Trennung zwischen Schicht 1 und Schicht 2 gibt es bei lokalen Netzwerktechnologien i.d.R. nicht, so daß die Präambel den allgemeinen Darstellungen folgend in diesem Abschnitt aufgeführt wird.

2.4.3 Vermittlungsschicht

Die Vermittlungsschicht transportiert Pakete über die Teilstrecken des Netzes von Endsystem zu Endsystem. Sie ist die einzige Schicht, deren Dienste mehr als nur ein Paar von Partnern betreffen [Kerner 1992]. Der Dienst der Vermittlungsschicht besteht also im wesentlichen in der Bestimmung der "Routen", auf denen Pakete weiterzuleiten sind. Die Vermittlungsschicht ist die erste "End to End"-Schicht, da sie die Dateneinheiten zwischen den Endsystemen übermittelt. Mehrere Arten der Vermittlung können dabei benutzt werden:

Durchschaltevermittlung: Beim Aufbau der Verbindung zwischen zwei Endsystemen wird eine Leitung durchgeschaltet, z.B. durch das Herstellen physikalischer Kontakte. Damit entsteht zwischen den Endgeräten eine Punkt-zu-Punkt-Verbindung. Der Aufgaben der Vermittlungsschicht beziehen sich in diesem Fall nur auf den Verbindungsauf- und -abbau. Ein Beispiel für die Durchschaltevermittlung ist das analoge Telefonnetz.

Speichervermittlung: Hierbei werden Pakete in jedem Transitsystem empfangen, gespeichert und auf der entsprechenden Ausgangsleitung weitergesendet. Dieses Verfahren wird im Englischen treffenderweise als *store & forward* bezeichnet. Das bekannteste Beispiel für die Speichervermittlung dürfte das Internet und das zugehörige Internetprotokoll (IP) sein.

Mehrpunktvermittlung: Mehrere Teilnehmer sind am selben Übertragungsmedium angeschlossen. Um Gesprächskollisionen zu vermeiden, muß der Zugriff auf das gemeinsame Medium geregelt werden. Diese Zugriffsregelung und ein begrenztes Routing zwischen Netzsegmenten wird im Falle von lokalen Netzen bereits in der Sicherungsschicht vorgenommen (siehe Abschnitt 2.4.2).

In Bezug auf die Sicherheitsdienste unterscheiden sich Durchschalte- und Speichervermittlung nur unwesentlich. In Durchschaltevermittlungen kann eine Authentisierung einmalig beim Verbindungsaufbau erfolgen, wenn später die hergestellte Route nicht verändert werden kann (sog. "Session Hijacking") und die Integrität der Datenpakete gewährleistet ist. Bei der Speichervermittlung gibt es keinen anderen Weg als jedes Paket einzeln zu authentifizieren.

Die Vermittlungsschicht bietet auf Grund ihrer "End to End"-Eigenschaft die Möglichkeit, eine Authentisierung sowohl des Quellrechners als auch des Zielrechners der Datenpakete vorzunehmen. Weitere Sicherheitsdienste der Vermittlungsschicht können Integrität und Vertraulichkeit der Nutzdaten sein. Eine Vertraulichkeit des Verkehrsflusses läßt sich in Schicht 3 nur bedingt erreichen. I.d.R. kann nur die Länge eines verschlüsselten Paketes verborgen werden, indem vor der Verschlüsselung einige Füllbytes an die Nutzdaten angehängt werden (sog. *Padding*) und deren Anzahl ebenfalls verschlüsselt im PCI-Teil des Paketes vermerkt wird [Kent 1998c]. Weitergehender Schutz der Verkehrsflußdaten läßt sich durch sog. "Onion-Routing" erreichen [Reed 1998]. Hierbei wird ein Netz von speziellen Routern verwendet, die untereinander über statische Verbindungen kommunizieren. Auf dem lokalen Rechner oder einem eingesetztem Proxy wird ein zufälliger Weg über die vorhandenen Router zum

Ziel gewählt. Für jeden auf dem Weg liegenden “Onion-Router” wird die Dateneinheit mit dessen öffentlichen Schlüssel chiffriert, nachdem sie mit der Adresse des folgenden “Onion-Routers” versehen wurde. Die entstehende Dateneinheit wird durch Füllbytes auf eine Standardlänge erweitert und an den ersten “Onion-Router” übergeben [Goldschlag 1999]. Jeder “Onion-Router” entschlüsselt die Dateneinheit einmalig und entnimmt die Adresse des nächsten “Onion-Routers”. Die Dateneinheit wird wieder auf die Standardlänge erweitert und dann weitergeleitet. Die Länge der Dateneinheit bleibt also auf ihrem Weg unverändert. Jeder “Onion-Router” entfernt eine “Zwiebelschale” der Dateneinheit durch die Entschlüsselung, kann jedoch nur die Adresse des nächsten Routers entnehmen. Durch die Entschlüsselung ändert sich die Dateneinheit auf jedem Teilabschnitt des Netzwerkes und kann nicht zurückverfolgt werden. Ein kompromittierter “Onion-Router” kann nur eine “Zwiebelschale” der Dateneinheit entfernen, so daß eine Verkehrsflußanalyse weiterhin unmöglich ist. Mit der Anzahl der eingesetzten “Onion-Router” steigt die gewährte Sicherheit gegen Verkehrsflußanalyse [Goldschlag 1999]. Ein wesentlicher Nachteil dieses Verfahrens ist jedoch der hohe Aufwand der mehrfachen Verschlüsselung vor dem Absenden. Bisherige Implementierungen von “Onion-Routing” sind über Proxies auf der Anwendungsebene erfolgt. Prinzipiell könnte dieses Verfahren jedoch auch in die Vermittlungsschicht integriert werden. “Onion-Routing” arbeitet jedoch nicht mit dynamischer Wegewahl zusammen, da die auf dem Weg liegenden Router vor dem Absenden der Dateneinheiten bekannt sein müssen. Verfahren zur Schlüsselverteilung und zur Anpassung an veränderte Netzwerktopologie wurden noch nicht implementiert [Reed 1998].

Im Normalfalle können die Zieladressen im PCI-Teil der Pakete jedoch nicht nicht verschlüsselt werden, da diese von den Transitsystemen benötigt werden, um ein Routing durchzuführen. Auch die Absenderadresse entzieht sich i.d.R. der Verschlüsselung. Zum einen ist sie für eine von Firewalls durchgeführte Plausibilitätskontrolle unverzichtbar,[14] zum anderen wird sie evtl. für die Auswahl des richtigen Schlüssels zum Dechiffrieren des Paketes benötigt. Weitere Einträge im PCI-Teil der Dateneinheiten von Schicht 3 betreffen die Fragmentierung. Die Sicherungsschicht benutzt auf Grund der verwendeten Netzwerktechnologie eine feste Rahmenlänge. Daher muß die Vermittlungsschicht Pakete größerer Länge zunächst in entsprechend kleinere Pakete zerteilen, um diese weiterzureichen. Dieser Vorgang wird auch Segmentierung oder Fragmentierung genannt. Eine solche Fragmentierung kann auch in einem Transitsystem notwendig sein, wenn die Daten auf einem Medium weitergeleitet werden, das eine kleinere Rahmengröße verwendet. Die Vermittlungsschicht muß solche Fragmente auf der Empfangsseite wieder zur ursprünglichen Dateneinheit zusammensetzen. Bei einer statischen Wegewahl kann dies bereits in einem Transitsystem erfolgen, da alle folgenden Fragmente das Transitsystem passieren müssen. Bei dynamischem Routing werden Fragmente erst im empfangenden Endsystem reassembliert. Ist die Rahmenlänge von Schicht 2 größer als die der Vermittlungsschicht, so kann diese mehrere Pakete der Schicht 3 zu einer SDU der Schicht 2 vereinigen. Auch hier muß das empfangende System diesen Vorgang wieder umkehren. Dieses Verfahren wird als *Blocking* bzw. *Deblocking* bezeichnet. Die entsprechenden Einträge im PCI-Teil der Pakete können

[14] Eine Firewall überprüft i.d.R., ob ein aus einem externen Netz stammendes Paket die Absenderadresse eines Rechners des internen Netz trägt. Oftmals wird versucht, durch Fälschen der Absenderadresse von Paketen zu Rechten zu gelangen, die nur Rechnern des internen Netzes gewährt werden [Morris 1985].

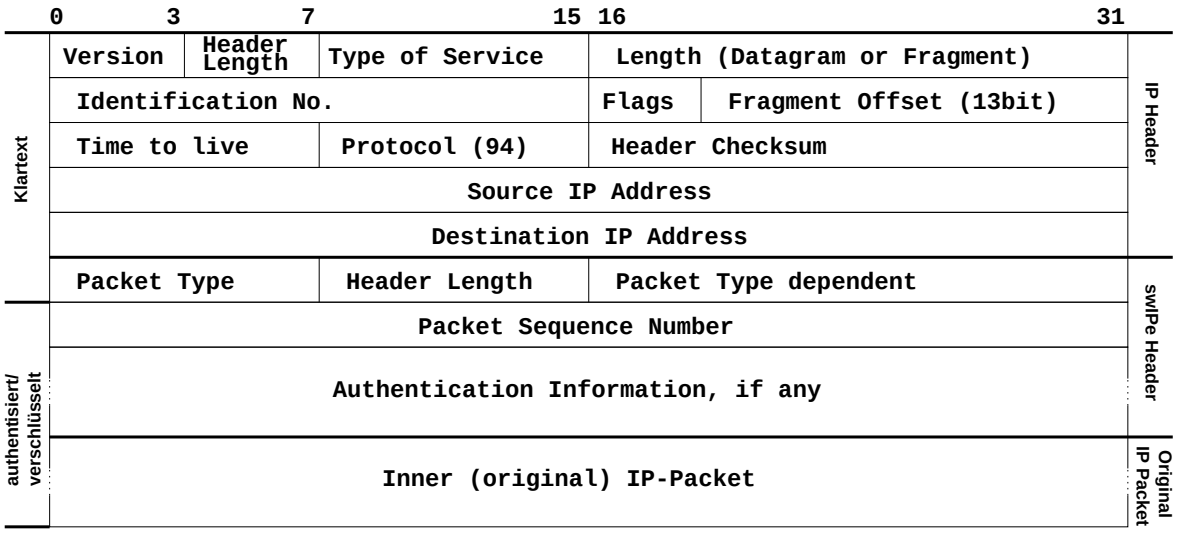


Abbildung 2.8: Das vom swIPe-Protokoll verwendete Paketformat [Ioannides 1993]

natürlich ebenfalls nicht verschlüsselt werden, da sie evtl. auf dem Wege durch das Netz verändert werden müssen.

Es bietet sich also auch in Schicht 3 kaum die Möglichkeit, Felder des PCI-Teils zu verschlüsseln. Zudem sind die PCI-Einträge bei den heute eingesetzten Protokollen genormt, so daß eine Erweiterung der Headereinträge nur durch Einführung neuer Standards und entsprechender Geräte möglich ist. Viele heutige Protokolle behelfen sich damit, eine PDU der Vermittlungsschicht um einen weiteren Kopf zu erweitern, der die benötigten Informationen wie z.B. Signatur und Art der verwendeten Verschlüsselung enthält. Ein solches Paket erhält nun wieder einen PCI-Teil des standardisierten Protokolls der Vermittlungsschicht. Durch diesen PCI-Teil kann das modifizierte Paket dennoch das Netzwerk durchqueren, bis es auf der Empfängerseite wieder entsprechend zerlegt wird. Dieses Verfahren wird als *Tunneling* oder auch *Encapsulation* bezeichnet. Abbildung 2.8 zeigt exemplarisch das Paketformat des *swIPe*-Protokolls [Ioannides 1993]. Der von *IPsec* eingesetzte “Tunnel Mode” ist ein Beispiel dafür, wie sich durch *Tunneling* auch eine weitgehende Vertraulichkeit des Verkehrsflusses erreichen läßt (siehe Abschnitt über *IPsec*). Schwierigkeiten kann beim *Tunneling* die Verlängerung der Pakete durch die zugefügten Header bereiten. Wird dabei die maximal zulässige Paketgröße überschritten, kommt es zur Fragmentierung. Sinnvoll ist hier, in höheren Schichten nur entsprechend kürzere Pakete zu generieren, um größere Performanzeinbußen zu vermeiden.

Die Vermittlungsschicht bietet die einzige Möglichkeit, “End to End”-Verschlüsselung an Zwischenstationen zu beenden [Nelson 1990]. Solche Zwischenstationen können Firewalls sein, die eingehende Pakete entschlüsseln und filtern, aber auch Gateways, die zwischen zwei Subnetzen von einem Verschlüsselungsverfahren auf ein anderes konvertieren. Tiefere Schichten besitzen keine Kenntnis des endgültigen Zielsystems, höhere Schichten kennen die Transitsysteme nicht.

Da die Vermittlungsschicht die Wegewahlentscheidungen trifft, ist sie auch der einzige Ort, um Schutzmechanismen gegen das Verfälschen von Routinginformationen und das daraus folgende Umleiten von Dateneinheiten zu implementieren [Estrin 1991]. Ein entsprechender Angriff muß im Falle von statischer Wegewahl Routingtabellen direkt in den Geräten (Router oder Rechner) verändern, benötigt also zunächst Zugriff auf diese Geräte. Beim dynamischen Routing kann es bereits ausreichen, Pakete mit gefälschten Routinginformationen zum Router zu senden, so daß dieser seine Tabellen automatisch an eine vorgetäuschte Topologie anpaßt [Cheswick 1996, S. 29]. Sicherheitsdienste in der Vermittlungsschicht sind also in jedem System notwendig, um das gewünschte Routing zu gewährleisten.

Die Vermittlungsschicht ist also offensichtlich für das Erbringen bestimmter Dienste die einzig geeignete Schicht. Da man bei den ersten Versuchen der Integration von Sicherheitsdiensten in Protokollschichten zunächst davon ausging, daß es genügen würde, in einer Schicht allumfassende Sicherheitsdienste zu erbringen, war sie das Ziel vieler Bemühungen. Ein weiterer Grund für die Auswahl der Vermittlungsschicht besteht darin, daß viele der bekannten Angriffsarten Schwächen des Schicht-3-Protokolls "IP" (Internetprotokoll) ausnutzen. Daher gibt es bereits einige Ansätze und Implementierungen, um Sicherheitsdienste in diese Protokollschicht zu integrieren und so die Schwächen zu beseitigen. Im folgenden sollen einige davon kurz vorgestellt werden, um zu zeigen, wie solche Lösungen aussehen können.

Beispiel: swIPe

Das Protokoll swIPe ist eine Implementation von Sicherheitsdiensten in der Vermittlungsschicht. Die angebotenen Dienste sichern die Vertraulichkeit und die Integrität der zu übertragenden Nutzdaten, verhindern Replay-Angriffe und ermöglichen die Authentisierung des Absenders eines Pakets.

Wesentliches Entwurfsziel war es, die IP-Funktionalität um Sicherheitsdienste zu erweitern, ohne jedoch die Struktur des Internetprotokolls selbst zu verändern. Um dennoch einige neue Protokollinformationen wie z.B. eine Sequenznummer der Pakete oder Authentisierungsinformation in die Pakete zu integrieren, wird das normale IP-Paket um einen weiteren Protokollheader ergänzt. Weiterhin wird gemäß den Anforderungen das normale IP-Paket und ein Teil des swIPe-spezifischen Headers verschlüsselt. Das auf diese Weise erzeugte Datenpaket wird nun erneut mit einem Standard-IP-Header versehen, der den normalen Transport des Paketes durch das Netz ermöglicht (siehe Abbildung 2.8). Auf der Empfängerseite werden die Informationen im swIPe-spezifischen PCI-Teil ausgewertet, Authentisierung und Entschlüsselung entsprechend ausgeführt und das originale IP-Paket an die normale IP-Implementation weitergereicht. Eine swIPe-Implementation bildet also im Prinzip zwei Teilschichten innerhalb der Vermittlungsschicht, von denen die eine eine Empfangsschicht zwischen der Sicherungsschicht und der IP-Implementation ist, die andere sich als Sendeschicht zwischen der Transportschicht und dem Internetprotokoll befindet (siehe Abbildung 2.9).

Ein swIPe-System besteht prinzipiell aus drei Einheiten; einer Verschlüsselungs-Einheit, einer "Keymanagement"-Einheit und einer "Policy"-Einheit. Die Policy-Einheit entscheidet, ob zu versendende Pakete von swIPe verarbeitet werden müssen, ob eingehende Pakete akzeptiert werden und welche kryptographischen Verfahren auf

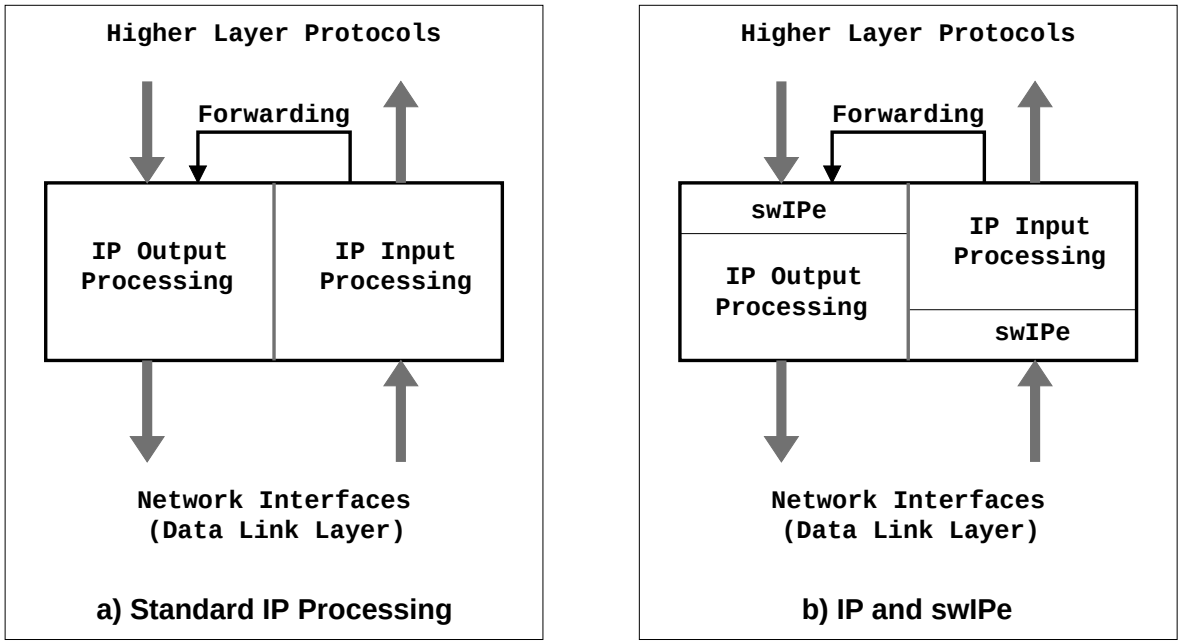


Abbildung 2.9: Die Paketverarbeitung des IP-Protokolls mit und ohne swIPe [Ioannides 1993]

die Pakete angewandt werden müssen. Die Keymanagement-Einheit versorgt die Verschlüsselungs-Einheit mit den notwendigen Schlüsseln und vereinbart mit anderen Keymanagement-Einheiten Session-Schlüssel. Sie ist zuständig für jede Art von Schlüsselaustausch. Die Verschlüsselungs-Einheit wendet die von der Policy-Einheit vorgegebenen Verfahren zur Sicherung von Integrität, Authentizität und Vertraulichkeit auf einzelne Pakete an.

Durch die Kapselung von IP-Paketen in IP-Paketen läßt sich swIPe nicht nur für sichere Verbindungen von Endsystem zu Endsystem einsetzen. Es ist auch möglich, in den äußeren IP-Header die Adresse einer Firewall einzutragen, so daß jedes Paket aus dem externen Netz an einen Rechner hinter der Firewall zunächst an die Firewall selbst gesendet wird. Diese kann die swIPe-Verarbeitung selbst übernehmen und an die Rechner im internen Netz nur noch die bereits entschlüsselten, von der Firewall akzeptierten IP-Pakete weiterleiten. Der Einsatz von swIPe ist dabei für die Rechner im internen Netz transparent. Ebenso kann eine Firewall alle ausgehenden Pakete mit swIPe verarbeiten. Somit bietet sich hier die Möglichkeit an, zwei weit entfernte Subnetze über das Internet zu verbinden, wobei die Firewalls an den Übergangsstellen zum Internet ver- und entschlüsseln können. Die Pakete werden also vor dem Verlassen des lokalen Netzes verschlüsselt und erst von der gegenüberliegenden Firewall wieder entschlüsselt (siehe auch den entsprechenden Absatz in Abschnitt 2.4.2). Auf diese Weise verbundene Netzwerke werden auch als “Virtual Private Networks” (VPN) bezeichnet.

Das swIPe-Protokoll hat in den letzten Jahren aus Mangel an standardisierten Lösungen eine weite Verbreitung gefunden. Es wird jedoch in nächster Zukunft von IPsec, dem neuen Standard der IETF (Internet Engineering Task Force), verdrängt werden. Da die beiden Protokolle nicht kompatibel zueinander sind, wird bereits an entsprechenden

Migrationsverfahren von swIPe zu IPsec gearbeitet [Fujie 1998].

Beispiel: IPsec

IPsec ist der von der IETF spezifizierte Standard für Sicherheitsdienste innerhalb des Internetprotokolls. Die Menge der von IPsec angebotenen Dienste beinhaltet die folgenden:

- Vertraulichkeit von übertragenen Daten
- Authentisierung des Paketursprungs
- Integrität von verbindungslosen Dateneinheiten
- Schutz gegen Angriffe durch wiederholte Pakete ("Replay-Angriffe")
- Begrenzte Vertraulichkeit des Verkehrsflusses
- Zugriffskontrolle

Dies entspricht dem vollem Umfang der von der CCITT vorgeschlagenen Sicherheitsdienste für Schicht 3, soweit sie in einem verbindungslosen Protokoll wie IP möglich sind [CCITT 1991].[15] Die Realisierung dieser Dienste geschieht in Form zweier getrennter Sicherheitsprotokolle, des "IP Authentication Header" (AH) und der "IP Encapsulating Security Payload" (ESP). Hinzu kommen weitere Schlüsselmanagementprotokolle und -funktionen. Die Protokolle AH und ESP erbringen die folgenden Dienste:

Authentication Header: Bietet verbindungslose Integrität, Authentisierung des Paketursprungs und einen optionalen Schutz gegen Paketwiederholungen.

Encapsulating Security Payload: Bietet Vertraulichkeit der Nutzdaten und eine begrenzte Vertraulichkeit des Verkehrsflusses. Weiterhin kann ESP für die Integritätssicherung, die Authentisierung des Paketursprungs und das Ablehnen von wiederholten Paketen eingesetzt werden.

IPsec benötigt für jedes zu bearbeitende Paket einen Sicherheitskontext ("security association", SA), wenn Sicherheitsdienste zur Anwendung kommen sollen. Die SA kennzeichnet eine Simplexverbindung zu einem anderen Rechner. Eine SA besteht aus einer IP-Zieladresse, einer Kennzeichnung des zu verwendenden Protokolls (entweder AH oder ESP) und dem "Security Parameter Index" (SPI). Dieser dient der Unterscheidung mehrerer Verbindungen, deren Zieladresse und verwendetes Protokoll identisch sind. Zwei Datenbanken sind für die korrekte Bearbeitung eines Paketes notwendig. Die "Security Policy Database" (SPD) besteht aus einer geordneten Liste von Policy Regeln. Jede dieser Regeln beinhaltet Auswahlkriterien (sog. Selektoren), die bestimmen, auf

[15] Der Schutz gegen Paketwiederholungen wird in der OSI Sicherheitsarchitektur unter verbindungsloser Datenintegrität behandelt. Weitere Dienste für verbindungsorientierte Protokolle sind Authentisierung der Verbindungsendpunkte und Vertraulichkeit und Integrität von Verbindungen.

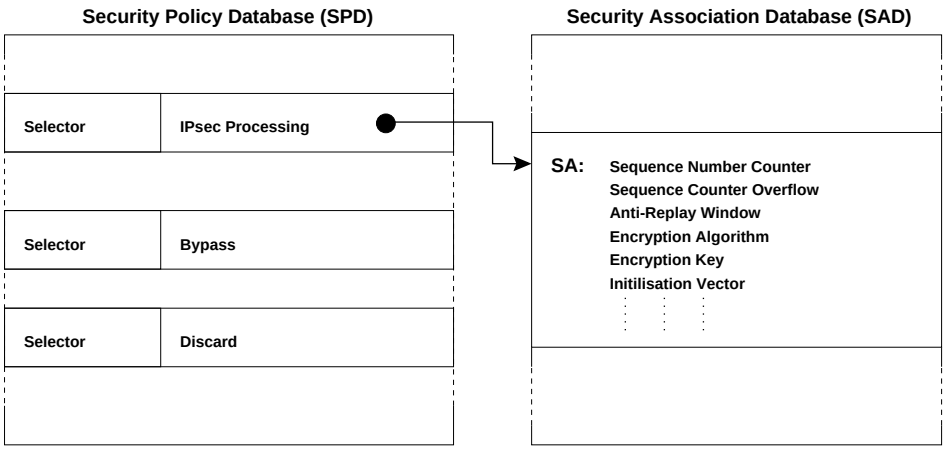


Abbildung 2.10: Die beiden Datenbanken von IPsec-Implementationen

welche Pakete die jeweilige Regel angewandt werden soll. Die SPD enthält getrennte Regeln für ausgehende und eingehende Pakete. Der Policy-Teil einer Regel gibt an, wie die ausgewählten Pakete verarbeitet werden sollen. Mögliche Vorgänge sind das Verwerfen oder Weiterleiten der gewählten Pakete sowie die Verarbeitung durch die IPsec-Funktionen. Für letzteres werden innerhalb der Regel Sicherheitskontexte (SA's) angegeben, die die anzuwendenden Verfahren bestimmen. Eine zweite Datenbank, die "Security Association Database", ordnet jeder SA die entsprechenden Verarbeitungsparameter wie z.B. den zu verwendenden Verschlüsselungsalgorithmus und Schlüssel zu. Abbildung 2.10 zeigt die inhaltliche Struktur und den Zusammenhang der beiden Datenbanken.

Für ein ausgehendes Paket wird also zunächst aus der SPD die betreffende Regel angewendet. Existiert keine passende Regel, wird das Paket verworfen. Wenn die Regel eine IPsec-Verarbeitung vorschreibt, so wird die angegebene SA aus der SAD herausgesucht. Der Eintrag zu dieser SA beinhaltet die anzuwendenden Verfahren und die dazu nötigen Informationen. Existiert keine SA für ein ausgehendes Paket, so wird eine neue angelegt und die entsprechende Regel der SPD um einen Verweis ergänzt. Eingehende Pakete enthalten bereits die Informationen, die als Schlüssel zur Suche in der SAD dienen. Ist in der SAD kein entsprechender Eintrag vorhanden, wird das Paket verworfen. Ansonsten wird es gemäß der gefundenen SA verarbeitet. Ist das Paket vollständig verarbeitet, so wird aus der SPD die entsprechende Policy-Regel herausgesucht. Nun wird verifiziert, ob die angewandte IPsec-Verarbeitung (die durch die Daten im Paket ausgewählt wurde) auch mit der lokal konfigurierten Policy übereinstimmt. Hierzu muß jede passende Policy Regel geprüft werden. Ist die durchgeführte IPsec-Verarbeitung in keiner dieser Regeln enthalten, so wird das Paket verworfen. War die Verarbeitung jedoch konform zur Policy, wird das Paket je nach Zieladresse entweder an die Transportschicht oder an den nächsten Rechner weitergereicht.

Es gibt zwei verschiedene Arten von "Security Associations". Zum einen ist dies der normale *Transport Modus* ("transport mode"), bei dem i.d.R. SDU's der Transportschicht von einem Endsystem zum anderen übertragen werden. Zum anderen ist dies der *Tunnel Modus* ("tunnel mode"), der z.B. im Falle von Sicherheitsgateways zur Anwendung

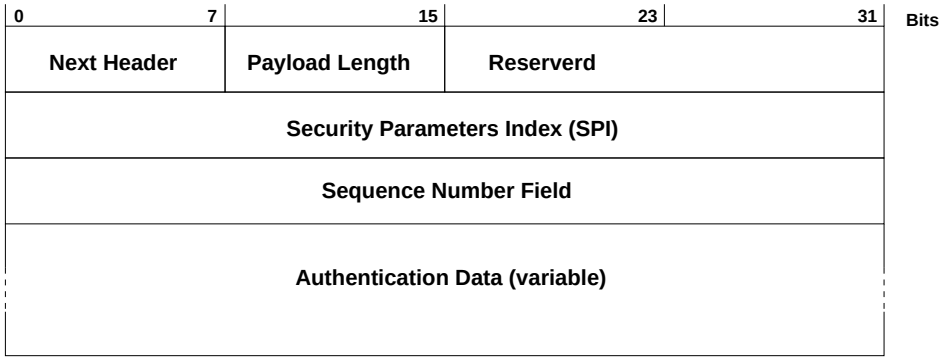


Abbildung 2.11: "IP Authentication Header" [Kent 1998b]

kommt. Hierbei wird das ursprüngliche IP-Paket um Protokollinformationen des AH- oder ESP-Protokolls ergänzt und dann erneut mit einem IP-Header versehen. Dieser Vorgang des Tunnelings ist notwendig, wenn es mehrere Pfade vom Quell- zum Zielsystem gibt, um sicherzustellen, daß alle Pakete dasselbe Sicherheitsgateway durchlaufen. Allerdings kann dieses Verfahren die ursprünglich selbstbestimmte, dynamische Wegewahl eines Transitsystems einschränken, in dem die zu durchquerenden Systeme durch entsprechende äußere Header vorgegeben werden. Dies bedeutet eine Verringerung der durch die dynamische Wegewahl erreichten Ausfallsicherheit. Auch eine Lastverteilung (z.B. auf parallele Firewalls) gestaltet sich schwierig, da die Pakete einer "Security Association" durch denselben Gateway-Rechner bearbeitet werden müssen (da andernfalls z.B. die Sequenznummern nicht geprüft werden können).

In den nächsten Abschnitten sollen die beiden Protokolle "Authentication Header" und "Encapsulating Security Payload" vorgestellt werden. Darauf folgt eine Beschreibung der für IPsec einsetzbaren Protokolle, die zum Erzeugen von "Security Associations" benutzt werden.

Authentication Header: Das "Authentication Header" Protokoll erweitert IP-Pakete um eine Datenstruktur, deren Einträge die Integrität der Daten und die Authentizität des Paketursprungs sicherstellen sollen. Diese Datenstruktur wird als "Authentication Header" bezeichnet. Sie bietet auch Schutz gegen Paketwiederholungen. Der "Authentication Header" besitzt die in Abbildung 2.11 dargestellte Struktur.

Das Feld "Next Header" identifiziert den Typ der auf den AH folgenden Daten. Die Werte sind dieselben, die auch im IP-Header im "Protocol"-Eintrag verwendet werden [Reinolds 1994]. "Payload Length" gibt die Länge des "Authentication Headers" an, der "Security Parameter Index" dient zur Identifizierung der zugehörigen SA wie oben beschrieben. Die Sequenznummer ist ein Zähler, der für jedes Paket der SA monoton erhöht wird. Da er durch die Integritätsprüfung abgesichert ist, können Modifikationen erkannt werden, so daß ein Verhindern von Replay-Angriffen möglich ist. Das Feld "Authentication Data" beinhaltet den Wert der Integritätsprüfung ("Integrity Check Value", ICV). In die Berechnung dieser Prüfsumme gehen beim Routing unveränderliche IP-Header-Informationen, der AH-Header selbst (wobei die "Authentication Data" für die

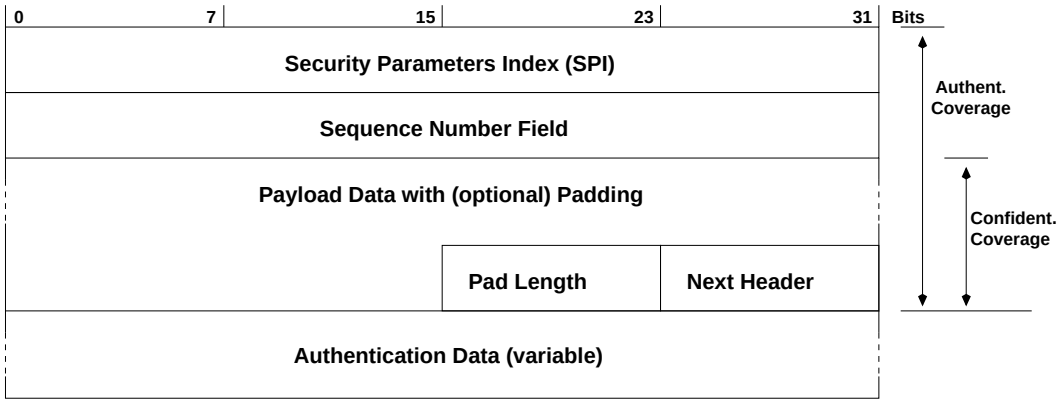


Abbildung 2.12: "IP Encapsulating Security Payload" [Kent 1998c]

Berechnung mit Nullen gefüllt wird) und die Nutzdaten des Pakets ein. Die exakte Beschreibung der Formate und Berechnungen der Einträge befindet sich in [Kent 1998b].

Encapsulating Security Payload: Zur Sicherung der Vertraulichkeit von Datenpaketen wurde das "Encapsulating Security Payload" Protokoll spezifiziert. Abbildung 2.12 zeigt die Struktur des Headers eines solchen ESP-Paketes. Die Nutzdaten können hierbei durch Hinzufügen von Füllbytes auf Blockgrößen abgestimmt werden, um den Einsatz von Blockchiffren zu ermöglichen. Der "Security Parameter Index" ermöglicht die Identifizierung der zugehörigen SA, die Sequenznummer bietet Schutz gegen Replay-Angriffe (wie oben beschrieben). Auch eine Authentisierung des Paketursprungs wird bei ESP ermöglicht, auf eine Verwendung des "Authentication Headers" kann daher verzichtet werden. Die in der "Authentication Data" enthaltene Integritätsprüfsumme wird über das gesamte ESP-Paket (mit Ausnahme der Prüfsumme selbst) berechnet. Bei Verwendung des Tunnel Modus wird der ursprüngliche IP-Header am Beginn der Nutzdaten eingefügt, so daß er gemeinsam mit den Nutzdaten auch verschlüsselt wird. Auf dem Weg zu einem Sicherheitsgateway sind also die endgültigen Empfänger der Pakete nicht ersichtlich, eine gewisse Vertraulichkeit des Verkehrsflusses auf dem Weg durch ein externes Netz ist somit gewährleistet.

IPsec Schlüsselverwaltung: Die IPsec-Spezifikation geht davon aus, daß ein externes Protokoll sowohl die Erzeugung, den Austausch, die Veränderung und das Löschen von "Security Associations" übernimmt. Die hierfür vorgesehene Lösung ist das Protokoll ISAKMP (Internet Security Associations & Key Management Protocol) [Maughan 1998]. ISAKMP spezifiziert nur Paketformate, um Informationen zur Schlüsselgenerierung und Authentisierungsdaten auszutauschen. ISAKMP ist somit unabhängig von den einzusetzenden Verschlüsselungs- und Authentisierungsmechanismen. Diese Aufgaben übernehmen wiederum weitere Protokolle. ISAKMP ist ein Protokoll der Anwendungsebene und bietet seine Dienste allen Netzwerkschichten an. Es

ist also nicht auf IPsec beschränkt, sondern kann z.B. auch für TLS[16] oder OSPF[17] benutzt werden, um eine Vervielfachung von Code in jeder Protokollschicht zu vermeiden [Maughan 1998]. ISAKMP verwaltet dementsprechend auch Informationen über Prozesse, Benutzer oder Portnummern, so daß der Aufbau einer "Security Association" von IPsec auch eine Authentisierung eines Benutzers beinhalten kann. Die Vermittlungsschicht alleine verfügt i.d.R. nur über auf Endsysteme (also Rechner) beschränkte Informationen. Die Verwendung von Informationen aus höheren Schichten ist in IPsec explizit vorgesehen, so daß die Selektoren der SPD ausdrücklich auch auf User-ID und Portnummern Bezug nehmen können. Hier kommt zum Ausdruck, daß IPsec als einzige Sicherheitsschicht gedacht war. Die Probleme, die bei der Integration von Sicherheitsdiensten in den Protokollstapel auftreten (vgl. 2.3.2), wurden durch eine Aufgabe der Schichtstruktur gelöst.

Resümee: IPsec bietet Sicherheitsdienste in der Vermittlungsschicht an. Diese werden mittels sog. "Security Associations" realisiert, welche Informationen zur Verschlüsselung, Authentisierung und Verhinderung von Replay-Angriffen bereitstellen. Diese "Security Associations" können für jedes Teilstück des Weges eines Paketes spezielle Sicherheitsvorgaben beinhalten, so daß im Prinzip die Verbindungslosigkeit des Internetprotokolls aufgehoben wird. Dies ist eine Einschränkung der dynamischen Wegewahl, wodurch die Ausfallsicherheit reduziert und Lastverteilung erschwert wird. Weiterhin werden Informationen aus höheren Schichten benutzt, um festzustellen, ob ein Paket akzeptiert bzw. weitergeleitet werden soll. Hierzu wird ein Protokoll der Anwendungsebene benutzt, daß diese Informationen zur Verfügung stellt. Dies ist eine Umkehrung des Dienstverhältnisses zwischen den Protokollschichten, da im Falle von IPsec die Vermittlungsschicht den Dienst eines Protokolls der Anwendungsschicht in Anspruch nimmt [Benecke 1998]. Die klare Schichtstruktur des Protokollstapels wird damit aufgehoben, was den Entwurf und die Modifikation zukünftiger Protokolle erschwert.

Die "Security Associations" können selbst wieder Gegenstand von Angriffen werden. Eine Manipulation von SA's kann die sichere Kommunikation beeinträchtigen. Beispielsweise könnten gültige Pakete auf der Empfängerseite fehlinterpretiert und verworfen werden [Benecke 1998].

Auch die Vielzahl an verschiedenen Protokollen und die damit verbundene Komplexität bieten Anlaß zu Bedenken bezüglich der erreichten Sicherheit. Insbesondere die sichere Konfiguration der angebotenen Dienste wird dadurch erschwert. Die Paketfilter-Funktionalität, die IPsec mittels der Regeln in der "Security Policy Database" realisiert, besitzt die gleiche Problematik einer gewöhnlichen Firewall: Die Verifizierung, ob mehrere hundert Regeln eine vorgegebene Policy umsetzen, ist schwierig.

Einige Sicherheitsmängel der ersten Entwürfe von IPsec wurden durch Einführung neuer Verfahren behoben. So war z.B. eine Sequenznummer noch nicht Bestandteil der ESP-Spezifikation [Atkinson 1995]. Vorschläge, die entsprechenden Sicherheitsdienste

[16] TLS (Transport Layer Security) bietet Sicherheitsdienste in der Sitzungsschicht an. Eine kurze Beschreibung erfolgt in Abschnitt 2.4.5.

[17] OSPF (Open Shortest Path First) ist ein Internet-Routingprotokoll. Eine Spezifikation von OSPF findet sich in [Moy 1998].

in andere, passendere Schichten zu integrieren [Bellovin 1996], wurden jedoch nicht realisiert. Das Ergebnis ist ein sehr komplexes Gesamtsystem, das die Schichtstruktur des Protokollstapels aufhebt.

Die Tatsache, daß IPsec den einzigen Standard für umfassende Sicherheitsdienste im Internet darstellt, wird IPsec trotz dieser Mängel zu einer weiten Verbreitung verhelfen.

Beispiel: SKIP

SKIP ("Simple Key-Management for Internet Protocols") ist ein weiteres Sicherheitsprotokoll der Vermittlungsschicht. Es ist ein Protokoll, daß ebenso wie ISAKMP geeignet ist, für den Einsatz der "Authentication Header" und "Encapsulating Security Payload" die Verwaltung und Verteilung entsprechender Schlüssel zu übernehmen. Hierzu werden sowohl der AH als auch der ESP mit einem zusätzlichen SKIP-Header versehen. Dieser Header enthält die Informationen, die in IPsec in der SAD verwaltet werden, also z.B. den verwendeten Verschlüsselungalgorithmus und Angaben über den verwendeten Schlüssel. Dies bedeutet, daß SKIP im Gegensatz zum Einsatz von IPsec mit ISAKMP keine Verbindungskontexte benötigt, das Protokoll bleibt also weiterhin verbindunglos. Somit sind auch dynamisches Routing und Lastverteilung ohne Schwierigkeiten einsetzbar [Aziz 1997]. Beim Einsatz von SKIP erhält zunächst jeder Rechner ein Schlüsselpaar eines asymmetrischen Verfahrens. Zwei Rechner, die miteinander kommunizieren wollen, vereinbaren nach dem Diffie-Hellman-Verfahren einen sog. "Master Key". Mit diesem Schlüssel können beliebige Schlüssel, sog. Paketschlüssel, chiffriert und in den SKIP-Header übertragen werden. Die Schlüssel können mit dem Masterkey entschlüsselt werden und zur Entschlüsselung oder Authentisierung des Paketinhaltes benutzt werden. Der Einsatz von häufig wechselnden Paketschlüsseln verhindert, daß der "Master Key" für die Verschlüsselung großer Datenmengen verwendet werden muß. Somit wird eine Kryptoanalyse erschwert und der "Master Key" kann über eine lange Zeitdauer eingesetzt werden. Dies ist in Umgebungen nützlich, in denen der "Master Key" manuell installiert wird, weil kein Schlüsselverteilungszentrum existiert [Caronni 1996].

Der SKIP-Header beinhaltet auch Felder, die eine Authentisierung und Verschlüsselung nicht nur in Abhängigkeit von der Absender- und Empfängeradresse eines Paketes ermöglichen [Aziz 1997]. Es läßt sich z.B. ein Namensraum mit Benutzernamen verwenden (Beispiel: benutzer@domain.net), so daß die Authentizität eines Paketes unabhängig von der Absenderadresse festgestellt werden kann. Dies ermöglicht beispielsweise die Verwendung von dynamischen IP-Adressen, ohne auf die Sicherheitsdienste verzichten zu müssen. Die Art der verwendeten Namensräume ist unabhängig vom SKIP-Protokoll. Werden hierbei Informationen benutzt, die der Vermittlungsschicht normalerweise nicht zur Verfügung stehen, handelt es sich um eine Verletzung der Schichtstruktur des Protokollstapels.

SKIP ist also im Gegensatz zu IPsec weiterhin ein verbindungsloses Protokoll. Der verwendete SKIP-Header bildet einen zusätzlichen Overhead von 28 Byte pro Paket, so daß die höheren Schichten kleinere Dateneinheiten generieren sollten, um eine Fragmentierung innerhalb der Vermittlungsschicht zu vermeiden. SKIP entspricht nur der ursprünglichen Spezifikation von IPsec, da die neue Fassung die Verwendung von Sicherheitskontexten bereits in den Standard integriert hat [Kent 1998a]. Dies bedeutet,

daß SKIP keine weite Verbreitung erreichen wird, obwohl es den Vorteil der Verbindungslosigkeit beibehält.

2.4.4 Transportschicht

Die Transportschicht ist im OSI-Modell der Beauftragte des aus den oberen drei Schichten bestehenden Anwendungssystems für den Nachrichtentransport. Sie vermittelt zwischen den Kommunikationsanforderungen der Anwendungen und den Fähigkeiten des Netzwerkes (z.B. bezüglich Durchsatz, Verzögerungen beim Verbindungsaufbau und bei der Datenübertragung und Restfehlerrate). Ein Dienstelement der Transportschicht enthält also im ICI-Teil sog. Dienstgüteparameter, die auf diese Weise vom Anwendungssystem an das Transportsystem weitergeleitet werden.

Die Transportschicht erhält ihre besondere Bedeutung für die Integration von Sicherheitsdiensten dadurch, daß sie in den heute üblichen TCP/IP-basierten Systemen die oberste, standardisierte Schicht bildet. Sie bietet somit die Gelegenheit, "End to End"-Sicherheit noch in den Protokollstapel, jedoch mit unmittelbarer Nähe zu den Anwendungen zu integrieren. Die Anwendungen können über denselben Mechanismus, über den sie ihre Dienstgüteanforderungen an die Transportschicht weiterreichen, auch Anforderungen bezüglich der benötigten Sicherheitsgüte stellen.

Die Dienste der Transportschicht können verbindungslos oder verbindungsorientiert sein. Der TCP/IP-Protokollstapel beinhaltet aus diesem Grund zwei Protokolle der Transportschicht. Dies sind das verbindungsorientierte *Transmission Control Protocol* (TCP) und das verbindungslose *User Datagram Protocol* (UDP). TCP gewährleistet die korrekte Übertragung von Daten und stellt sicher, daß die Reihenfolge der Pakete beim Absenden auch im empfangenden System erhalten bleibt. Weiterhin realisiert TCP eine Flußkontrolle, um z.B. bei starker Sendeleistung das Empfangssystem nicht zu überlasten. TCP wird beispielsweise von den Diensten *FTP, HTTP* und *RSH* verwendet. UDP bietet hingegen nur eine "best effort" Übertragung von einzelnen Datagrammen an. Es findet i.d.R. dort Verwendung, wo Paketverluste hinnehmbar sind (oder sich diese durch sehr einfache Mechanismen in den Anwendungen selbst beheben lassen) und eine hohe Performanz gefordert ist. Besonders geeignet ist UDP z.B. für sog. "Frage-Antwort-Protokolle", bei denen auf eine Anfrage an einen Server die entsprechende Antwort als Bestätigung interpretiert wird. Die Dienste *NFS* (Network File System) und *NIS* (Network Information System) benutzen UDP für die Datenübertragung.[18] Für die Integration von Sicherheitsdiensten in die Transportschicht bedeutet dies, daß sowohl TCP als auch UDP erweitert werden müssen, um allen Anwendungen Sicherheitsdienste anbieten zu können.

Da UDP zu den Diensten der Vermittlungsschicht nur das Adressieren von Anwendungen hinzufügt, ist auch der UDP-Header sehr einfach. Er enthält nur den Quell- und Ziel-SAP der Transportschicht, die Paketlänge und eine Prüfsumme. Die Information, für welchen Zielrechner das Datagramm bestimmt ist, wird über den ICI-Teil an die Vermittlungsschicht weitergereicht. Sicherheitsdienste könnten im Falle von UDP die Authentizität von Anwendungen (gekennzeichnet durch ihren SAP, die sog. Portnummer) sowie die Integrität und die Vertraulichkeit der Kommunikation zwischen

[18] Neuere Versionen von NFS verwenden mittlerweile das TCP Protokoll.

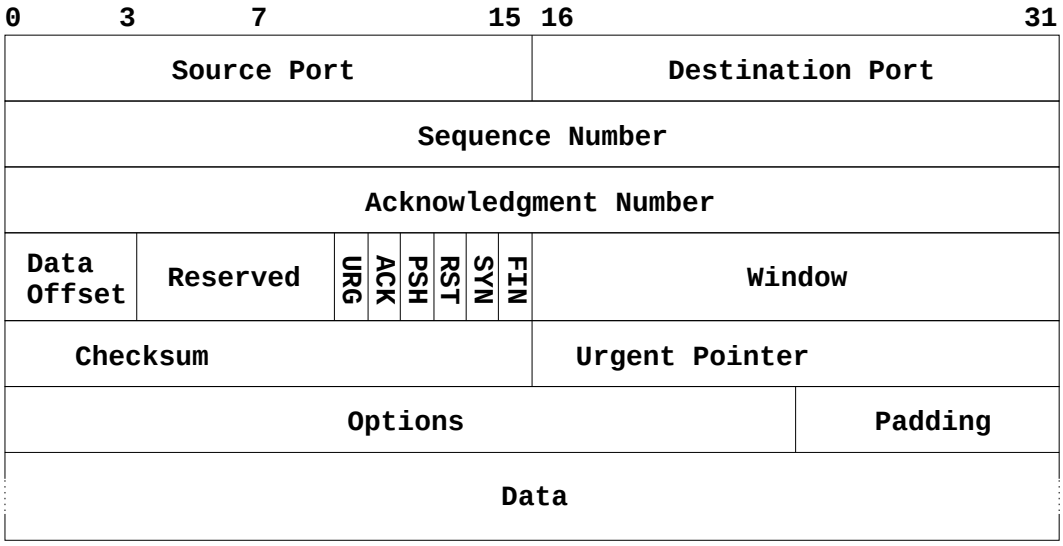


Abbildung 2.13: Struktur eines TCP-Segments [Hunt 1992]

Anwendungsprozessen gewährleisten. Auch hier bietet sich auf Grund des standardisierten Paketformates *Tunneling* an, um die benötigten Informationen zur Authentisierung und Verschlüsselung mit den Dateneinheiten zu übertragen. Die wesentliche Ergänzungsmöglichkeit zu den Sicherheitsdiensten, die eine Vermittlungsschicht anbieten kann, wäre hier also die Berücksichtigung von anwendungsspezifischen Sicherheitsanforderungen.

Der Header von TCP-Segmenten[19] ist wesentlich komplexer als der von UDP-Nachrichten. Abbildung 2.13 zeigt die Struktur eines TCP-Segments. Einige der Einträge des Headers könnten bei TCP verschlüsselt übertragen werden, da sie zunächst beim Empfang der Dateneinheit und der Auswahl des korrekten Schlüssels noch nicht benötigt werden. Da TCP Verbindungen zwischen Endsystemen herstellt, könnte bei Schlüsseln, die jeweils für ein Paar von Rechnern gelten, die gesamte Kontrollinformation verschlüsselt werden. Die Adressen der Endsysteme werden erst im IP-Header eingesetzt, so daß ein korrektes Routing der Segmente zu ihrem Ziel weiterhin möglich ist. Im empfangenden System muß dann durch ein (unverschlüsseltes) Bit signalisiert werden, ob das Segment verschlüsselt oder im Klartext abgesendet wurde. Bei verbindungsspezifischen Schlüsseln werden die Portnummern zur Auswahl des korrekten Schlüssels benötigt, sie müssen also im Klartext übertragen werden. Eine Verschlüsselung der Sequenz- und Bestätigungsnummern setzt voraus, daß die Entschlüsselung stattfindet, bevor die ursprüngliche Sendereihenfolge hergestellt werden kann [Diffie 1985]. Daher kann die Verschlüsselung des Anwendungsdatenstroms nicht in Rückkopplungsmodi wie CBC, CFB oder OFB stattfinden. Alternativ könnte durch Einfügen eines Initialisierungswertes in den Header gewährleistet werden, daß jedes Segment einzeln dechiffrierbar ist. Die Prüfsumme könnte durch eine kryptographisch abgesicherte ersetzt werden, die vorgegebene Größe von 16 Bit ist jedoch nicht ausreichend, um einen Angriff

[19] Die Dateneinheiten werden im Falle von TCP als Segmente (“segments”) bezeichnet, bei UDP werden sie Nachrichten (“messages”) genannt.

zu verhindern.

Da die PCI-Felder der Schicht 4 von keinem Transitsystem benötigt werden, besteht die Möglichkeit, den Header beliebig zu erweitern. Diese Erweiterungen müssen vom sendenden und vom empfangenden System gleichermaßen beherrscht werden. Gegebenenfalls läßt sich die Kompatibilität beider Partnerinstanzen durch Verwendung der reservierten Bits beim Verbindungsaufbau bereits feststellen.[20] Sind auf einer Seite keine Sicherheitsmechanismen vorhanden, wird die Kommunikation je nach Policy entweder unverschlüsselt geführt oder abgebrochen. Wenn beide Partnerinstanzen über die entsprechenden Sicherheitsdienste verfügen, können nach dem Verbindungsaufbau kryptographische Informationen und Nutzdaten nach einem beliebigen Protokoll ausgetauscht werden. Änderungen an den Headerformaten bereiten jedoch Schwierigkeiten mit Firewall-Komponenten, die TCP-Headerinformationen von passierenden Dateneinheiten auswerten. Wird auf Veränderungen im Header verzichtet, so können nach dem Austausch der kryptographischen Informationen beim Verbindungsaufbau verschlüsselte Nutzdaten über das ansonsten unveränderte TCP übertragen werden. Eine Datenflußanalyse läßt sich bei unverschlüsselten Headern jedoch nicht verhindern.

Ein Problem kann im Falle von TCP das sog. “Passive Open” sein. Hierbei gibt ein Serverprozeß nur an, an welchem SAP er auf eingehende Verbindungen warten will. Er spezifiziert jedoch nicht den SAP der Gegenseite. Beliebige Clientprozesse können somit den wartenden Server ansprechen. Geht beim Server ein Verbindungswunsch ein, muß das empfangene Segment zuerst entschlüsselt werden. Werden asymmetrische Algorithmen verwendet, so muß das Segment mit dem öffentlichen Schlüssel des Serverprozesses chiffriert sein. Beim Einsatz von symmetrischen Verfahren muß der Server zunächst den korrekten Schlüssel herausfinden. Dies kann jedoch nicht geschehen, wenn der Clientprozeß seinen Transportschicht-SAP dynamisch zugewiesen bekommt. Die einzige Lösungsmöglichkeit besteht darin, daß der Clientprozeß von seinem lokalen TCP immer denselben SAP zugewiesen bekommt. Auf der Serverseite kann dieser SAP des Clients benutzt werden, um den entsprechenden Schlüssel auszuwählen [Diffie 1985]. Eine andere Möglichkeit wäre, die für den Verbindungsaufbau benötigten Segmente um ein Feld zu erweitern, daß den Absender identifiziert. Diese Lösung ist allerdings nicht kompatibel zum normalen TCP.

Erste Ansätze, Sicherheitsdienste in TCP zu integrieren, sind bereits sehr früh vorgeschlagen worden [Diffie 1985, Voydock 1985]. Einige der bestehenden Implementationen von Sicherheitsdiensten in der Transportschicht sollen in den folgenden Abschnitten vorgestellt werden.

Beispiel: Secure Transport Library

Die in [Jordan 1993] vorgestellte Implementation von Sicherheitsdiensten wurde als Schicht über der eigentlichen Transportschicht konzipiert. Diese verhält sich transparent gegenüber gewöhnlichen TCP- und UDP-Anwendungen. Die Implementierung der “Secure Transport Library” erfolgte jedoch als Bibliothek. Dies bedeutet, daß die Sicher-

[20] Einzelheiten des normalen Verbindungsaufbaus von TCP werden im Abschnitt über *Secure TCP* (Seite 47) beschrieben. Auf das gegenseitige Erkennen von Protokollinstanzen mit Sicherheitsdiensten wird dort ebenfalls eingegangen.

heitsdienste nicht in das Kommunikationssubsystem integriert wurden. Die Verwendung einer Bibliothek zur Realisierung von Sicherheitsdiensten entspringt der Schwierigkeit, Änderungen an bestehenden Betriebssystemen vorzunehmen, wenn der Quellcode nicht verfügbar ist. Die neue Bibliothek bildet jedoch die zu TCP gehörigen Systemaufrufe exakt nach, so daß im Falle eines dynamischen Linkvorganges beim Programmlauf der Binärcode von Programmen nicht verändert werden muß. Ein Umgehen der Sicherheitsdienste durch Verwendung anderer Bibliotheken kann somit jedoch nicht ausgeschlossen werden.

Bei Unix-Systemen kann über den Systemaufruf `setsockopt()` das Verhalten der Transportschicht auf die Bedürfnisse der Anwendung angepaßt werden. Die bestehenden Einstellungen wurden um neue, sicherheitsspezifische Einträge ergänzt. So läßt sich mit der Option `SEC_SECURITY` angeben, ob Sicherheitsdienste überhaupt verwendet werden. Durch `SEC_MYNAME` und `SEC_RMTNAME` lassen sich der eigene Name und der des entfernten Rechners angeben. Diese Namen werden beim Verbindungsaufbau benutzt, um von einem Kerberos Authentisierungsserver einen Sitzungsschlüssel, ein sog. *Ticket*, zu erhalten. Dieser Sitzungsschlüssel wird dann von beiden Kommunikationspartnern für die Verbindung benutzt. Um die Sicherheitsdienste transparent zu gestalten, können in mehreren Dateien Vorgabewerte konfiguriert werden. Eine Anwendung, die die entsprechenden Socket-Optionen nicht spezifiziert, wird eine Verbindung gemäß diesen Sicherheitsvorgaben erhalten. Der Systemaufruf `connect()` führt bei Verwendung der Bibliothek zu einer Anfrage an den Kerberos-Server, der einen Sitzungsschlüssel zurückliefert. Mit diesem Schlüssel wird dann eine TCP-Verbindung aufgebaut. Der Systemaufruf `accept` wartet analog dazu zunächst auf den Erhalt eines Sitzungsschlüssels vom Kerberos-Server. Erst dann folgt der Aufruf der ursprünglichen Funktion `accept`.

Ein weiterer Schwerpunkt der “Secure Transport Library” ist die Absicherung der von UDP angebotenen Gruppenkommunikationsdienste. Dazu wurde das Protokoll des Kerberos Authentisierungsservers um entsprechende Mechanismen erweitert, mit denen Gruppenschlüssel verwaltet werden können. Die “Secure Transport Library” stellt eine Möglichkeit dar, die Verwendung von Sicherheitsdiensten unter Unix-Betriebssystemen zu vereinfachen. Eine weite Verbreitung hat sie jedoch in den letzten Jahren nicht erreicht.

Beispiel: Secure TCP

Eine in das Kommunikationssystem integrierte Lösung stellt *Secure TCP* dar [Tsutsumi 1995]. Hierbei wurde das TCP-Protokoll modifiziert, um die Verwendung von Sicherheitsdiensten zu ermöglichen. Der Aufbau einer Verbindung erfolgt bei TCP durch den sog. “three-way handshake” [Stevens 1994]. Die initiierende Protokollinstanz sendet ein Segment mit gesetztem `SYN`-Bit an die Gegenseite. Diese bestätigt ihre Bereitschaft, die Verbindung anzunehmen durch die Rücksendung eines Segments, bei dem sowohl das `SYN`- als auch das `ACK`-Bit gesetzt sind. Nun wird auch der Erhalt dieses Segments von der initiierenden Seite durch ein ACK-Segment bestätigt. Durch diesen Austausch von Nachrichten wird sichergestellt, daß beide Seiten über die Bereitschaft der Gegenseite informiert wurden. Abbildung 2.14 stellt diesen Vorgang noch einmal graphisch dar.

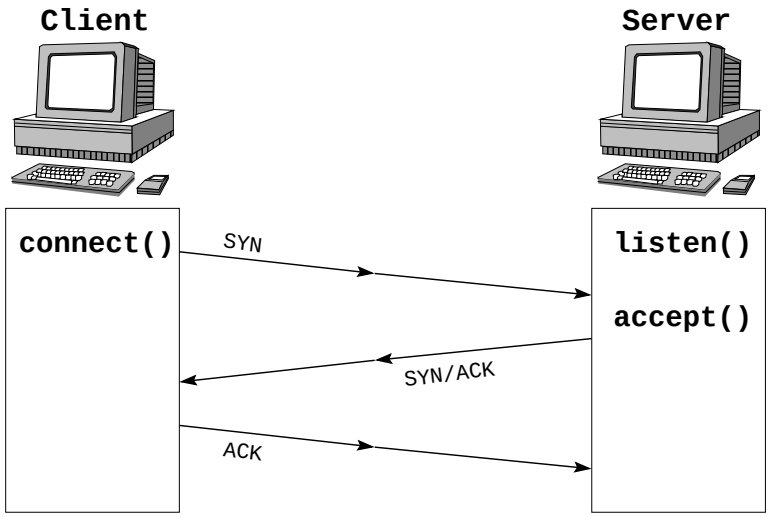


Abbildung 2.14: Der Verbindungsaufbau von TCP durch den “three-way handshake”

Secure TCP erweitert diesen Vorgang des Verbindungsaufbaus auf einen Vier-Wege-Handschlag, bei dem festgestellt werden kann, ob es sich bei der Gegenseite ebenfalls um eine *Secure TCP*-Implementation oder um ein normales TCP handelt. Auch der Austausch von kryptographischen Informationen wird in den Verbindungsaufbau integriert. Die initiierende Protokollinstanz des Clients sendet dabei zunächst ein Segment mit gesetztem SYN-Bit. In diesem Segment wird eine Liste der auf dem Client verfügbaren Verschlüsselungsverfahren übertragen. Der Server sendet ein Segment mit gesetztem SYN- und ACK-Bit zurück. Dieses Segment enthält zusätzlich eine Angabe über den vom Server gewählten Algorithmus und einen entsprechenden öffentlichen Schlüssel des Servers. Hierauf sendet der Client ein ACK-Segment, das einen öffentlichen Schlüssel des Clients enthält. Der Server bestätigt den Empfang dieses Schlüssels mit einem ACK-Segment. Abbildung 2.15 zeigt diese Erweiterung des Verbindungsaufbaus.

Ist auf der Serverseite keine Implementation eines *Secure TCP* vorhanden, so enthält das SYN/ACK-Segment des Servers keine ausgewählte Chiffre und keinen Schlüssel. Die Protokollinstanz des Clients kann daraufhin entweder die Verbindung abbrechen oder eine unverschlüsselte Verbindung aufbauen. Dieser Verbindungsaufbau funktioniert mit einem normalen TCP, da dieses in den Segmenten des Verbindungsaufbaus keine Daten mitsendet. Andere Erweiterungen des TCP-Protokolls, die bereits eigene Daten in diesen Segmenten übertragen, wie z.B. *Transaction TCP* [Braden 1994] können nicht mit *Secure TCP* zusammen eingesetzt werden. Da der normale Header von TCP nur Optionen von bis zu 40 Bytes vorsieht, müssen kryptographische Informationen, die diese Länge überschreiten, im Datenbereich des Segments übertragen werden. Um dies der Gegenseite zu signalisieren wird eines der reservierten Bits benutzt. Ist dieses Bit gesetzt, wird der “Urgent Pointer” benutzt, um den Anfang der übrigen Daten zu kennzeichnen.

Secure TCP spezifiziert keinen Mechanismus, der eine Authentisierung der Kommunikationspartner ermöglicht. Dies soll von einer externen Instanz (z.B. einem Kerberos-Server) übernommen werden. Somit müssen Authentisierung und Verschlüsselung auch

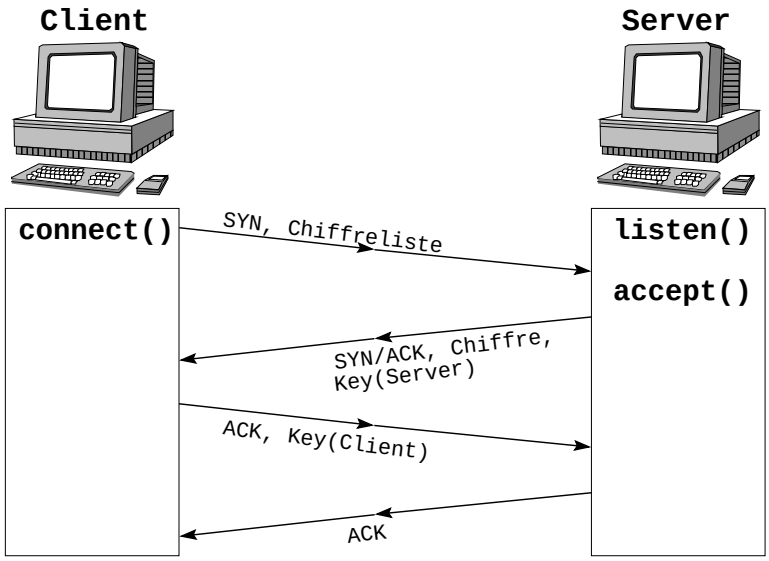


Abbildung 2.15: Der erweiterte Verbindungsaufbau von *Secure TCP*, ein "four-way handshake" [Tsutsumi 1995]

getrennt voneinander konfiguriert werden. Die direkte Verwendung von asymmetrischen Verschlüsselungsverfahren bereitet einige Probleme bezüglich der Performanz. So wird der Durchsatz bei einer Segmentgröße von 1000 Bytes von ca. 6 MBit/s auf etwa 1 MBit/s reduziert [Tsutsumi 1995]. Durch die Beschränkung auf TCP stehen Anwendungen, die UDP verwenden, keine Sicherheitsdienste zur Verfügung. Auch diese Bemühung um Integration von Sicherheitsdiensten in das TCP-Protokoll hat zu keinen weiteren Entwicklungen geführt.

2.4.5 Höhere Schichten

Die Sicherheitsarchitektur für OSI Netzwerke sieht für die Sitzungsschicht keine Sicherheitsdienste vor. Die Darstellungsschicht verfügt über Kenntnisse von einzelnen Datenfeldern, so daß auf dieser hohen semantischen Ebene die Vertraulichkeit von einzelnen Einträgen innerhalb der Nutzdaten gewährleistet werden kann. Auch die Nichtleugbarkeit mit der Beweisbarkeit der Herkunft und der Auslieferung von Dateneinheiten kann hier implementiert werden. In die Darstellungsschicht können weiterhin die meisten der Mechanismen eingebracht werden, die auch in niedrigeren Schichten verwirklicht werden können. Selbst Einzelheiten des Verkehrsflusses können durch Einbringen von zufällig erzeugten Dateneinheiten verschleiert werden. Auch in der Anwendungsschicht können alle diese Sicherheitsdienste angesiedelt werden, so daß beide Schichten hier gemeinsam behandelt werden. Die Tatsache, daß in diesen Schichten alle Sicherheitsdienste mit Ausnahme der sicheren Wegewahl integriert werden können, macht sie zum Gegenstand vieler entsprechender Überlegungen.

Jedoch sind auf heutigen Systemen diese höheren Schichten i.d.R. gar nicht vorhanden. Die TCP/IP-Protokolle finden eine recht gute Entsprechung in den unteren vier Schichten des OSI-Modells. Sie bilden das Transportsystem, welches gewöhnlich vollständig in

den Kern des Betriebssystems integriert ist. Die höheren Schichten, die vom OSI-Modell spezifiziert werden, bilden das sog. Anwendungssystem. Diese wurde jedoch nicht in einer Fortsetzung des TCP/IP Protokollstapels realisiert. Vielmehr werden manche Dienste, die nach dem OSI-Modell von den oberen Schichten angeboten werden, von einzelnen Anwendungen erbracht (z.B. NFS, NIS, FTP oder RSH). Spart diese Vorgehensweise zunächst einmal den Overhead, der durch drei Schichten verursacht wird, bereitet sie jedoch in Bezug auf die Sicherheit gravierende Schwierigkeiten (siehe Abschnitt 2.3).

Als Lösung für diese Probleme bietet sich eine weitere Schicht oberhalb der Transportschicht an, die Sicherheitsdienste für Anwendungen anbietet. Jede Schicht oberhalb des normalen Protokollstapels erzwingt jedoch Änderungen in allen vorhandenen Anwendungen, damit diese Sicherheitsdienste auch benutzt werden.[21] Bisherige Lösungen bestehen daher i.d.R. nur aus einer Bibliothek, die den Anwendungen Sicherheitsdienste anbietet. Somit können weniger sicherheitskritische Anwendungen weiterhin das unveränderte Transportsystem nutzen, während nur die kritischen Anwendungen an die Bibliotheksfunktionen angepaßt werden müssen. Dies bedeutet jedoch auch, daß Sicherheitsdienste nicht vom Systemadministrator vorgeschrieben werden können, da es jedem Benutzer freisteht, ungesicherte Anwendungen einzusetzen. Als Beispiel einer solchen Lösung wird im folgenden Abschnitt das weit verbreitete *TLS* vorgestellt.

Eine weitere Möglichkeit ist das Tunneln von Anwendungsprotokollen. Existiert ein Protokoll, das eine vertrauliche Kommunikation und eine Authentisierung der Kommunikationspartner ermöglicht, so können alle Dateneinheiten von anderen Protokollen mittels dieses einen, sicheren Protokolls übertragen werden. Auf der Empfängerseite übergibt die Protokollinstanz des sicheren Protokolls diese Dateneinheiten wieder an die Anwendungsprotokolle. Ein Beispiel für dieses Vorgehen ist die Verwendung der *Secure Shell* (SSH) [Ylönen 1996]. Durch einen entsprechenden Aufruf von SSH läßt sich ein Tunnel von einem SAP der Transportschicht zu einem anderen erstellen, der dann von der danach gestarteten Anwendung benutzt wird. Die Anwendung muß dabei angewiesen werden, die Verbindung zum *lokalen* Rechner zu öffnen, da der bestehende SSH-Tunnel dann die Übertragung zum Zielrechner übernimmt. Diese Lösung ist also nicht transparent für den Anwender. Weiterhin läßt sich diese Methode nicht für Anwendungen einsetzen, die dynamisch vergebene SAPs benutzen. SSH selbst wurde mit der Möglichkeit versehen, das X11-Protokoll zu tunneln, andere Protokolle, deren SAPs nicht vorhersehbar sind, können jedoch nicht abgesichert werden.[22] Der Umweg über SSH ist also höchstens als Übergangslösung anzusehen, bis Sicherheitsdienste in das Kommunikationssystem integriert wurden.

Beispiel: TLS

Ein Beispiel für die Sicherheitsdienste oberhalb des Transportsystems bildet *TLS* (Transport Layer Security) [Dierks 1999]. Es ist die Weiterentwicklung des von der Firma Netscape entwickelten *Secure Socket Layers* (SSL) [Freier 1996]. Der Name TLS legt zwar nahe, daß es sich um eine Implementation innerhalb der Transportschicht handelt,

[21] Ein anderer Weg wurde im Falle der "Secure Transport Library" gewählt. Hier wird die Schnittstelle der Transportschicht vollständig nachgebildet, um diese Änderungen zu vermeiden (siehe Seite 46).

[22] Ein Überblick über die Verwendung von SSH als Tunnel für andere Anwendungsprotokolle findet sich in [Bogen 1998].

dies ist jedoch nicht der Fall. Dies bedeutet, daß jede Anwendung, die die Sicherheitsdienste von TLS in Anspruch nehmen soll, entsprechend angepaßt werden muß. TLS besteht aus zwei Protokollschichten. Die untere davon ist das *TLS Record Protocol*. Es setzt auf einem verläßlichen Transportprotokoll (also z.B. TCP) auf und bietet Vertraulichkeit und Integrität der übertragenen Dateneinheiten. Vertraulichkeit wird durch symmetrische Verschlüsselung unter Verwendung von Sitzungsschlüsseln erreicht. Die Integrität sichert ein Message Authentication Code (MAC), der mittels verschiedener Verfahren (SHA, MD5 u.a.) generiert wird. Das TLS Record Protocol ist auch in der Lage, zu große Dateneinheiten zu fragmentieren und Fragmente zu reassemblieren, wenn dies nötig ist. Oberhalb des TLS Record Protocols befindet sich eine zweite Schicht, das TLS Handshake Protocol. Dieses soll drei verschiedene Eigenschaften der Verbindung gewährleisten:

- Die Authentizität der Gegenseite. Hierzu werden asymmetrische Verfahren eingesetzt. Es kann eine Authentisierung sowohl der Quelle als auch des Ziels einer Verbindung erfolgen.
- Vertraulichkeit des Schlüsselaustauschs.
- Verläßlichkeit des Schlüsselaustauschs. Kein Angreifer kann in den Schlüsselaustausch eingreifen, ohne daß dies von den Kommunikationspartnern bemerkt wird.

Das TLS Handshake Protocol wird also beim Aufbau einer TLS-Verbindung benötigt, um die entsprechenden kryptographischen Informationen zu liefern. Diese bestehen aus einer eindeutigen Sitzungsnummer, einem Zertifikat der Gegenseite, den zu verwendenden Algorithmen für Datenkompression, Verschlüsselung und Authentisierung. Das Zertifikat ist ein X509-Zertifikat, mit dem eine Authentisierung durchgeführt werden kann [CCITT 1997]. Weiterhin existiert ein Eintrag, der angibt, ob aus der aktuellen TLS Verbindung neue Verbindungen zwischen den Kommunikationspartnern initiiert werden können, die denselben Sitzungsschlüssel verwenden. Dies reduziert bei vielen Verbindungen zwischen zwei Prozessen den Aufwand für Schlüsselaustausch und Authentisierung. Diese Informationen des TLS Handshake Protocols werden dann vom TLS Record Protocol für die Verarbeitung der zur Verbindung gehörigen Dateneinheiten benutzt.

SSL wurde ursprünglich zur Absicherung von WWW-Verbindungen entwickelt. Es ist in vielen Web-Browsern direkt in der Anwendung integriert. Eine Integration in das Kommunikationssubsystem ist weder für SSL noch für TLS erfolgt. Dementsprechend ist es ein Protokoll der Anwendungsschicht. Trotz der fehlenden Integration in den TCP/IP Protokollstapel erreicht TLS eine weite Verbreitung. Es gibt bereits einige Bemühungen, Anwendungen an TLS anzupassen [Newman 1999].

2.5 Resümee

Die heute weit verbreiteten TCP/IP-Protokolle sind nicht dazu geeignet, die Vertraulichkeit, Integrität und Authentizität von übertragenen Daten zu gewährleisten. Es gibt

jedoch kryptographische Verfahren, die dies ermöglichen. Der Einsatz dieser Verfahren kann durch Integration der entsprechenden Algorithmen und Protokolle in einzelne Anwendungen erfolgen. Die dadurch entstehende Protokollvielfalt bringt jedoch einige Schwierigkeiten bezüglich der Sicherheit, der Verifizierbarkeit und der Konfigurierbarkeit dieser Anwendungen mit sich.

Eine Integration in den Protokollstapel des Kommunikationssystems könnte dieses Problem beheben. Die Schichtstruktur des Protokollstapels wirft jedoch die Frage auf, in welcher Schicht die Implementierung von Sicherheitsdiensten erfolgen soll. In die Auswahl der Schicht fließt auch die Art der angebotenen Sicherheitsdienste mit ein. Die Realisierung von "End to End"-Sicherheit kann nur ab der Vermittlungsschicht erfolgen. Sicherheitsdienste in tieferliegende Schichten bieten den Vorteil, daß die Schnittstellen dieser Schichten standardisiert sind und sich somit schnelle, herstellerunabhängige Lösungen in Form von Hardware realisieren lassen. Sie können jedoch nur einzelne Datenleitungen von einem Netzwerkknoten zum nächsten Nachbarn absichern, so daß für die Kommunikation zwischen zwei Endsystemen jedes dazwischenliegende System vertrauenswürdig sein muß. Dies trifft jedoch für Kommunikation über Netzwerke unterschiedlichster Betreiber (wie z.B. im Falle des Internets) nicht zu.

Somit kommen eher die höherliegenden Schichten ab der Vermittlungsschicht für die Integration von Sicherheitsmechanismen in Betracht. Zur Verhinderung des weit verbreiteten Fälschens von Absenderadressen der IP-Schicht, das die Grundlage für eine Vielzahl von Angriffen bildet, wurde von der IETF die Vermittlungsschicht als Ziel der Bemühungen ausgewählt. Im Laufe des Entwicklungsprozesses des Sicherheitsstandards *IPsec* wurde jedoch festgestellt, daß sich nur mit tiefgreifenden Änderungen in der Vermittlungsschicht eine umfassende Sicherheit der Kommunikation erreichen läßt. Dies bedeutet de facto die Aufgabe der verbindungslosen Kommunikation und der dynamischen Wegewahl (der Aufbau einer *Security Association* entspricht dem Aufbau einer festen Route, die Datagramme können auch mit einer Sequenznummer versehen werden). Weiterhin wird die Schichtstruktur des Protokollstapels verletzt, um auch in der Vermittlungsschicht bereits anwendungsabhängige Sicherheitsanforderungen zu realisieren.

Sicherheitsdienste in der Vermittlungsschicht sind die einzige Möglichkeit, Angriffe auf die Routingmechanismen des Internets zu unterbinden. Weitere Mechanismen (z.B. Verhinderung von *Replay*-Angriffen durch Vergabe von Sequenznummern oder Authentisierung von Benutzern und Prozessen) sollten jedoch sinnvollerweise in passendere, höhere Schichten integriert werden. Die Transportschicht bietet sich hierfür an, da im TCP/IP-Protokollstapel keine höheren Schichten in das Kommunikationssubsystem integriert wurden. Die Transportschicht stellt also die Schnittstelle zu den Anwendungen dar und ist somit in der Lage, anwendungsspezifische Sicherheitsdienste anzubieten, ohne die Schichtstruktur des Protokollstapels zu verletzen.

Die bisher existierenden Implementierungen von Sicherheitsdiensten in der Transportschicht weisen jedoch einige Schwächen auf, die ihrer Verbreitung entgegenstehen. Die einzige Lösung, die in den Betriebssystemkern integriert ist, ist das *Secure TCP* [Tsutsumi 1995]. Dieses setzt jedoch asymmetrische Verschlüsselungsverfahren ein und liefert daher nur einen geringen Datendurchsatz. Eine Authentisierung der Kommunikationspartner ist ebenfalls nicht vorgesehen. Die Integration von flexiblen, erweiterbaren Sicherheitsdiensten in die Transportschicht ist also nach wie vor notwendig. In den

folgenden Kapiteln wird daher anhand eines Prototyps gezeigt, wie eine entsprechende Erweiterung aussehen kann. Hierzu wurde das verbindungsorientierte Protokoll TCP ausgewählt, da es die Dienste der Vermittlungsschicht um weitreichende Fähigkeiten erweitert und die Kommunikationsgrundlage der meisten heutigen Anwendungen darstellt.

Kapitel 3

Entwurf eines sicheren TCP

In diesem Kapitel erfolgt der Entwurf eines modifizierten TCP-Protokolls. Dieses soll zusätzlich zu den durch TCP erbrachten Diensten Sicherheitsmechanismen beinhalten. Zunächst werden die grundlegenden Anforderungen an ein solches System angeführt. Darauf aufbauend wird beschrieben, wie die Sicherheitsdienste in das Transportprotokoll integriert werden können. Weiterhin werden geeignete Algorithmen und Protokolle ausgewählt, mit denen die Sicherheitsdienste realisiert werden können. Es folgt eine Spezifikation des resultierenden Gesamtprotokolls. Zum Abschluß wird die Struktur einer entsprechenden Implementation und ihrer Schnittstellen entworfen.

3.1 Allgemeines Anforderungsprofil

Einmal etablierte und standardisierte Protokolle sind langlebige Wesen. Insbesondere Netzwerkprotokolle sind schwer durch neue zu ersetzen, da sie jeweils nicht nur lokal, sondern auch auf der Seite der anderen Kommunikationspartner eingesetzt werden müssen. Im Internet würde ein Protokollwechsel das Umstellen von Millionen von Rechnern erfordern. Wenn die neuen Protokolle modifizierte Schnittstellen besitzen, müssen auch die angrenzenden Protokollschichten entsprechend angepaßt werden. Insbesondere im Falle von Hardware-Implementationen des zu ersetzenden Protokolls ist ein Protokollwechsel langwierig und kostspielig. Da neue Protokolle noch nicht in jahrelangen Tests und allen möglichen Konfigurationen bewährt haben, erfordern sie oftmals auch das spätere Nachbessern an Mängeln des Entwurfs. Der nötige Aufwand vergrößert sich dadurch zusätzlich [O'Malley 1991].

Eine andere Vorgehensweise ist die Weiterentwicklung eines etablierten Protokolls, ohne jedoch die Kompatibilität zu diesem zu bewahren. Nötige Ergänzungen zu einem alten Protokoll können auf diese Weise ohne Rücksicht auf die Vorgängerprotokolle angebracht werden. Da diese inkompatiblen Protokolle sich jedoch nur sehr langsam durchsetzen, muß das alte Protokoll weiterhin verfügbar gehalten werden. Insbesondere ist eine Entscheidung notwendig, welches der Protokolle für eine Kommunikationsbeziehung zum Einsatz kommen soll. Für eine eingehende Verbindung bedeutet dies, daß das vom Client benutzte Protokoll vom Server identifiziert werden muß [O'Malley 1991]. Das IP-Protokoll kennt eine solche Identifizierungsmöglichkeit, jedoch können hierbei

nur 2^8 Protokolle unterschieden werden. Für unterschiedliche Versionen einer Vielzahl von Protokollen ist dies nicht ausreichend. Ein zusätzliches Protokoll zur Verwaltung und Erkennung von verfügbaren Protokollen und Protokollversionen wäre eine Lösung des Problems, ist jedoch sehr aufwendig. Zusätzlich ist die auf diese Weise entstehende Protokollvielfalt unter Sicherheitsgesichtspunkten nicht zu vertreten, da die Interaktionsmöglichkeiten zwischen den Protokollen zu neuen Angriffsmöglichkeiten führen [Kelsey 1997].

Es bleibt die Möglichkeit, Protokolle unter Bewahrung der **Kompatibilität** an neue Bedürfnisse anzupassen. Die modifizierte Protokollversion erfordert somit keinen sofortigen Umstieg aller Kommunikationspartner auf das neue Protokoll. In Bezug auf Sicherheitsdienste ist jedoch zu beachten, daß die neuen Dienste nur dann eingesetzt werden können, wenn beide Kommunikationspartner über das entsprechend erweiterte Protokoll verfügen. Insbesondere ist festzulegen, ob andernfalls eine Kommunikation ohne Sicherheitsdienstgüte geführt oder die Verbindung abgebrochen werden soll.

Eine weitere Anforderung ist die **Erweiterbarkeit** und Anpaßbarkeit der Sicherheitsdienste. Mit der zunehmenden Leistung von Prozessoren steigen auch die Anforderungen an kryptographische Algorithmen. Wachsende Erkenntnisse über Schwächen einzelner Algorithmen erfordern entsprechende Modifikationen oder Neuentwicklungen. Dies bedeutet, daß oftmals entweder die Schlüssellänge vergrößert oder ein neuer Algorithmus eingesetzt werden muß. Protokolle, die solche Verfahren verwenden, sollten leicht an diese geänderten Gegebenheiten angepaßt werden können. Insbesondere eine wachsende Länge der auszutauschenden kryptographischen Informationen bereitet bei der Verwendung heutiger Protokolle Schwierigkeiten, da diese oftmals nur über Einträge mit fester Länge verfügen. Das Austauschverfahren der kryptographischen Informationen sollte möglichst eine hohe **Orthogonalität** besitzen, d.h. es muß unabhängig von der Art der zu übertragenden Informationen (Schlüssel, Authentisierungscodes u.ä.) und den verschiedenen Zeitpunkten der Datenübertragung (beim Verbindungsaufbau oder bei bereits bestehender Verbindung) sein.

Wichtig für die Implementierung von Sicherheitsdiensten ist die **Konfigurierbarkeit**. Sicherheitsdienste innerhalb des TCP-Protokolls sollten sich auf zweierlei Weise konfigurieren lassen. Zum einen sind Vorgaben des Systemadministrators notwendig, die für Anwendungen, welche von sich aus keine Sicherheitsdienste verwenden, eine Standardsicherheitsdienstgüte spezifizieren. Weiterhin ist eine Schnittstelle sinnvoll, die es Anwendungen ermöglicht, die von ihnen gewünschte Sicherheitsdienstgüte anzugeben. Im Konfliktfalle sollten vorgegebene Mindestanforderungen des Administrators nicht von der Anwendung unterschritten werden können. Andernfalls könnten auf diese Weise die Sicherheitsdienste umgangen werden.

Die Sicherheitsdienste sollten zusätzlich eine möglichst hohe **Transparenz** besitzen. Sicherheitsmechanismen bringen häufig Unannehmlichkeiten mit sich, wie z.B. wiederholte Paßwortabfragen oder lange Wartezeiten. Aus Bequemlichkeit werden die Sicherheitsdienste dann oftmals umgangen oder außer Kraft gesetzt. Beispiele hierfür sind einfache bzw. nicht eingerichtete Paßwörter oder hostbasierte Autorisierung. Eine hohe Transparenz und eine einfache Handhabung der Sicherheitsdienste gewähren die Akzeptanz und den reibungslosen Einsatz der Sicherheitsdienste.

Eine der wichtigsten Anforderungen an die Sicherheitsdienste ist die **Skalierbarkeit**.

Da ein Einsatz auch in großen Netzwerken wie dem Internet möglich sein soll, muß gewährleistet sein, daß keine Engpässe entstehen. Eine Verschlüsselung kann i.d.R. jeweils auf den Endsystemen vorgenommen werden, die Schlüsselverteilung und das Überprüfen von Zertifikaten muß jedoch auf anderem Wege erfolgen, da die entsprechenden Daten nicht auf jedem Rechner vorgehalten werden können. Ein zentraler Server bedeutet zwar eine einfache Lösung dieses Problems, bildet jedoch in großen Netzwerken einen Engpaß. Zusätzlich führt ein Ausfall dieses zentralen Servers dazu, daß die angebotenen Sicherheitsdienste im gesamten Netz nicht mehr angeboten werden. Die Komponenten der Sicherheitsdienste sollten also eine hohe **Verfügbarkeit** garantieren. Im engen Zusammenhang damit steht das Kriterium der **Robustheit**. Sicherheitsdienste sollten gegen Angriffe weitgehend immun sein. Dies gilt nicht nur für die kryptographischen Verfahren und Protokolle. Die Implementierung muß auftretende Fehler (z.B. Pakete mit Überlänge oder unerwartete Einträge in PCI-Feldern) rechtzeitig erkennen können, um nicht in unerwartete Zustände zu geraten.

3.2 Sicherheitsdienste im Transmission Control Protocol

Zum Schutz von Verbindungen des *Transmission Control Protocols* werden Sicherheitsmechanismen benötigt. Sowohl hinsichtlich der Dienste, die angeboten werden, als auch der Verfahren, mit denen diese realisiert werden, ist eine Auswahl notwendig. Hierfür müssen die Eigenschaften des TCP-Protokolls berücksichtigt werden, um einen sinnvollen und implementierbaren Entwurf zu erhalten. Zunächst erfolgt eine generelle Beschreibung der Integrationsmöglichkeiten von Sicherheitsdiensten in das TCP-Protokoll. In den darauf folgenden Abschnitten werden die möglichen Sicherheitsdienste vorgestellt und entsprechende Algorithmen und Protokolle ausgewählt.

Verbindungsaufbau

Das Transmission Control Protocol stellt höheren Schichten verbindungsorientierte Kommunikation zur Verfügung. Daher bietet es sich unter dem Gesichtspunkt der Kompatibilität an, beim Verbindungsaufbau festzustellen, ob der Kommunikationspartner ebenfalls über das erweiterte Protokoll verfügt. Es kann versucht werden, diese Überprüfung der Sicherheitsfunktionalität des Kommunikationspartners in den normalen Verbindungsaufbau zu integrieren, ohne das Standardformat der entsprechenden Segmente zu verändern. Auf diese Weise läßt sich die Kompatibilität zum normalen TCP bewahren. Haben sich die beiden Partnerinstanzen so als Instanzen eines sicheren TCPs zu erkennen gegeben, können sie sich auf ein beliebiges anderes Protokoll einigen, das von beiden beherrscht wird. Jedoch erscheint ein Beibehalten der wesentlichen Form des Protokolls TCP sinnvoll, da sich dieses bereits seit Jahrzehnten bewährt hat. Der Verzicht auf Veränderungen am TCP-Header bedeutet insbesondere, daß dieser nicht verschlüsselt werden kann. Somit kann eine Datenflußanalyse nicht verhindert werden. Der Einsatz von Firewall-Komponenten, die auch die TCP-Header auf Plausibilität überprüfen, bleibt dadurch jedoch möglich.

Zusätzlich zur Erkennung der Sicherheitsdienste des Kommunikationspartners bietet es sich an, beim Aufbau einer TCP-Verbindung bereits die zur Erbringung der angefor-

derten Sicherheitsdienstgüte erforderlichen Algorithmen auzuhandeln. Diese Algorithmen sollten nicht statisch vorgegeben werden, da kryptographische Verfahren durch neue Erkenntnisse über Angriffsmöglichkeiten und technischen Fortschritt veralten. Die Integration neuerer Verfahren kann daher notwendig werden.

Zum Aushandeln der Verfahren muß jeweils der Kommunikationspartner darüber informiert werden, welche Algorithmen auf der Gegenseite verfügbar sind. Auch der Einsatz unterschiedlicher Algorithmen für die beiden Kommunikationsrichtungen kann sinnvoll sein, wenn z.B. einer der Kommunikationspartner über Verschlüsselungshardware für bestimmte Algorithmen verfügt. In die eine Richtung wird die Kommunikation dann hardwareunterstützt verschlüsselt, die Kommunikation in die andere Richtung kann durch einen Algorithmus, der effizient in Software implementierbar ist, chiffriert werden. Somit sollten für jede Kommunikationsrichtung eigene Algorithmen abgemacht werden. Da für den Einsatz jeglicher kryptographischer Verfahren Schlüssel benötigt werden, muß neben den Verfahren für Verschlüsselung, Prüfsummenbildung etc. auch eine Einigung auf das zu verwendende Schlüsselaustauschverfahren erfolgen.

Zusätzlich müssen beim Verbindungsaufbau weitere Daten für die gewählten Algorithmen ausgetauscht werden wie z.B. die Schlüssel für eine gewählte Verschlüsselungsart.

Der Austausch dieser Vielzahl von Informationen sollte in möglichst wenigen Datentransfers erreicht werden, um eine allzugroße Verzögerung beim Verbindungsaufbau gegenüber dem normalen TCP zu vermeiden. Insbesondere bei vielen kurzzeitigen TCP-Verbindungen, wie sie typischerweise beim Einsatz des HTTP-Protokolls vorkommen, wären sonst starke Geschwindigkeitseinbußen die Folge.[1]

Übertragung der Nutzdaten

Nach dem Verbindungsaufbau, bei dem die Algorithmen und zu verwendenden Schlüssel etc. ausgehandelt wurden, können die eigentlichen Nutzdaten der Anwendung übertragen werden. Jedes dafür verwendete Segment muß jedoch um zusätzliche Informationen erweitert werden, um die vereinbarten Sicherheitsdienste zu ermöglichen. Zur Sicherung der Integrität ist z.B. eine kryptographisch gesicherte Prüfsumme in jedem Segment notwendig, die eine Kontrolle über die Unversehrtheit der Daten ermöglicht.

Weiterhin kann es bei lange bestehenden Verbindungen notwendig sein, neue Schlüssel auszuhandeln, damit nicht große Datenmengen mit demselben Schlüssel chiffriert werden. Diese kryptographischen Informationen sollten ebenfalls in den normalen TCP-Segmenten übertragen werden, wobei sie von den Nutzdaten der Anwendung unterscheidbar sein müssen.

Die folgenden Unterabschnitte beinhalten die Auswahl von Sicherheitsdiensten und den dafür benötigten Algorithmen und Protokollen. Der für diese Verfahren benötigte Austausch von Daten (z.B. von Schlüsseln und Zertifikaten) bildet die Grundlage des Protokollentwurfs im Abschnitt 3.3.

[1] Das *Hypertext Transfer Protocol* (HTTP) besitzt jedoch ab der Version 1.1 die Möglichkeit, mehrere Anfrage-Antwort-Paare über eine TCP-Verbindung zu multiplexen [Fielding 1999].

3.2.1 Vertraulichkeit

Die Vertraulichkeit von zu übertragenden Daten läßt sich durch Verschlüsselung sicherstellen. Da die Kompatibilität zum TCP-Protokoll bestehen bleiben muß, können die Header der TCP-Segmente allerdings nicht verschlüsselt werden. Dies bedeutet, daß sich eine Verkehrsflußanalyse kaum verhindern läßt, da sowohl der Absender als auch der Adressat im Paketkopf vermerkt sind. Wird jeweils der Datenteil der Segmente verschlüsselt und lassen sich darin befindliche Daten von zusätzlicher Protokollinformation unterscheiden, so können Segmente mit entsprechender Kennung und ansonsten zufälligem Dateninhalt erzeugt und verschlüsselt werden. Der Empfänger kann dann nach der Entschlüsselung erkennen, daß es sich um ein Zufallspaket handelt und es verwerfen. Da jedoch das Bestehen der Verbindung ohnehin bekannt ist, dürfte dieses Verfahren, das nur die Menge und Zeitpunkte der einzelnen Datensegmente verschleiert, wenig praktischen Nutzen haben.

Da der Verschlüsselungsvorgang nur einen möglichst geringen Zeit- und Rechenaufwand verursachen darf, um hohe Übertragungsraten zu ermöglichen, kommt hierfür nur ein symmetrisches Verfahren in Betracht. Den Schwierigkeiten der sicheren, geheimen Verteilung der entsprechenden Schlüssel kann durch Einsatz eines hybriden Verfahrens begegnet werden. Zum einen können die geheimen Schlüssel des symmetrischen Verfahrens übertragen werden, nachdem sie durch einen asymmetrischen Algorithmus verschlüsselt wurden. Hierzu muß der öffentliche Schlüssel des Kommunikationspartners bekannt sein. Eine andere Möglichkeit ist der Einsatz eines speziellen Schlüsselaustauschverfahrens wie z.B. *Diffie-Hellman* [Diffie 1976]. Um das Diffie-Hellman-Verfahren gegen "Man-in-the-Middle"-Angriffe abzusichern, müssen die auszutauschenden Informationen allerdings signiert werden, so daß auch hier der öffentliche Schlüssel des Kommunikationspartners bekannt sein muß.

Da eine lokale Kenntnis aller benötigten, öffentlichen Schlüssel in großen Netzwerken unpraktikabel ist, bietet es sich an, diese bei jedem Verbindungsaufbau auszutauschen. Dies geschieht am besten in Form von Zertifikaten, in denen sowohl der Besitzer des Schlüssels als auch der Schlüssel selbst enthalten sind. Das Zertifikat wird von einer sog. Zertifizierungsinstanz (CA = Certification Authority) ausgestellt, die das Zertifikat signiert und somit zusichert, daß der enthaltene öffentliche Schlüssel dem angegebenen Besitzer gehört. Weitere Inhalte des Zertifikates können der Gültigkeitszeitraum des Zertifikates und der Name der ausstellenden CA sein. Der Name der CA ist einerseits ein Kriterium um abzuschätzen, wie aussagekräftig das Zertifikat ist (wieviel Vertrauen man der Zertifizierungsinstanz entgegenbringt). Andererseits wird diese Information benötigt, um im Falle von widerrufbaren Zertifikaten die Gültigkeit des Zertifikates durch eine Anfrage an die Zertifizierungsinstanz zu überprüfen.

Vor dem Austausch der Zertifikate ist jedoch die Kenntnis der einzusetzenden Algorithmen notwendig, um ein Zertifikat mit einem entsprechenden Schlüssel zu versenden. Hat ein Kommunikationspartner das Zertifikat seines Gegenübers erhalten, prüft er zunächst dessen Gültigkeit. Ist das Zertifikat gültig und nach der eigenen Sicherheits-Policy annehmbar, so sendet er sein eigenes zurück.

Der für die Verschlüsselung notwendige Informationsaustausch zwischen beiden Kommunikationspartnern wird in Abbildung 3.1 noch einmal dargestellt.

Partner A			**Partner B**
List(Key Exchange Algorithms)	$\rightarrow$		
		$\leftarrow$	Choice(Key Exchange Algorithm)
List(symmetric Algorithms)$_{B\rightarrow A}$	$\rightarrow$	$\leftarrow$	List(symmetric Algorithms)$_{A\rightarrow B}$
Choice(symmetric Algorithm)$_{A\rightarrow B}$	$\rightarrow$	$\leftarrow$	Choice(symmetric Algorithm)$_{B\rightarrow A}$
Certificate$_A$	$\rightarrow$	$\leftarrow$	Certificate$_B$
Key Exchange Values	$\rightarrow$	$\leftarrow$	Key Exchange Values

Abbildung 3.1: Informationsaustausch für die Verschlüsselung

Die auszutauschenden Informationen sollten jedoch in möglichst wenigen Datensegmenten übermittelt werden, um den Verbindungsaufbau zu beschleunigen. Insbesondere bei Verbindungen über Medien mit großen Verzögerungszeiten (z.B. via Satellit) bedeutet jeder Austausch von Datensegmenten eine beträchtliche Zeitspanne. Der normale TCP-Verbindungsaufbau mit seinem "Drei-Wege-Handshake" sollte nicht unnötig erweitert werden.

Hat Kommunikationspartner B die Liste der möglichen Schlüsselaustauschalgorithmen von A erhalten, so wählt er den ersten Vorschlag aus, der folgenden Anforderungen genügt:

- Der Algorithmus wird von B unterstützt.
- B besitzt ein Zertifikat, das einen Schlüssel enthält, der zu dem Schlüsselaustauschverfahren paßt (mit dem sich z.B. signierte Diffie-Hellman Werte verifizieren lassen).

Ist ein solcher Algorithmus verfügbar, wählt B diesen aus und sendet diese Wahl an A. Zu diesem Zeitpunkt ist B bereits in der Lage, ein entsprechendes Zertifikat auszuwählen, und dieses im gleichen Datensegment mitzusenden. Im Falle des Diffie-Hellman-Verfahrens benötigt B das Zertifikat von A nicht, um den Schlüsselaustauschwert an A zu versenden. Er kann ebenfalls im selben Datensegment verschickt werden. Beim Übertragen von verschlüsselten Schlüsseln muß der öffentliche Schlüssel des Kommunikationspartners bereits bekannt sein, so daß im generellen Falle der Schlüsselaustauschwert erst nach Erhalt des Zertifikates der Gegenseite versandt werden kann.

Abbildung 3.2 zeigt noch einmal den notwendigen Informationsaustausch beim Verbindungsaufbau. Hierbei wurden die Informationen schon weitgehend zu Datenpaketen zusammengefaßt, um mit einer geringen Anzahl von Austauschvorgängen auszukommen.

3.2.2 Integrität

Mit Hilfe von Einweg-Hashfunktionen kann die Integrität der übertragenen Daten sichergestellt werden. Dazu wird ein Hashwert über die Daten gebildet und dieser im Segment selbst mitverschickt. Damit ein Angreifer nicht nach einer Modifikation der

Partner A		**Partner B**
List(Key Exchange Algorithms) List(symmetric Algorithms)$_{B\rightarrow A}$	$\rightarrow$	
	$\leftarrow$	Choice(Key Exchange Algorithm) Choice(symmetric Algorithm)$_{B\rightarrow A}$ List(symmetric Algorithms)$_{A\rightarrow B}$ Certificate$_B$
Choice(symmetric Algorithm)$_{A\rightarrow B}$ Certificate$_A$ Key Exchange Values	$\rightarrow$	
	$\leftarrow$	Key Exchange Values

Abbildung 3.2: Optimierter Informationsaustausch für die Verschlüsselung

Daten den Hashwert erneut berechnen und ins Segment eintragen kann, muß dieser in verschlüsselter Form übertragen werden. Bei einem asymmetrischen Verfahren kann zur Verschlüsselung sowohl der öffentliche Schlüssel des Empfängers dienen, als auch der geheime Schlüssel des Absenders, wenn das Verfahren kommutativ ist ($D_{K_{public}}(E_{K_{secret}}(M)) = D_{K_{secret}}(E_{K_{public}}(M)) = M$). Letzteres Verfahren bestätigt zugleich auch die Authentizität des Absenders, da nur er den benötigten geheimen Schlüssel besitzt. Bei beiden Verfahren ist die Kenntnis des öffentlichen Schlüssels des Kommunikationspartners erforderlich, im ersteren zum Verschlüsseln auf der Seite des Senders, im letzteren zum Entschlüsseln auf der Seite des Empfängers. Wurde bereits ein Sitzungsschlüssel für die Verschlüsselung der Verbindung vereinbart, kann dieser ebenfalls für die Verschlüsselung des Hashwertes benutzt werden. Dies reduziert den Verschlüsselungsaufwand und die nötige Länge des Hashwertes, da symmetrische Algorithmen i.d.R. weniger Rechenzeit benötigen und kleinere Blockgrößen verwenden.

Vor Einsatz einer Hashfunktion muß der zu verwendende Algorithmus zunächst zwischen den Kommunikationspartnern ausgehandelt werden. Auch hier kann dies einmalig beim Verbindungsaufbau erfolgen. Es kann ebenfalls sinnvoll sein, für unterschiedliche Kommunikationsrichtungen verschiedene Hashalgorithmen zu verwenden, um eine optimale Unterstützung durch eventuell vorhandene Hardware zu ermöglichen (siehe Seite 57).

Da für die Sicherung der Vertraulichkeit bereits der Austausch eines Zertifikates und eine Vereinbarung eines Sitzungsschlüssels notwendig ist (siehe Abschnitt 3.2.1), bietet es sich an, dasselbe Verfahren auch hier einzusetzen. Nach dem Zertifikatsaustausch kann sowohl für die Verschlüsselung als auch für ein Signaturverfahren ein entsprechender Sitzungsschlüssel vereinbart werden. Der resultierende Informationsaustausch für die Gewährleistung der Integrität wird in Abbildung 3.3 gezeigt.

Die Übereinstimmung dieses Vorgangs mit dem Austausch der notwendigen Informationen für die Verschlüsselung ermöglicht eine einfache, einheitliche Integration beider Mechanismen in den Verbindungsaufbau. Abschnitt 3.3 zeigt den resultierenden Verbindungsaufbau für die gemeinsame Nutzung aller angebotenen Dienste.

Da beim Verbindungsaufbau zunächst einmal die Algorithmen für die Integritätsprüfung ausgehandelt werden müssen, kann die Integrität dieser ersten Segmente noch nicht gewährleistet werden. Wenn ein Angreifer die Liste der verfügbaren Algorithmen oder die erfolgte Auswahl verändert, wird der Empfänger jedoch die

Partner A		**Partner B**
List(Key Exchange Algorithms) List(Signature Algorithms)$_{B\rightarrow A}$	$\rightarrow$	
	$\leftarrow$	Choice(Key Exchange Algorithm) Choice(Signature Algorithm)$_{B\rightarrow A}$ List(Signature Algorithms)$_{A\rightarrow B}$ Certificate$_B$
Choice(Signature Algorithm)$_{A\rightarrow B}$ Certificate$_A$ Key Exchange Values	$\rightarrow$	
	$\leftarrow$	Key Exchange Values

Abbildung 3.3: Optimierter Informationsaustausch für die Sicherung der Integrität

Verbindung beenden, wenn die Algorithmen nicht seinen Sicherheitsanforderungen entsprechen. Somit kann ein Angreifer, der Pakete mithören und verändern kann, das Zustandekommen der Verbindung verhindern, was er jedoch ohnehin durch Zerstören oder Einbehalten aller Segmente erreichen kann. Es besteht die Möglichkeit, die Kommunikationspartner gezielt mit modifizierten Segmenten zu versorgen, so daß Empfänger und Sender verschiedene Annahmen über die einzusetzenden Algorithmen haben. Aus diesem Grunde sollte beim Schlüsselaustausch ebenfalls die Information in signierter Weise übertragen werden, zu welchem Algorithmus der zu generierende Schlüssel gehört. Der weitere Verlauf des Verbindungsaufbaus kann jedoch nicht mehr gestört werden. Da das übertragene Zertifikat signiert ist, wird eine Änderung an diesem auf der Empfängerseite erkannt. Ein Verändern der signierten Diffie-Hellman-Werte bleibt ebenfalls nicht unbemerkt. Beim Austausch von verschlüsselten Schlüsseln können diese von einem Angreifer verändert werden. Dies bedeutet, daß die Entschlüsselung bzw. Integritätsprüfung aller folgenden Datensegmente fehlschlägt. Eine falsche Entschlüsselung sollte also für den Empfänger erkennbar gemacht werden, indem die Ausgangsdaten mit entsprechenden Markierungen versehen werden, deren Vorhandensein nach der Entschlüsselung überprüft werden kann.

Die Integrität der Daten sollte auch im Falle eines sog. Replay-Angriffs gewährleistet sein. Da der verwendete Schlüssel zum Signieren des Segmentes für jede Verbindung erneut ausgehandelt wird, wird das Einspielen eines alten Segmentes aus einer vorigen Verbindung vom Empfänger erkannt. Das Erkennen eines alten Segmentes, das derselben Verbindung entstammt, läßt sich durch eine Sequenznummer ermöglichen, die für jedes Segment jeweils erhöht wird. Als einfache Lösung bietet es sich an, die von TCP verwendete Sequenznummer mit in die Berechnung der Prüfsumme einzubeziehen. Da diese aus 32 Bits besteht, könnte ein Segment nach 2^{32} übertragenen Bytes wiederholt werden. Das Vereinbaren eines neuen Schlüssels vor dem Überlauf des Sequenzzählers bietet dann Schutz gegen diesen Angriff. Ein wiederholtes Einspielen der ersten Segmente einer Verbindung ist weiterhin möglich, da diese keine sichere Integritätsprüfsumme enthalten. Jedoch stellt das Übertragen der Zertifikate und der darauf aufbauende Schlüsselaustausch erneut sicher, daß nur der Eigner des Zertifikates Nutzdaten über diese Verbindung senden bzw. empfangen kann.

3.2.3 Authentizität

In den vorigen Abschnitten wurden Zertifikate benutzt, um einen Schlüssel für den benötigten Schlüsselaustausch zu übertragen. Beim Schlüsselaustausch beweist der Kommunikationspartner bereits, daß das übertragene Zertifikat wirklich zu ihm gehört, da er hierfür den zugehörigen geheimen Schlüssel benötigt. Im Falle des Diffie-Hellman-Verfahrens kann nur der Eigner des Zertifikates den zu übertragenden Diffie-Hellman-Wert korrekt signieren. Werden verschlüsselte Schlüssel übertragen, wird der Sitzungsschlüssel mit dem Public Key aus dem Zertifikat verschlüsselt. Nur der Eigner des Zertifikates kann diese Verschlüsselung rückgängig machen, um den korrekten Sitzungsschlüssel zu erhalten. Übermittelt ein Angreifer ein fremdes Zertifikat, so gelangt er nicht in den Besitz des korrekten Sitzungsschlüssels. Wird jedes folgende Datensegment verschlüsselt und kann nach der Entschlüsselung festgestellt werden, ob die Daten korrekt entschlüsselt wurden (z.B. durch vorgeschriebene Einträge in den Ausgangsdaten oder durch eine Prüfsumme innerhalb der chiffrierten Daten), sind solche Angriffe ausgeschlossen. Ein *Message Authentication Code* in jedem Segment reicht bereits aus, um dieses zu gewährleisten.

Die auf diese Weise zugesicherte Authentizität bedeutet jedoch nur, daß der Kommunikationspartner wirklich derjenige ist, dessen Zertifikat übermittelt wurde. Die Auswahl des Kommunikationspartners geschieht im Falle von TCP jedoch nicht auf Grund von Zertifikaten, es wird als Adressat die IP-Adresse und die Portnummer des Empfängers angegeben. Ob der antwortende Gegenüber also wirklich zu dem angegebenen Paar von IP-Adresse und Portnummer gehört, läßt sich nicht ohne weiteres dem Zertifikat entnehmen.

Ausgestellte Zertifikate könnten als Eigentümer die IP-Adresse und die Portnummer enthalten, so daß der Empfänger des Zertifikates überprüfen kann, ob es sich beim Gegenüber auch um den gewünschten Kommunikationspartner handelt. Jedoch ist der Verwaltungsaufwand von Zertifikaten für jede Kombination von IP-Adresse und Portnummer sehr hoch. Zudem bekommen Prozesse, die eine Verbindung initiieren, oftmals dynamisch vom Betriebssystem eine beliebige, unbenutzte Portnummer zugewiesen.[2] Ein entsprechendes Zertifikat für einen Clientprozeß kann also zuvor nicht erstellt werden, da die lokale Portnummer erst zur Laufzeit vergeben wird.

Die Eigentümer der Zertifikate sollten also eher generischer Natur sein. Es könnte sich dabei z.B. um den Namen und die Email-Adresse eines Benutzers handeln. Bei Servern kann ebenso der Name des Dienstes (z.B. “www.subdomain.domain” oder “ftp.subdomain.domain”) oder der des entsprechenden Administrators und die Email-Adresse des zuständigen Ansprechpartners im Zertifikat eingetragen werden. Eine automatische Verifizierung, ob der Antwortende auch der gewünschte Kommunikationspartner ist, ist dann jedoch schwierig, da diese Angaben nicht unbedingt mit bestimmten IP-Adressen und Portnummern korrelieren. Eine Email-Adresse enthält i.d.R. nur den Domain-Namen, nicht jedoch den vollständigen Namen eines bestimmten Rechners. Die Namen von Diensten lassen sich evtl. durch einen DNS-Request auflösen, müssen aber nicht unbedingt immer zu einer bestimmten IP-Adresse gehören. Dies ist z.B. bei

[2] Der Bereich der Portnummern von 0 bis 1024 wird von vielen Betriebssystemen nur an privilegierte Prozesse vergeben. Dies sind i.d.R. Serverprozesse, die auf festgelegten Portnummern auf eingehende Verbindungen warten. Der restliche Bereich bis 65535 steht für Benutzerprozesse zur Verfügung.

verteilten Webservern der Fall, bei denen die DNS-Anfragen mit den verschiedenen IP-Adressen der Webserver beantwortet werden (Round-Robin-DNS oder Subnet Sorting).

Da der Schlüsselaustausch auf dem im Zertifikat enthaltenen Schlüssel aufbaut, kann nur der Eigentümer des übertragenen Zertifikats diesen erfolgreich durchführen. Ein spezieller Schlüssel und ein dazugehöriger Algorithmus zum Signieren der Segmente könnten beim Verbindungsaufbau ausgehandelt werden. Dies würde bedeuten, daß jedes zu übertragende Segment einen *Message Authentication Code* enthalten müßte, der die Authentizität sicherstellt. Dieser MAC würde z.B. beim Einsatz des DSA (Digital Signature Algorithm) eine Länge von 20 Bytes (160 Bits) haben. Da jedoch bereits für die Sicherung der Integrität ein solcher Algorithmus Verwendung findet, kann dieser für die Sicherung sowohl der Integrität als auch der Authentizität des Segmentes benutzt werden. Dadurch bleibt in jedem Segment mehr Raum für die Nutzdaten der Anwendung. Andernfalls müßte der MAC für die Authentisierung jeweils unabhängig von dem für die Integrität berechnet werden, so daß weitere Rechenzeit benötigt wird.

Für die Sicherung der Authentizität können bei der Berechnung des MAC weitere Einträge des TCP-Headers mit einbezogen werden. Die Portnummern von Sender und Empfänger könnten z.B. ebenfalls in die Berechnung einfließen. Ein Verfahren, das bereits beim Generieren der Prüfsumme von UDP-Datagrammen benutzt wird, ist die Verwendung eines sog. Pseudo-Headers bei der Berechnung. Hierfür wird die Prüfsumme nicht nur über den normalen UDP-Header gebildet. Es wird dem Datagramm ein zusätzlicher Kopfteil vorangestellt, in dem zusätzlich die IP-Adressen von Absender und Empfänger sowie die Länge des Datagramms enthalten sind. Über diesen Kopfteil und die Nutzdaten wird nun die Signatur berechnet. Der Pseudo-Header wird jedoch nicht übertragen. Auf der Empfängerseite wird erneut der Pseudo-Header gebildet und den Nutzdaten vorangestellt. Nun erst wird die übertragene Prüfsumme verifiziert. Mit diesem Verfahren kann im Falle von TCP ebenfalls nicht nur die Korrektheit der im Segment enthaltenen Portnummern, sondern auch die der IP-Adressen geprüft werden. Somit werden auch die vollständigen SAPs der Verbindung (IP-Adresse und Portnummer) durch die Prüfsumme gesichert.

3.2.4 weitere Dienste

Weitere Sicherheitsdienste werden zunächst nicht implementiert. Eigenschaften wie Unleugbarkeit sind i.d.R. stark abhängig von der Anwendung, die Unleugbarkeit einer TCP-Verbindung und aller darüber übertragenen Daten ist nur schwer realisierbar. Da die “Message-Authentication-Codes” nur über einzelne Segmente gebildet werden, müßte eine zusätzliche Gesamtsignatur über alle übertragenen Daten gebildet oder es müßten alle Einzelsignaturen mitprotokolliert werden. Diese Art von Signaturen lassen sich von Anwendungen sehr viel einfacher realisieren, da diese über die gesamten, unsegmentierten Daten verfügen.

Eine Zusicherung der Anonymität von Benutzern läßt sich durch Verwendung von Zertifikaten gewährleisten, die keine Nutzeridentität enthalten, sondern stattdessen nur einen zufälligen Wert. Es könnten auch für alle Nutzer eines Netzwerkes Zertifikate verwendet werden, die nur den Domainnamen des Netzes als Identität angeben (oder ein Zertifikat für einen anonymen Nutzer wie z.B. *niemand@meine-heimat.de*). Jedes

Partner A		**Partner B**
List(Key Exchange Algorithms)	$\rightarrow$	
List(symmetric Algorithms)$_{B \rightarrow A}$		
List(Signature Algorithms)$_{B \rightarrow A}$		
	$\leftarrow$	Choice(Key Exchange Algorithm)
		Choice(symmetric Algorithm)$_{B \rightarrow A}$
		Choice(Signature Algorithm)$_{B \rightarrow A}$
		List(symmetric Algorithms)$_{A \rightarrow B}$
		List(Signature Algorithms)$_{A \rightarrow B}$
		Certificate$_B$
Choice(symmetric Algorithm)$_{A \rightarrow B}$	$\rightarrow$	
Choice(Signature Algorithm)$_{A \rightarrow B}$		
Certificate$_A$		
Key Exchange Values		
	$\leftarrow$	Key Exchange Values

Abbildung 3.4: Optimierter Informationsaustausch für alle Sicherheitsdienste

Zertifikat sollte jedoch einen anderen Schlüssel enthalten. Auf diese Weise wäre weiterhin die Vertraulichkeit der Verbindungen verschiedener Systembenutzer gewährleistet.

Weitere Ergänzungen wie z.B. besser geeignete Sequenznummern als die von TCP verwendeten Bestätigungsnummern sollten sich leicht im nachhinein implementieren lassen. Eine Datenflußanalyse läßt sich bei Verwendung von Verschlüsselung erschweren, indem entsprechend gekennzeichnete Segmente mit zufälligem Inhalt in den Datenstrom eingefügt werden. Dies verhindert zumindest Rückschlüsse auf die Menge der übertragenen Daten. Eine Datenflußanalyse läßt sich jedoch auf der gewählten Ebene nicht vollständig vermeiden, da die Header aus Gründen der Kompatibilität mit dem bestehenden TCP-Protokoll nicht verschlüsselt werden können. Aus diesem Grunde werden entsprechende Dienste zunächst nicht implementiert werden. Die gewählten Datenformate sollten jedoch eine spätere Ergänzung ermöglichen.

3.3 Protokoll des sicheren TCP

In den folgenden Abschnitten wird ein verändertes TCP-Protokoll entworfen, das die vorgestellten Sicherungsmechanismen enthält. Weiterhin werden die hierfür benötigten Formate der zu übertragenden Daten spezifiziert.

3.3.1 Informationsaustausch beim Verbindungsaufbau

Der in den vorigen Abschnitten für einzelne Sicherheitsdienste gezeigte Austausch von Informationen beim Verbindungsaufbau muß so zusammengefaßt werden, daß für alle Sicherheitsdienste die benötigten Informationen übermittelt werden. Es findet also nicht für jeden Sicherheitsmechanismus nacheinander der entsprechende Datenaustausch statt, sondern es werden in wenigen Segmenten die Parameter für alle Sicherheitsdienste ausgehandelt. Abbildung 3.4 zeigt das gemeinsame Aushandeln aller Sicherheitsdienste.

Partner A			**Partner B**
List(Key Exchange Algorithms)	$\rightarrow$	$\leftarrow$	List(Key Exchange Algorithms)
List(symmetric Algorithms)$_{B\rightarrow A}$			List(symmetric Algorithms)$_{A\rightarrow B}$
List(Signature Algorithms)$_{B\rightarrow A}$			List(Signature Algorithms)$_{A\rightarrow B}$
Choice(Key Exchange Algorithm)	$\rightarrow$	$\leftarrow$	Choice(Key Exchange Algorithm)
Choice(symmetric Algorithm)$_{A\rightarrow B}$			Choice(symmetric Algorithm)$_{B\rightarrow A}$
Choice(Signature Algorithm)$_{A\rightarrow B}$			Choice(Signature Algorithm)$_{B\rightarrow A}$
Certificate$_A$			Certificate$_B$
Key Exchange Values	$\rightarrow$	$\leftarrow$	Key Exchange Values

Abbildung 3.5: Simultaner Informationsaustausch für die Sicherheitsdienste. Die gleichzeitige Auswahl des Schlüsselaustauschverfahrens muß auf beiden Seiten zum selben Ergebnis führen.

Die bisher vorgestellten Abläufe beim Verbindungsaufbau gehen davon aus, daß es eine klare Abgrenzung gibt zwischen demjenigen, der eine Verbindung initiiert (Client) und dem annehmenden Kommunikationspartner (Server). Im normalen TCP-Protokoll gibt es jedoch auch die Möglichkeit, daß zwei Instanzen gleichzeitig aktiv eine Verbindung aufbauen können (sog. *Simultaneous Open*). In der Praxis kommt es jedoch sehr selten zu einem gleichzeitigen Verbindungsaufbau. Gewöhnlich folgt der Kommunikationsablauf dem Client-Server-Modell, höchstens im Bereich der Zeitsynchronisationsprotokolle könnten gleichzeitige Anfragen nach der Uhrzeit der Partnerseite erfolgen.[3] Ein Unterbinden des *Simultaneous Open* kann einfach dadurch geschehen, daß ein Kommunikationspartner, der bereits ein SYN-Segment versendet hat, auf ein Segment mit gesetztem SYN- und gesetztem ACK-Bit wartet. Erhält er nur ein SYN-Segment, kann er durch ein Segment mit RST-Bit die Verbindung unterbrechen.

Andererseits kann versucht werden, auch das Aushandeln der kryptographischen Algorithmen so zu gestalten, daß ein gleichzeitiges Initiieren der Verbindung von beiden Seiten aus möglich ist. Dabei erhalten beide Kommunikationspartner zu Beginn die Auswahllisten der Algorithmen. Da diese ohnehin in symmetrischer Weise ausgetauscht werden (jede Seite versendet eine Liste der Algorithmen für eine Kommunikationsrichtung und erhält eine Auswahl daraus zurück), ergibt sich hier keine Schwierigkeit. Nur die Einigung über das Schlüsselaustauschverfahren muß eindeutig für beide Kommunikationspartner erfolgen. Da beide Seiten zu Beginn bereits die entsprechende Auswahlliste des Gegenübers erhalten, muß sichergestellt werden, daß beide Seiten dasselbe Verfahren auswählen. Zur Auswahl stehen ohnehin nur Verfahren, die sowohl in der gesendeten Liste als auch in der empfangenen Liste vorhanden sind. Als Einigungsalgorithmus kann z.B. die Auswahl der niedrigsten oder der höchsten gemeinsamen Nummer eines Schlüsselaustauschverfahrens dienen. Das Wählen der höchsten Nummer entspricht der Auswahl des zuletzt neu spezifizierten Schlüsselaustauschverfahrens. Unter der Annahme, daß in bestehenden, alten Verfahren evtl. Sicherheitslücken bekannt werden können, ist die Wahl eines neuen Verfahrens vorzuziehen. Abbildung 3.5 zeigt den Informationsaustausch in diesem speziellen Fall.

[3] Ein weiteres Beispiel wäre ein Cache-Update des *Network File Systems*, bei dem jeweils der Gegenüber als untergeordneter (secondary) Cacheserver konfiguriert ist, so daß beide den untergeordneten Server gleichzeitig über Veränderungen informieren wollen.

Erkennung der Partnerinstanz

Die Headerstruktur von TCP bietet mehrere Möglichkeiten, die Signalisierung von vorhandenen Sicherheitsfunktionen beim Verbindungsaufbau auf eine kompatible Weise zu realisieren. Innerhalb des Headers eines TCP-Segments wurden bereits vorsorglich einige Bits reserviert, um zukünftige Erweiterungen des Protokolls zu ermöglichen. Diese Bits könnten genutzt werden, um dem Kommunikationspartner beim Verbindungsaufbau zu signalisieren, daß der Einsatz von Sicherheitsfunktionen möglich und erwünscht ist. Weiterhin gibt es innerhalb des Headers die Möglichkeit, bis zu 40 Bytes zusätzlicher Informationen in das "Options"-Feld einzutragen. Solche Optionen werden bisher von einigen Implementationen genutzt, um das Verhalten von TCP für einen hohen Datendurchsatz über spezielle Übertragungsmedien zu optimieren [Postel 1981, Jacobson 1992, Matthis 1996]. Eine weitere Ergänzung, das sog. *Transaction TCP* (T/TCP), nutzt diese Optionen, um den Verbindungsaufbau abzukürzen und somit einen hohen Transaktionsdurchsatz z.B. bei Datenbankanfragen zu erzielen [Braden 1994].

Die Einträge in den Optionen können von einer TCP-Implementierung ignoriert werden. Ebenso muß davon ausgegangen werden, daß die reservierten Bits nicht von jeder TCP-Implementierung ausgewertet werden. Dies ermöglicht Erweiterungen, bei denen der Client zunächst beim Verbindungsaufbau die reservierten Bits oder die Optionen benutzt, um der Partnerinstanz mitzuteilen, daß er zusätzliche Erweiterungen des TCP-Protokolls beherrscht. Aus der Antwort des Servers geht dann hervor (ebenfalls aus den reservierten Bits oder den Optionen), ob dieser ebenfalls über die Erweiterungen verfügt. Diese Ergänzungen des Headers ermöglichen also eine Erkennung der Partnerinstanz. Da die reservierten Bits nur sehr wenige Möglichkeiten bieten, Informationen auszutauschen, und sie für weitere TCP-Erweiterungen nötig sein könnten, bieten sich die Optionen des TCP-Headers an, um die Identifikation der Partnerinstanz zu ermöglichen.

Auswahl der einzusetzenden Algorithmen

Anstelle der Signalisierung, daß Sicherheitsdienste vorhanden sind, könnte auch gleich eine Bekanntgabe der verfügbaren Algorithmen erfolgen. Hierfür müßte das Datenformat dieser Informationen sehr kurz gehalten werden, um in das Options-Feld zu passen. Dieses Vorgehen würde die Anzahl der benötigten Segmente für den Austausch der kryptographischen Informationen reduzieren. Geht man von einer solchen Option aus, die zunächst einmal die Optionsnummer und die Länge der Option enthält (entsprechend [Braden 1989]), gefolgt jeweils von den Listen der Verfahren für Schlüsselaustausch, Verschlüsselung und Prüfsummenbildung, benötigt diese Option bereits einen Großteil der maximal verfügbaren Bytes des Options-Feldes.

Enthält jede Liste z.B. drei Verfahren, sind jeweils 3 Bytes nötig, zuzüglich der entsprechenden Einträge zur Kennzeichnung des Listenendes. Die Option hätte also eine Länge von 13 Bytes (Typ + Länge + 3 x 3 Algorithmen + 2 Trennzeichen). Die Antwort würde aus einer Folge von Bytes bestehen, die die ausgewählten Algorithmen anzeigen. Weiterhin würde die Antwort auch eine ähnlich lange Folge von Algorithmenlisten enthalten, um die Verschlüsselungs- und Prüfsummenverfahren in die Gegenrichtung

Partner A		**Partner B**
Option(Security available)	→	
	←	Option(Security available) List(Key Exchange Algorithms) List(symmetric Algorithms)$_{A\rightarrow B}$ List(Signature Algorithms)$_{A\rightarrow B}$
Choice(Key Exchange Algorithm) Choice(symmetric Algorithm)$_{A\rightarrow B}$ Choice(Signature Algorithm)$_{A\rightarrow B}$ List(symmetric Algorithms)$_{B\rightarrow A}$ List(Signature Algorithms)$_{B\rightarrow A}$ Certificate$_A$	→	
	←	Choice(symmetric Algorithm)$_{B\rightarrow A}$ Choice(Signature Algorithm)$_{B\rightarrow A}$ Certificate$_B$ Key Exchange Values
Key Exchange Values	→	

Abbildung 3.6: Endgültiger Informationsaustausch für alle Sicherheitsdienste. Zunächst wird über eine TCP-Option nur das Vorhandensein der Sicherheitsdienste signalisiert. Die Sendereihenfolge wird dadurch verändert, so daß der Partner A, der die Verbindung initiiert, das Schlüsselaustauschverfahren auswählt.

auszuwählen. Eine solche Antwort würde also etwa 20 Bytes der TCP-Optionen benötigen. Da bereits einige TCP-Optionen zusätzlich verwendet werden, wären die Optionen für zukünftige Erweiterungen kaum noch zu verwenden (z.B. benutzt die Zeitstempel-Option bereits 10 Bytes [Stevens 1994, S. 253]). Unter diesem Gesichtspunkt bietet es sich an, die Optionen nur für die Erkennung einer Partnerinstanz mit Sicherheitsfunktionen zu nutzen, das Aushandeln der Algorithmen jedoch durch Einträge in den Nutzdaten der Segmente zu realisieren. Dies erhöht zwar die Anzahl der für den Verbindungsaufbau benötigten Segmente, läßt jedoch Raum für zukünftige Erweiterungen des TCP-Protokolls. Abbildung 3.6 zeigt den resultierenden Verbindungsaufbau für das normale, einseitige Initiieren einer Verbindung. Der simultane Verbindungsaufbau aus Abbildung 3.5 wird nur um den gleichzeitigen Austausch der neuen TCP-Option ergänzt, der zur Erkennung der Partnerinstanz nötig ist.

Die Listen von Algorithmen werden auf jedem Endsystem jeweils für einzelne Verbindungen konfiguriert. Beim Verbindungsaufbau werden die zu der Verbindung passenden Listen ausgewählt und übermittelt (siehe auch Abschnitt 3.4.2). Die Listen sollten nicht alle lokal verfügbaren Algorithmen enthalten, sondern nur die, die auch der lokalen Sicherheitspolicy entsprechen. Sonst könnte es vorkommen, daß ein Client eine Liste an den Server übermittelt, dieser sich einen Algorithmus aussucht, und der Client diese Auswahl wiederum ablehnen muß, da sie nicht der Policy genügt.

Es ist denkbar, die Reihenfolge der Algorithmen nach den lokalen Präferenzen anzugeben. Ein Algorithmus, der für eine Verbindung in Frage kommt und für den Unterstützung in Form eines Kryptoprozessors vorliegt, könnte als erstes Element der Liste eingesetzt werden. Der Kommunikationspartner müßte dann den ersten Algorithmus auswählen, der lokal verfügbar ist und gleichzeitig der eigenen Policy genügt. Eine solche Strategie läßt sich jedoch nicht erzwingen, da die Fairneß der Kommunikationspartners bei der Algorithmenauswahl nicht überprüfbar ist. Der Kommunikationspart-

Partner A			**Partner B**
Auswahlliste(a,b,c)$_{B \to A}$	$\rightarrow$		
		$\leftarrow$	Auswahl(c)$_{B \to A}$ Auswahlliste(c,b,a)$_{A \to B}$
Auswahl(c)$_{A \to B}$	$\rightarrow$		

Abbildung 3.7: Unfaires Auswählen bei Präferenzen nach Elementreihenfolge. Der Partner A gibt eine Präferenz für den Algorithmus a an, Partner B wählt jedoch c, obwohl seine eigene Auswahlliste zeigt, daß er auch über Algorithmus a verfügt. Partner B verhält sich also unfair. Dies ließe sich jedoch von Partner A nicht feststellen, wenn in der Auswahlliste von Partner B der Algorithmus a nicht vorhanden wäre.

ner könnte sich unfair verhalten, indem er einen beliebigen Algorithmus auswählt. Dies läßt sich von der Gegenseite nicht feststellen, da i.d.R. nicht bekannt ist, über welche Algorithmen der Kommunikationspartner verfügt. Abbildung 3.7 zeigt ein Beispiel für solch eine unfaire Auswahl. Eine Priorisierung der Listen läßt sich also nicht forcieren, so daß die Reihenfolge der Elemente höchstens einen Hinweis geben kann, jedoch nicht zwingend berücksichtigt werden muß.

Weiterhin sollten die Listen L_A, die von einem Rechner A an einen Kommunikationspartner B versandt werden, die Algorithmen enthalten, die bei der Übertragung vom Rechner B zum Rechner A verwendet werden können. Der Sender von Daten entscheidet somit darüber, mit welchem Algorithmus diese Daten letztendlich verschlüsselt werden. Der Kommunikationspartner B kann aus dieser Liste frei einen Algorithmus wählen.

Wenn ein Kommunikationspartner B eine Liste L_A von Algorithmen erhält, macht er einen Abgleich mit der eigenen Liste L_B von Algorithmen, die für die Verbindung in Frage kommen. Diese Liste muß nicht zwingend dieselbe Liste sein, die dieser Rechner auch als Auswahlliste an den Partner A übermittelt. Auf diese Weise könnte eine Policy umgesetzt werden, die besagt: Für das, was ich sende, genügt Algorithmus x, für das, was mein Gegenüber sendet, genügt dieser Algorithmus jedoch nicht. Algorithmus x wäre also nur in der Liste der Sendealgorithmen vorhanden. Solch eine Asymmetrie könnte z.B. im Falle von Datenbankanfragen sinnvoll sein, bei der in den Anfragen ein Passwort übermittelt wird, welches gut geschützt werden soll. Die Antworten des Servers könnten evtl. keiner oder einer schnelleren Verschlüsselung bedürfen, da dies sonst für den Server eine zu hohe Belastung darstellen könnte. Das Verwenden von einer oder zwei Listen ist jedoch nur eine lokale Implementierungsfrage dar, wichtig für das Protokoll ist nur, daß eine Einigung über die Algorithmen durchgeführt wird und diese auch den Einsatz verschiedener Algorithmen für die beiden Kommunikationsrichtungen ermöglicht.

Austausch kryptographischer Informationen

Sind die Algorithmen ausgewählt, müssen entsprechende kryptographische Informationen ausgetauscht werden.

Die maximale Länge der Optionen von 40 Bytes macht sie ungeeignet, kryptographische

Informationen zu transportieren. Dies bedeutet, daß das erste SYN-Segment, das den Verbindungsaufbau initiiert, z.B. nicht gleich ein Zertifikat des Absenders enthalten kann, um sog. “SYN-Flooding-Angriffe” zu unterbinden [Schuba 1997] Für Zertifikate oder Schlüssel eines asymmetrischen Verfahrens sind 320 Bits ohnehin nicht mehr ausreichend. Doch auch bei Schlüsseln von symmetrischen Verfahren oder bei Hashwerten verursacht die Begrenzung auf 320 Bits Schwierigkeiten, zumal die Schlüssellängen auf Grund der zunehmenden Rechenleistung der Hardware stets wachsen müssen, um gleichbleibende Sicherheit zu garantieren. I.d.R. müssen mehrere solcher Informationen beim Verbindungsaufbau ausgetauscht werden, da z.B. jeder Schlüssel mit einer Signatur versehen werden muß, um “Man in the Middle”-Angriffen vorzubeugen. Weitere Informationen, die in einem Zertifikat enthalten sein können, wie z.B ein Verweis auf die ausstellende CA oder Angaben über das Verschlüsselungsverfahren, für das der Schlüssel geeignet ist, benötigen zusätzlichen Speicherplatz. Die TCP-Optionen sind daher ebenfalls nicht geeignet, diese Informationen aufzunehmen.

Aus den vorhergehenden Abschnitten geht hervor, daß nur die Erkennung von vorhandenen Sicherheitsdiensten über Einträge in den Optionen realisiert werden sollten. Diese Einträge werden jeweils nur beim Verbindungsaufbau benötigt. Die Spezifikation der Formate dieser Optionen erfolgt im nächsten Abschnitt. Die weiteren Informationen, die zum Aushandeln der Algorithmen und zum Austausch von Schlüsseln benötigt werden, können dann innerhalb der Nutzdaten der TCP-Segmente übertragen werden. Hierfür müssen diese gekennzeichnet werden, damit sie von den Daten der Anwendung unterschieden werden können.

Diese Einträge in den Nutzdaten werden mehrere Funktionen erfüllen. Einige finden nur zur einmaligen Übertragung der Zertifikate beim Verbindungsaufbau Verwendung. Andere enthalten Schlüsselaustauschinformationen und werden sowohl beim Verbindungsaufbau als auch beim erneuten Schlüsselaustausch im Falle von langanhaltenden Verbindungen benutzt. Wieder andere werden in jedem Segment nach erfolgtem Verbindungsaufbau übertragen, da sie jeweils die Prüfsummen für das Segment enthalten. Die entsprechenden Datenformate werden in Abschnitt 3.3.4 entworfen.

3.3.2 Änderungen am TCP Zustandstransitionsdiagramm

Die tiefgreifenden Änderungen am Verbindungsaufbau führen zu einer starken Erweiterung des entsprechenden Protokollautomaten. Abbildung 3.8 zeigt noch einmal den Teil des Zustandstransitionsdiagramms, der den Verbindungsaufbau beschreibt.

Für den Verbindungsaufbau unter Verwendung der Sicherheitsdienste wird der Protokollgraph für den normalen Verbindungsaufbau verändert. Abbildung 3.9 zeigt das geänderte Zustandstransitionsdiagramm. Bei Einsatz der Sicherheitsdienste wird jeweils in einem SYN-Segment die Sicherheitsoption (in der Abbildung mit `SECU` bezeichnet) mitgesendet. Entsprechend wurde eine neuer Zustand `SYN_SECU_SENT` eingeführt, der sich vom normalen `SYN_SENT` nur durch die gesendete Sicherheits-Option unterscheidet.

Für jede Verbindung muß nun geprüft werden, ob sie ohne Verwendung der Sicherheitsdienste zustande kommen darf. Zur Überprüfung stehen die IP-Adressen und die Portnummern von Absender und Empfänger zur Verfügung. Wird von der Gegenseite nicht

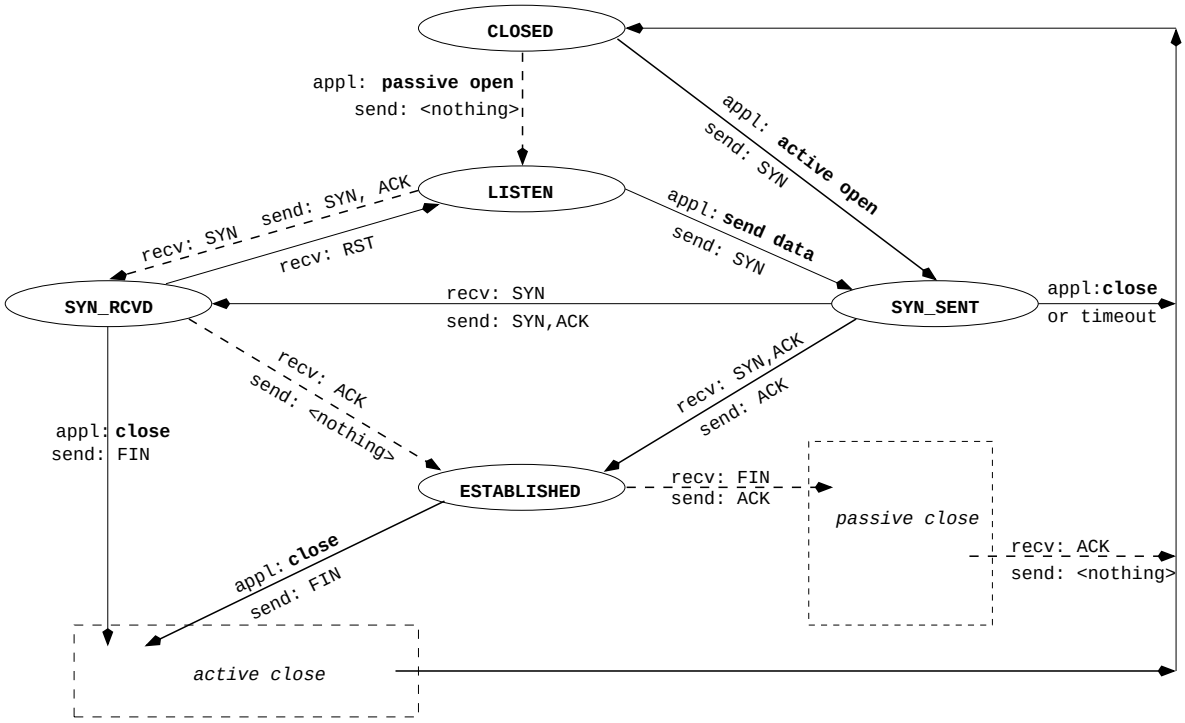


Abbildung 3.8: Zustandstransitionsdiagramm von TCP (aus [Stevens 1994]). Der Abbau der Verbindung wird hier nur angedeutet (gestrichelte Boxen).

durch die Sicherheits-Option das Vorhandensein von Sicherheitsdiensten signalisiert, muß der Verbindungsaufbau gegebenenfalls abgebrochen werden. Dies geschieht z.B. beim Empfang eines SYN-Segmentes ohne Sicherheits-Option im Zustand `LISTEN`. Sind Sicherheitsdienste erforderlich, wird ein RST-Segment zurückgesendet. Das lokale TCP verbleibt im Zustand `LISTEN`. Der Zustand `Listen` erhält also eine Transition, die zum selben Zustand zurückführt. Ebenso wird die Verbindung abgebrochen, wenn auf eine gesendete Sicherheitsoption (Zustand `SYN_SECU_SENT`) die Option in der Antwort nicht vorhanden ist, Sicherheitsdienste jedoch erforderlich sind. Werden Sicherheitsdienste nicht benötigt, dann erfolgt bei Empfang eines Segmentes ohne Sicherheitsoption ein Übergang zu den Zuständen des normalen TCP. Die entsprechenden Transitionen sind in Abbildung 3.9 gestrichelt dargestellt.

Wird die Sicherheits-Option vom Client mitgesendet, so antwortet der Server mit einem Segment, das sowohl die TCP-Option für Sicherheitsdienste als auch bereits die Listen der verfügbaren und gewünschten Algorithmen enthält. Der weitere Ablauf des Verbindungsaufbaus ist aus Abbildung 3.9 ersichtlich. Die empfangenen und gesendeten Informationen entsprechen dem in Abbildung 3.6 gezeigten Informationsaustausch.

Der simultane Verbindungsaufbau erfordert weitere Zustände, da hierbei die Aufteilung der kryptographischen Informationen auf die einzelnen Segmente verändert wird. So enthält das zweite Segment nur die Listen der verfügbaren Algorithmen, die Auswahl der Verfahren wird jeweils im dritten Segment mitgesendet. Abbildung 3.10 zeigt den entsprechend geänderten Teil des Protokollautomaten. Der daraus resultierende

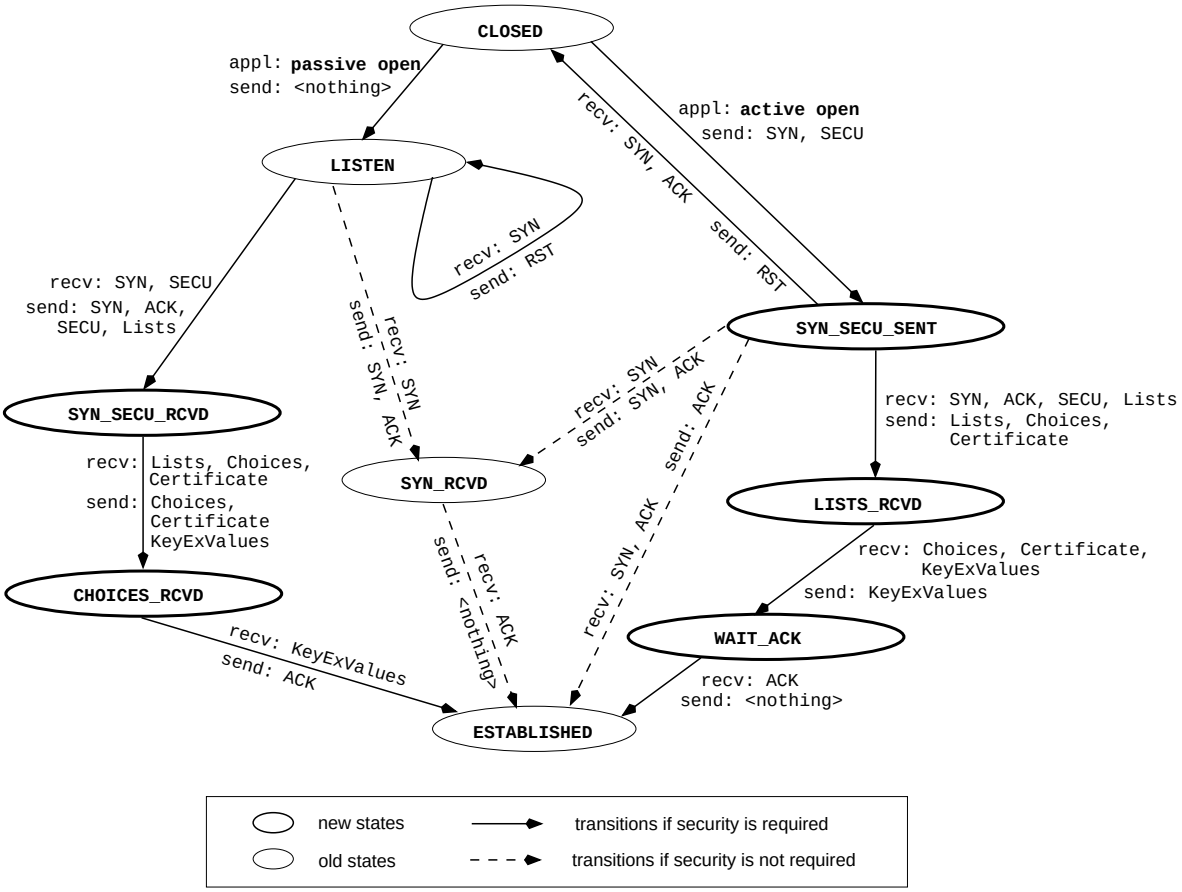


Abbildung 3.9: Zustandstransitionsdiagramm des geänderten Verbindungsaufbaus. Die Transitionen zum Abbau der Verbindung (z.B. beim Systemaufruf `close()` aus der Anwendung) wurden der Übersichtlichkeit halber nicht dargestellt.

Informationsaustausch entspricht dem in Abbildung 3.5 gezeigten Ablauf.

Der Übergang vom Zustand `SYN_SECU_SENT` zurück in den Zustand `CLOSE` wird auch beim Empfang eines SYN-Segmentes (also ohne gesetztes ACK-Bit, d.h beim simultanen Verbindungsaufbau) ohne Sicherheits-Option erfolgen. Das ACK-Bit muß hier also nicht mehr ausgewertet werden (vergl. Abbildung 3.9).

Für jeden der neuen Zustände wird auf den Empfang eines unerwarteten Segmentes (also z.B. ein Segment mit Algorithmenlisten im Zustand `CHOICES_SENT`) der Verbindungsaufbau durch ein RST-Segment abgebrochen. Diese Strategie ist sinnvoll, da bei einem Segment, das eine gültige Sequenznummer, jedoch nicht protokollgerechten Inhalt trägt, von einem Angriff ausgegangen werden kann. Die entsprechenden Transitionen zum Abbruch der Verbindung wurden der Übersichtlichkeit halber nicht eingezeichnet.

3.3.3 Formate der neuen Optionen

Über Einträge in den TCP-Optionen wird dem Kommunikationspartner signalisiert, daß

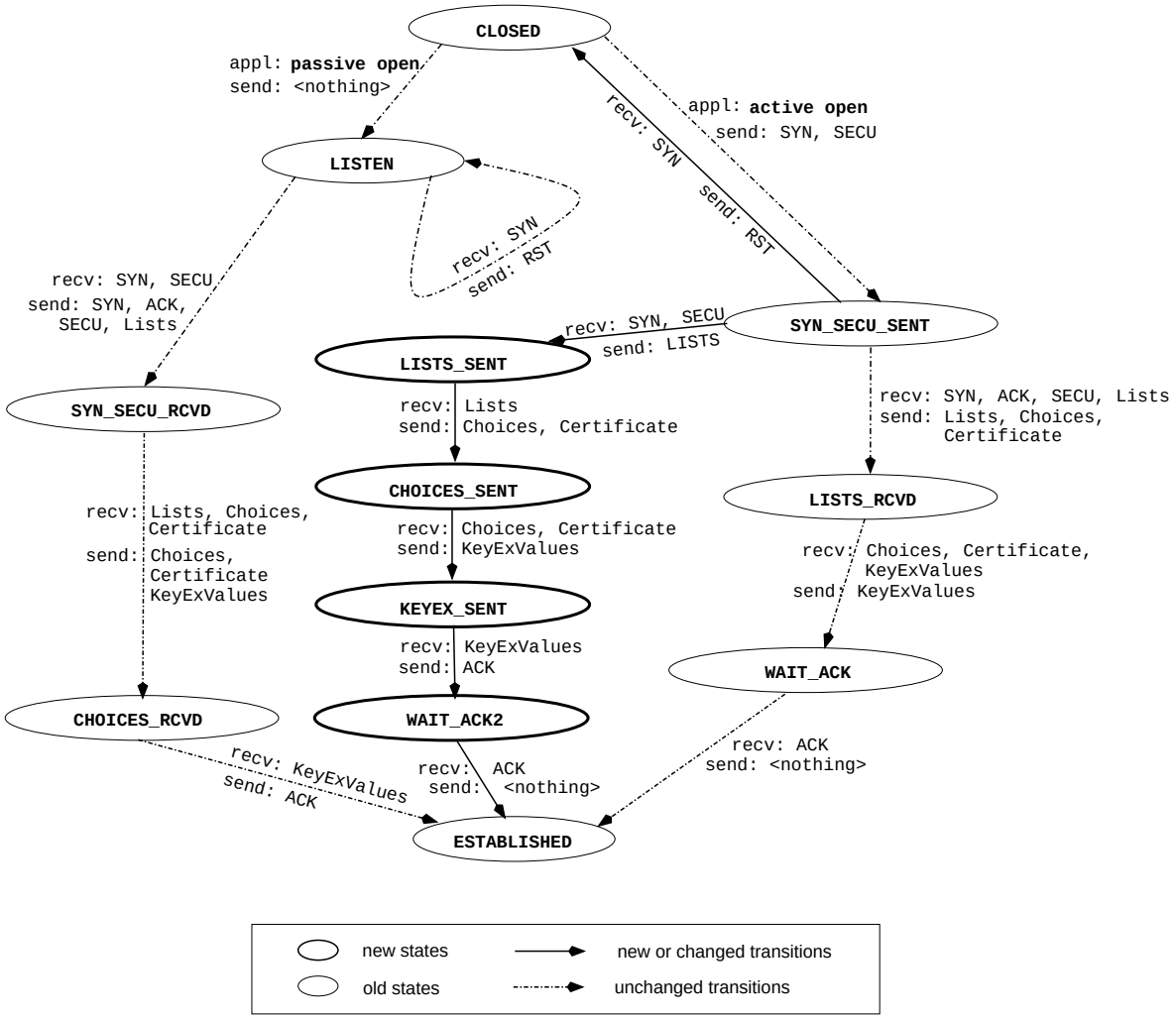


Abbildung 3.10: Änderungen des Zustandstransitionsdiagramms für “Simultaneous Open”. Die Zustände für den normalen Verbindungsaufbau ohne Sicherheitsdienste wurden zur besseren Übersichtlichkeit ausgelassen.

Sicherheitsdienste eingesetzt werden können. Hierzu genügt eine Option, die als Signal dient und keine weiteren Daten beinhaltet. Abbildung 3.11 zeigt die Struktur der neuen Option.

Eingeleitet wird die Option von der Typ-Nummer 20,[4] gefolgt von der Längenangabe der Option. Die Längenangabe ist notwendig, um TCP-Implementationen, die diese Option nicht kennen, die weitere Verarbeitung der folgenden Optionen zu ermöglichen [Braden 1989]. Auf den Erhalt dieser Option antwortet die Partnerinstanz mit einem Segment, in dem diese Option ebenfalls vorhanden ist. Dieses Segment kann bereits Einträge in den Nutzdaten enthalten, die zum Aushandeln der Sicherheitsparameter dienen, da sichergestellt ist, daß der Kommunikationspartner über Sicherheitsdienste

[4]Die Typ-Nummern 0 – 5, 8 und 11 – 13 sind bereits vergeben [Stevens 1994, Matthis 1996, Braden 1994].

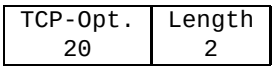


Abbildung 3.11: Format der TCP-Option, die das Vorhandensein von Sicherheitsdiensten signalisiert.

verfügt.

3.3.4 Formate der Einträge in die Nutzdaten

Nach der Signalisierung der vorhandenen Sicherheitsdienste werden die einzusetzenden Algorithmen zwischen den Partnerinstanzen ausgehandelt. Dies sollte in einem Schritt für alle Sicherheitsdienste erfolgen, um nicht unnötig viele Segmente hierfür auszutauschen. Es muß hierbei sichergestellt werden, daß die später auszutauschenden Zertifikate auch zu den ausgewählten Algorithmen passen. Der enthaltene Schlüssel muß also z.B. im Falle des signierten Diffie-Hellman-Schlüsselaustauschs zum Signieren geeignet sein. Über folgende Eigenschaften muß beim Verbindungsaufbau eine Einigung erzielt werden:

- Verschlüsselungsverfahren für die Richtung von A zu B.
- Verschlüsselungsverfahren für die Richtung von B zu A.
- Signaturverfahren für die Richtung von A zu B.
- Signaturverfahren für die Richtung von B zu A.
- Zu benutzendes Schlüsselaustauschverfahren.
- Typ des Zertifikates, das von A an B übertragen wird.
- Typ des Zertifikates, das von B an A übertragen wird.

Jeweils ein Kommunikationspartner muß also zunächst die Information übertragen, über welche Verfahren zur Verschlüsselung und Signierung er verfügt, welche Zertifikatstypen von ihm unterstützt werden und welche Schlüsselaustauschverfahren er beherrscht. Für jede Auswahlmöglichkeit wird eine Liste der verfügbaren Verfahren bzw. Formate gebildet, die in der Reihenfolge der gewünschten Priorität geordnet ist. Somit sollte vom Gegenüber der erste Eintrag ausgewählt werden, der auf Grund der unterstützten Verfahren und der eigenen Policy akzeptabel ist.

Eine statische Aufteilung des Nutzdatenbereichs eines TCP-Segmentes in entsprechende Einträge ist hierbei nicht möglich. Einerseits ist die Anzahl der benötigten Felder von den verwendeten Sicherheitsdiensten abhängig. Wird keine Integrität und Authentizität benötigt, muß auch keine entsprechende Prüfsumme übertragen werden. Andererseits variiert die Länge der einzelnen Felder ohnehin in Abhängigkeit von den ausgewählten Algorithmen, da z.B. verschieden Hashalgorithmen unterschiedlich lange Prüfsummen verwenden.

Daher wird das folgende Format für die Einträge definiert: Jeder benötigte Eintrag in die Nutzdaten wird zunächst durch ein Feld eingeleitet, das den Typ oder Zweck des Eintrags angibt.[5] Dies hat den Vorteil, daß jede Nachricht für sich interpretiert werden kann, ohne zusätzliche Informationen auszuwerten, die nicht in der Nachricht selbst stehen.[6] Darauf folgt eine Längenangabe des Inhaltes, um somit die Verarbeitung des nächsten Eintrags zu ermöglichen. Es handelt sich bei dieser Angabe um die Gesamtlänge des Eintrags in Bytes. Hierauf folgen evtl. benötigte Daten des Eintrags, wie z.B. die enthaltene Algorithmenliste.

Für den Verbindungsaufbau werden zunächst Einträge benötigt, die das Aushandeln des Schlüsselaustauschverfahrens, der Verschlüsselungsalgorithmen und der Prüfsummenalgorithmen ermöglichen. Dabei werden jeweils zwei Einträge benötigt. Einer beinhaltet die Auswahllisten der vorhandenen Algorithmen, der zweite beinhaltet als Antwort den ausgewählten Algorithmus. Zusätzlich werden Einträge benötigt, die das kryptographische Material beinhalten. Für jede Art von Zertifikat und für jeden Wert eines Schlüsselaustauschverfahrens muß ein solcher Eintrag spezifiziert werden. Auch für die in jedem Segment enthaltenen Nutzdaten und die Prüfsumme (MAC) muß ein Eintrag existieren. Ein weitere Eintrag wird Auffüllbytes kennzeichnen und deren Anzahl angeben. Diese können nötig sein, um die Anzahl der zu verschlüsselnden Bytes auf die Blocklänge des Verschlüsselungsalgorithmus aufzufüllen. Abbildung 3.12 zeigt die verschiedenen Einträge und deren Struktur.

Auch die Zuordnungen der Nummern zu den Algorithmen muß einheitlich für alle Implementierungen vergeben werden, wie dies z.B. im Falle der Portnummern (sog. *Assigned Numbers*) bereits durch die IANA (Internet Assigned Numbers Authority) geschieht [Reinolds 1994]. Für die Implementierung, die im Rahmen dieser Arbeit vorgenommen wurde, wird zunächst nur eine minimale Menge an Algorithmen verwendet. Es kommen jeweils nur ein Schlüsselaustauschverfahren, ein Verschlüsselungsverfahren und ein Prüfsummenalgorithmus zum Einsatz. Weiterhin wird für die Verschlüsselung und die Prüfsummenbildung jeweils das sog. "Nullverfahren" definiert, bei dessen Einsatz keine Verschlüsselung bzw. keine Prüfsummenbildung vorgenommen wird. Für die Darstellung eines Algorithmus wird ein Byte reserviert, so daß jeweils bis zu 255 Algorithmen implementiert werden könnten. Ein Nullbyte beendet die kryptographischen Daten dieses Eintrags und kann somit als Füllzeichen verwendet werden. Der Eintrag 1 kennzeichnet das Nullverfahren. Abbildung 3.13 zeigt die in der Implementierung eingesetzten Verfahren und ihre Kennungen.

3.4 Struktur des Systems

Zusätzlich zum Protokoll selbst müssen auch noch die Schnittstellen und die Konfigurationsdateien definiert werden. Die Integration der Sicherheitsdienste in die Transportschicht eines Unix-Systems ermöglicht es, von den Anwendungen aus die Sicherheitsparameter abzufragen und zu verändern, da sich keine weitere Schicht zwischen

[5] Im Englischen bezeichnet man diese Daten als "tagged", d.h. als mit einem Etikett versehen.

[6] Ist z.B. der Zweck einer Nachricht abhängig von dem Zeitpunkt des Empfangs, so läßt sich der Protokollablauf durch ein wiederholtes Abspielen einer aufgezeichneten Nachricht beeinflussen. Jede Nachricht sollte Kennzeichen enthalten, die die korrekte Interpretation ermöglichen [Abadí 1994].

Eintrag	**Struktur**					
Liste von Schlüsselaustauschverfahren	Identifier 01 01	Länge xx xx	Verfahren 1	Verfahren 2	...	Verfahren n
Auswahl eines Schlüsselaustauschverfahrens	Identifier 02 02	Länge 00 05	Verfahren			
Liste von Verschlüsselungsverfahren	Identifier 03 01	Länge xx xx	Verfahren 1	Verfahren 2	...	Verfahren n
Auswahl eines Verschlüsselungsverfahrens	Identifier 04 02	Länge 00 05	Verfahren			
Liste von Prüfsummenverfahren	Identifier 05 01	Länge xx xx	Verfahren 1	Verfahren 2	...	Verfahren n
Auswahl eines Prüfsummenverfahrens	Identifier 06 02	Länge 00 05	Verfahren			
Zertifikat	Identifier 07 01	Länge xx xx	Zertifikat			
Wert eines Schlüsselaustauschverfahrens	Identifier 08 <Typ>	Länge xx xx	Wert			
Nutzdaten	Identifier 09 01	Länge xx xx	Nutzdaten			
Füllbytes	Identifier 0a 01	Länge xx xx	Füllbytes			
Prüfsumme des Segments	Identifier 0b <Typ>	Länge xx xx	MAC			

Abbildung 3.12: Übersicht über die verschiedenen Einträge in die Nutzdaten der Segmente

Transportschicht und Anwendungen befindet. Die Systemfunktionen `getsockopt` und `setsockopt` können zu diesem Zweck erweitert werden, so daß sie zusätzliche Informationen über die für die zugehörige Verbindung geltenden Sicherheitseinstellungen liefern und diese Parameter verändern.

Weiterhin müssen die Sicherheitsdienste konfigurierbar sein, damit die TCP-Verbindungen von solchen Anwendungen, die keine Einstellung der Sicherheitsdienste vornehmen, voreingestellte Sicherheitsmechanismen verwenden. Die Konfiguration sollte mittels entsprechender Dateien sowohl global vom Systemadministrator als auch zusätzlich von einzelnen Benutzern des Systems vorgenommen werden können.

3.4.1 Schnittstellenbeschreibung

Anwendungen, die für die Kommunikation das TCP-Protokoll verwenden, benutzen weiterhin die hierfür bereitgestellten Systemaufrufe (z.B. `open`, `close`, `connect`, `accept`, `read` und `write`). Beim Aufbau einer TCP-Verbindung werden die vorkonfigurierten Sicherheitsmechanismen eingesetzt. Über den Systemaufruf `setsockopt` können vor dem Verbindungsaufbau einige Sicherheitseinstellungen von den Anwendungen verändert werden. Hierzu werden die vorhandenen Socketoptionen um einige

Mechanismus	Algorithmus	Kennziffer
Schlüsselaustausch	Diffie-Hellman	01
Verschlüsselung	Nullverfahren	01
	DES	02
Prüfsummen	Nullverfahren	01
	HMAC-MD5	02

Abbildung 3.13: Die implementierten Verfahren und deren Kennungen

Name der Option	Bedeutung	mögliche Werte
TCP_KEYEXCHOICE	verwendetes Schlüsselaustausch-verfahren	0 - 255
TCP_KEYEXLIST	Priorität der Schlüsselaus-tauschverfahren	Folge von Bytes
TCP_ENCINCHOICE	verwendeter Verschlüsselungsal-gorithmus für eingehende Daten	0 - 255
TCP_ENCOUTCHOICE	verwendeter Verschlüsselungsal-gorithmus für ausgehende Daten	0 - 255
TCP_ENCRYPTLIST	Liste der erlaubten Ver-schlüsselungsalgorithmen für diese Verbindung	Folge von Bytes
TCP_MACINCHOICE	verwendeter Signaturalgorithmus für eingehende Daten	0 - 255
TCP_MACOUTCHOICE	verwendeter Signaturalgorithmus für ausgehende Daten	0 - 255
TCP_SIGNLIST	Liste der erlaubten Signatural-gorithmen für diese Verbindung	Folge von Bytes

Abbildung 3.14: Die neu eingeführten Socketoptionen

sicherheitsspezifische Einträge ergänzt. Diese Ergänzungen ermöglichen ein Abfragen und Verändern der Algorithmen, die für die Verbindung genutzt werden sollen. Abbildung 3.14 zeigt die neu eingeführten Optionen.

Für Verschlüsselung, Signatur und Schlüsselaustauschverfahren lassen sich mittels `getsockopt` die für eine bestehende Verbindung gewählten Verfahren erfragen. Dies geschieht über die Optionen, deren Namen die Endung `CHOICE` besitzen. Z.B. liefert der Aufruf `getsockopt(TCP_KEYEXCHOICE)` die Nummer des verwendeten Schlüsselaustauschverfahrens zurück. Diese Optionen lassen sich nicht mittels `setsockopt` verändern, da sie nur das Ergebnis der erfolgten Einigung über die Algorithmen wiedergeben. Die Abfrage dieser Werte ist natürlich erst nach dem erfolgten Verbindungsaufbau sinnvoll. Vorher liefert der entsprechende Aufruf den Wert 0 zurück. Die anderen Optionen (mit der Endung `LIST`) können vor dem Aufbau der Verbindung mittels `setsockopt` gesetzt werden. Übergeben wird hierbei die Liste der Algorithmen, die beim Verbindungsaufbau übertragen werden soll. Ein `getsockopt` liefert die Liste der aktuellen Vorgabewerte. Diese Liste wird im Folgenden als Anwendungsliste bezeichnet. Erst während des Verbindungsaufbaus können die wirklich zu verwendenden Listen ermittelt werden, da diese auch von der IP-Adresse und der Portnummer des Kommunikationspartners abhängig sind. Wenn von beiden Partnern die Adresse und Portnummer bekannt sind, wird die Anwendungsliste mit der vom Administrator vorgegebenen

Liste (Administratorliste) zusammengeführt. Dies geschieht, indem aus beiden Listen eine neue generiert wird. Diese enthält nur jene Algorithmen der Anwendungsliste, die auch in der Administratorliste vorhanden sind. Die Reihenfolge der verbleibenden Elemente der Anwendungsliste bleibt erhalten. Setzt z.B. die Anwendung als Liste der Verschlüsselungsverfahren die Liste (5,2,4,1) und wird als zugehörige Liste beim Verbindungsaufbau die Liste (4,5) ermittelt, so resultiert daraus die Liste (5,4). Diese wird beim Verbindungsaufbau an den Kommunikationspartner übertragen. Die Anwendung erhält auf diese Art und Weise die Möglichkeit, eigene Vorgaben zu setzen, diese werden aber auf die vom Administrator zugelassenen Werte beschränkt. Gibt es keine Algorithmen, die sowohl von der Anwendung als auch vom Administrator ausgewählt wurden, kommt eine Verbindung nicht zustande. Dieses Zusammenführen der Vorgaben von Administrator und Anwendung ist bereits die Umsetzung einer sinnvollen Policy, vorstellbar sind natürlich auch andere Policy-Entscheidungen (so daß z.B. die Konfiguration des Administrators über die Reihenfolge der Algorithmen entscheidet, oder die Administratorkonfiguration nur dann verwendet werden, wenn die Anwendung keine Vorgaben macht).

3.4.2 Konfigurationsdateien

Um die Sicherheitsmechanismen zu bestimmen, die bei neuen Verbindungen Verwendung finden sollen, werden entsprechende Vorgaben in Konfigurationsdateien abgelegt. Um einer ausgehenden Verbindung die Standardwerte für die Sicherheitsmechanismen zuzuordnen, enthalten diese Dateien für IP-Adressen und Portnummern des Absenders (lokaler Rechner) und des jeweiligen Empfängers die zu verwendenden Algorithmen. Damit nicht für jede Zieladresse und jeden Dienst einzelne Einträge in die Konfigurationsdateien notwendig sind, können die IP-Adressen und Portnummern durch Wertebereiche vorgegeben werden. Eine Adressangabe `134.100.*.*` bezeichnet dabei alle Zielrechner, deren IP-Adresse mit `134.100` beginnt. Weiterhin können Intervalle vorgegeben werden, um eine differenziertere Konfiguration zu erlauben. Durch `134.100.*.*/0-1023` werden die ersten 1024 Portnummern aller Rechner in der Domain `134.100` konfiguriert. Diesen Angaben, die eine Verbindung kennzeichnen, folgen dann die Vorgaben der einzusetzenden Sicherheitsdienste. Es existiert jeweils eine eigene Konfigurationsdatei für eingehende und für ausgehende Verbindungen. Sinnvolle Einträge für die Konfiguration für ausgehende Verbindungen wären z.B. die folgenden:

```
134.100.13.244:*  134.100.*:23       exclist=01 enclist=02 maclist=02
134.100.13.244:*  134.100.*:80       nosecu exclist=01 enclist=02,01 maclist=02
134.100.13.244:*  134.100.8.63:22    nosecu exclist=01 enclist=02,01 maclist=02
```

Die Einträge besitzen als lokale Adresse jeweils die IP-Adresse des lokalen Rechners. Die Portnummer wird nicht spezifiziert, da die Clientanwendungen, die gewöhnlich einen Verbindungsaufbau initiieren, i.d.R. dynamisch eine freie Portnummer vom Betriebssystem zugeteilt bekommen. Der erste Eintrag gilt also für alle Verbindungen vom lokalen Rechner (IP-Adresse 134.100.13.244), die zu Rechnern innerhalb der Domain 134.100 auf Port 23 hergestellt werden sollen. Port 23 kennzeichnet für gewöhnlich den Dienst `telnet`. Als Schlüsselaustauschverfahren wird Diffie-Hellman gewählt, zur Verschlüsselung der Algorithmus DES und für die Signatur HMAC-MD5. Beherrscht der Zielrechner einen dieser Algorithmen nicht, so kommt die Verbindung nicht zustande.

Dies ist in der Praxis sinnvoll, da im Telnet-Protokoll Benutzername und Passwort im Klartext übermittelt werden.

Der zweite Eintrag gilt wiederum für alle ausgehenden Verbindungen des lokalen Rechners, die zu einem Rechner innerhalb der Domäne 134.100 gehen, diesmal jedoch auf den Zielport 80. Dies ist der Dienst `http`. Der Eintrag `nosecu` signalisiert, daß für diese Verbindungen keine Sicherheitsdienste notwendig sind. Die Verbindung kommt also auch dann zustande, wenn der Zielrechner keine Sicherheitsdienste im TCP-Protokoll unterstützt. Sind auf dem Zielrechner entsprechende Sicherheitsdienste vorhanden, so kommt vorrangig DES als Verschlüsselungsverfahren zum Einsatz, auch das Null-Verfahren (also keine Verschlüsselung) ist möglich.

Der letzte Eintrag gilt für alle Verbindungen vom lokalen Rechner zum Server 134.100.9.63 auf den Port 22 (Dienst `ssh`). Da `ssh` (= Secure Shell) bereits auf Anwendungsebene Verschlüsselung und Authentisierung unterstützt, sind auch hier keine Sicherheitsdienste zwingend erforderlich.

Auf die gleiche Weise können Sicherheitsdienste für eingehende Verbindungen in einer zweiten Datei konfiguriert werden. Hierbei werden i.d.R. für laufende Serverprozesse, die auf festen Ports auf eingehende Verbindungen warten, Sicherheitsvorgaben gemacht. Zwei sinnvolle Eintrage wären z.B. die folgenden:

```
134.100.13.244:23  *:*    exclist=01 enclist=02 maclist=02
134.100.13.244:22  *:*    nosecu exclist=01 enclist=02,01 maclist=02,01
```

Der erste Eintrag erlaubt eingehende Telnet-Verbindungen nur, wenn DES-Verschlüsselung und HMAC-MD5-Signaturen verwendet werden. Der zweite gibt für eingehende SSH-Verbindungen keine Sicherheitsdienste zwingend vor, auch kann bei vorhandenen Sicherheitsdiensten das Nullverfahren bei Verschlüsselung und Signatur ausgehandelt werden.

Kapitel 4

Implementation des sicheren TCP

In diesem Kapitel wird die Implementierung der Sicherheitsdienste beschrieben. Zunächst erfolgt eine Darstellung der Auswahl des Betriebssystems und der hierfür ausschlaggebenden Kriterien. Darauf folgt eine kurze Beschreibung der Struktur der bestehenden TCP-Implementierung. Die notwendigen Änderungen und die hinzugefügten Neuerungen werden in Abschnitt 4.3 vorgestellt. Abschließend werden die verwendeten Werkzeuge beschrieben, die zum Testen der Implementierung eingesetzt wurden.

4.1 Auswahl des Betriebssystems

Bei der Auswahl des Betriebssystems ist zunächst entscheidend, daß der Quellcode der TCP/IP-Implementierung verfügbar sein muß, um Änderungen daran vornehmen zu können. Hierfür kommen im wesentlichen die freien Unix-Derivate (FreeBSD, OpenBSD, NetBSD, Linux) in Frage. Auch das Betriebssystem Solaris erfüllt diese Bedingung, da der Quellcode dem Autor der vorliegenden Arbeit im Rahmen eines *Non-Disclosure-Agreements* zur Verfügung steht. Dies bedeutet jedoch trotzdem eine starke Einschränkung, da die Veröffentlichung der erfolgten Implementierung sich schwer mit der Geheimhaltung der Solaris-Quellcodes vereinbaren läßt.

Ein weiteres Kriterium für die Betriebssystemauswahl ist die verfügbare Dokumentation des Netzwerk-Quellcodes. Im Falle von Linux ist, nicht zuletzt auf Grund der raschen Weiterentwicklung des Systems, die erhältliche Dokumentation nur sehr spärlich. Der Netzwerkcode der BSD-Derivate ist hingegen außergewöhnlich gut dokumentiert [Stevens 1995]. Die verfügbare Dokumentation der Quellcodes von Solaris beschränkt sich auf die in den Dateien angebrachten Kommentare.

Die TCP-Implementierung von Solaris weist im Gegensatz zu den anderen genannten Systemen den Vorteil auf, aus einem Modul zu bestehen, das bei Bedarf dynamisch geladen werden kann. Bei den anderen Systemen muß jeweils ein neuer Kernel generiert werden, mit dem das System neu gestartet werden muß. Dieses Vorgehen erhöht zum einen die Dauer der Entwicklungszyklen (Edit-Compile-Run), da jeweils ein Systemneustart notwendig ist, um vorgenommene Änderungen zu testen. Das Modulkonzept von Solaris ermöglicht zusätzlich, die Sicherheitsdienste als ein Modul verfügbar zu machen, das nur noch zur Laufzeit gegen das Original ausgetauscht werden muß.

Die kaum vorhandene Dokumentation und die Tatsache, daß das TCP-Modul von Solaris aus einer einzigen, ca. 350 Kilobyte großen Quelldatei besteht, die folglich recht unübersichtlich ist, führten jedoch zu einer Entscheidung zu Gunsten der BSD-Derivate. Die drei frei verfügbaren BSD-Derivate FreeBSD, NetBSD und OpenBSD unterscheiden sich stark in den Implementierungen der Internetprotokolle. Die TCP-Implementierung von NetBSD entspricht noch weitestgehend dem Stand der gut dokumentierten 4.4-BSD-Quellen. Zudem sind keine wesentlichen Erweiterungen des TCP-Protokolls vorgenommen worden, die die Integration von Sicherheitsdiensten erschweren könnten. Es erwies sich jedoch als schwierig, ein NetBSD zusammen mit einer vorhandenen Linux-Installation auf einer Festplatte zu installieren, die über eine große Anzahl von Zylindern verfügt. Zudem bietet NetBSD keine komfortable Entwicklungsoberfläche wie z.B. die Verwendung virtueller Konsolen an.

Da bei der Entwicklung von OpenBSD starkes Gewicht auf die Integration von Sicherheitsmechanismen gelegt wird, ist in die TCP-Quellen z.B. bereits eine Möglichkeit integriert, TCP-Segmente mit einem MD5-Hashwert zu signieren [Heffernan 1998]. Die TCP-Optionen könnten bei Verwendung dieses Verfahrens bereits ohne die zu integrierenden Sicherheitsdienste vollständig belegt werden, ein Zusammenspiel beider Dienste somit unmöglich sein.

Daher fiel die Auswahl des Betriebssystems auf FreeBSD in der Version 3.4. Diese pragmatische Entscheidung ist jedoch nicht unproblematisch, da FreeBSD bereits über einige Erweiterungen von TCP verfügt. Insbesondere das Zusammenspiel zwischen Transaction-TCP [Braden 1994] und den zu integrierenden Sicherheitsdiensten führt zu einigen Schwierigkeiten (siehe Abschnitt 4.3).

4.2 Struktur der bestehenden TCP-Implementierung

Die Implementierung der Internetprotokolle IP, TCP und UDP befindet sich im Verzeichnis `netinet` der Betriebssystem-Quellen. Die zu TCP gehörigen Dateien tragen jeweils einen Präfix "`tcp_`" in ihrem Dateinamen. Tabelle 4.1 zeigt die einzelnen Dateien und beschreibt kurz ihre hauptsächliche Funktion.

Der normale Betrieb von TCP wird im wesentlichen durch die Module `tcp_input.c`, `tcp_output.c`, `tcp_timer.c` und `tcp_usrreq.c` realisiert. `tcp_input.c` enthält die Funktion `tcp_input()`, die jeweils dann aufgerufen wird, wenn das IP-Protokoll ein Paket empfangen und dieses als TCP-Segment identifiziert hat. Zum empfangenen Segment wird die zugehörige Verbindung gesucht und die Daten des Segmentes werden in Abhängigkeit vom Zustand der Verbindung und der Sequenznummer in die Empfangswarteschlange des zugehörigen Sockets einsortiert. Wenn die Anwendung auf neue Daten wartet (z.B. beim Systemaufruf `recv()`), wird sie gegebenenfalls wieder in den ausführbaren Zustand versetzt. Beim Auf- und Abbau einer Verbindung wird jeweils ihr Zustand entsprechend dem empfangenen Segment geändert. Befindet sich die Verbindung im Zustand `LISTEN`, so wird beim Empfang eines Segmentes mit gesetztem SYN-Bit der Zustand auf `SYN_RECEIVED` gesetzt. Es folgt ein Aufruf der Funktion `tcp_output()`, um das erforderliche Antwortsegment (mit gesetztem SYN- und ACK-Bit) zu generieren.

`tcp.h`	Deklaration des TCP-Segmentkopfes, der TCP-Optionen, der Socketoptionen und einiger Konstanten.
`tcp_debug.c`	Der Code zur Ausgabe von Debug-Informationen.
`tcp_debug.h`	Deklaration der Funktionen von `tcp_debug.c`.
`tcp_fsm.h`	Deklaration der Zustandsmaschine (*finite state machine*) von TCP.
`tcp_input.c`	Verarbeitung empfangener Segmente.
`tcp_output.c`	Generierung und Versand ausgehender Segmente.
`tcp_seq.h`	Definition einiger Makros zur Vereinfachung des Umgangs mit Sequenznummern.
`tcp_subr.c`	Einige Hilfsfunktionen, z.B. Allozieren, Initialisieren und Freigeben von Datenstrukturen einer TCP-Verbindung.
`tcp_timer.c`	Funktionen zur Realisierung der Zeitgeber-Steuerung.
`tcp_timer.h`	Definition aller Zeitgeber-Werte für TCP.
`tcp_usrreq.c`	Implementierung der Benutzerinteraktionen von TCP (z.B. Socket-Optionen).
`tcp_var.h`	Deklaration des TCP-Kontrollblocks (= Datenstruktur zur Aufnahme aller Zustandsinformationen für jeweils eine Verbindung).

Abbildung 4.1: Dateien der TCP-Implementierung

Das Modul `tcp_output.c` stellt die Funktion `tcp_output()` zur Verfügung. Diese wird jeweils aufgerufen, wenn die Anwendung dem Betriebssystem Daten zum Versenden übergeben hat (z.B. durch den Systemaufruf `send()`). Ein Aufruf von `tcp_output()` erfolgt auch von `tcp_input()` aus, um nach dem Empfang eines Segmentes eine entsprechende Bestätigung an den Sender zu generieren. Beim Aufruf wird zunächst überprüft, ob das Senden eines Segmentes nötig ist. Im Zustand `SYN_RECEIVED` muß z.B. ein entsprechendes Antwortsegment generiert werden, um den Verbindungsaufbau fortzusetzen. Ist die Verbindung im Zustand `ESTABLISHED`, ist z.B. dann ein Segment zu versenden, wenn entweder der Empfang von Daten zu bestätigen ist (sog. *Acknowledgements*) oder wenn sich Anwendungsdaten in der Sendewarteschlange befinden, die nicht mehr zurückgehalten werden sollen.[1]

Das Modul `tcp_timer.c` beinhaltet einige Funktionen, die für das Setzen von Zeitgebern zuständig sind und bei Ablauf dieser Zeitgeber die notwendigen Aktionen ausführen. Hierzu gehört beispielsweise das Aufrufen von `tcp_output()`, wenn der Zeitgeber für eine erneute Übertragung eines Segmentes (*Retransmit-Timer*) abgelaufen ist. Das Modul `tcp_usrreq.c` beinhaltet alle Funktionen, die direkt durch die Systemaufrufe der Anwendung ausgeführt werden, u.a. also `connect()`, `disconnect()`, `listen()`, `bind()` und `accept()`. Beispielsweise wird im Falle eines Aufrufs von `connect()` ein Verbindungsaufbau initiiert, indem der Zustand der Verbindung auf `SYN_SENT` gesetzt und die Funktion `tcp_output()` aufgerufen wird, um das erste SYN-Segment abzusenden.

[1] TCP versendet nicht sofort jegliche Anwendungsdaten, sondern wartet i.d.R. ab, ob nicht gleich darauf weitere Daten von der Anwendung übergeben werden. Somit können die Daten gebündelt in einem Segment übertragen werden, sofern ihre Länge die maximale Länge eines Segmentes nicht übersteigt.

4.3 Änderungen an der TCP-Implementierung

Aus dem Aufbau der TCP-Implementierung läßt sich leicht ersehen, daß für die Integration der Sicherheitsdienste zunächst die Module `tcp_input.c` und `tcp_output.c` erweitert werden müssen, so daß in die zu übertragenden Segmente zusätzlich zu den normalen Anwendungsdaten auch die kryptographischen Informationen integriert und diese beim Empfänger entsprechend ausgewertet werden. Zusätzlich müssen in `tcp_fsm.h` die neuen Zustände eingefügt werden, die für den geänderten Verbindungsaufbau notwendig sind. Die neuen Socketoptionen erfordern eine Erweiterung des Moduls `tcp_usrreq.c`.

Die kryptographischen Algorithmen können jeweils in neuen Modulen implementiert werden. Hierzu gehören einzelne Module für verschiedene Verschlüsselungs- und Signaturverfahren ebenso wie generelle kryptographische Algorithmen z.B. für das Erzeugen von Zufallszahlen und für das Rechnen mit sehr großen Zahlen.

In den folgenden Abschnitten werden die durchgeführten Erweiterungen des Protokolls jeweils in der Reihenfolge der Durchführung beschrieben.

4.3.1 Ein- und Ausschalten der Sicherheitsdienste

FreeBSD bietet durch den Systemaufruf `sysctl()` die Möglichkeit, den Wert einiger Variablen des Betriebssystemkerns auszulesen und zu verändern. Die Namen dieser Variablen sind hierarchisch gegliedert, so daß z.B. alle Variablen, die zur TCP-Implementierung gehören, welche wiederum unter den Oberbegriff Internet-Protokoll fällt, den Präfix `net.inet.tcp` tragen. Es wurde eine entsprechende Variable `net.inet.tcp.tcpsec` integriert, mit der sich die Verwendung der Sicherheitsdienste ein- und ausschalten läßt. Dies gefährdet die Sicherheit des Rechners nicht, da die Systemaufrufe zum Verändern dieser Einstellung nur vom Systemadministrator benutzt werden können.

Bei der Überprüfung, ob Sicherheitsdienste notwendig sind, wird jeweils der Wert dieser Variable abgefragt. Beim Initiieren einer Verbindung wird nur dann in den Zustand `SYN_SECU_SENT` gewechselt, wenn diese Variable gesetzt ist. Andernfalls wird der normale TCP-Verbindungsaufbau durchlaufen, auch die TCP-Option, die das Vorhandensein von Sicherheitsdiensten signalisiert, wird nicht gesendet. Auf bestehende TCP-Verbindungen hat der Wert dieser Variablen jedoch keinen Einfluß, unabhängig davon, ob es sich um ungesicherte oder gesicherte Verbindungen handelt.

4.3.2 Austausch der Sicherheitsoption

Zunächst wurden die Funktionen `tcp_output()` und `tcp_input()` so verändert, daß beim Generieren eines Segmentes im Zustand `SYN_SENT` und `SYN_RECEIVED` die Kennung der vorhandenen Sicherheitsdienste in die Optionen eingetragen und auf der Empfängerseite ausgewertet wird.

Beim Empfang eines Segmentes mit gesetztem SYN-Bit wird zunächst überprüft, ob die Kennung der Sicherheitsdienste in den Optionen enthalten ist. Es wird die neue Funk-

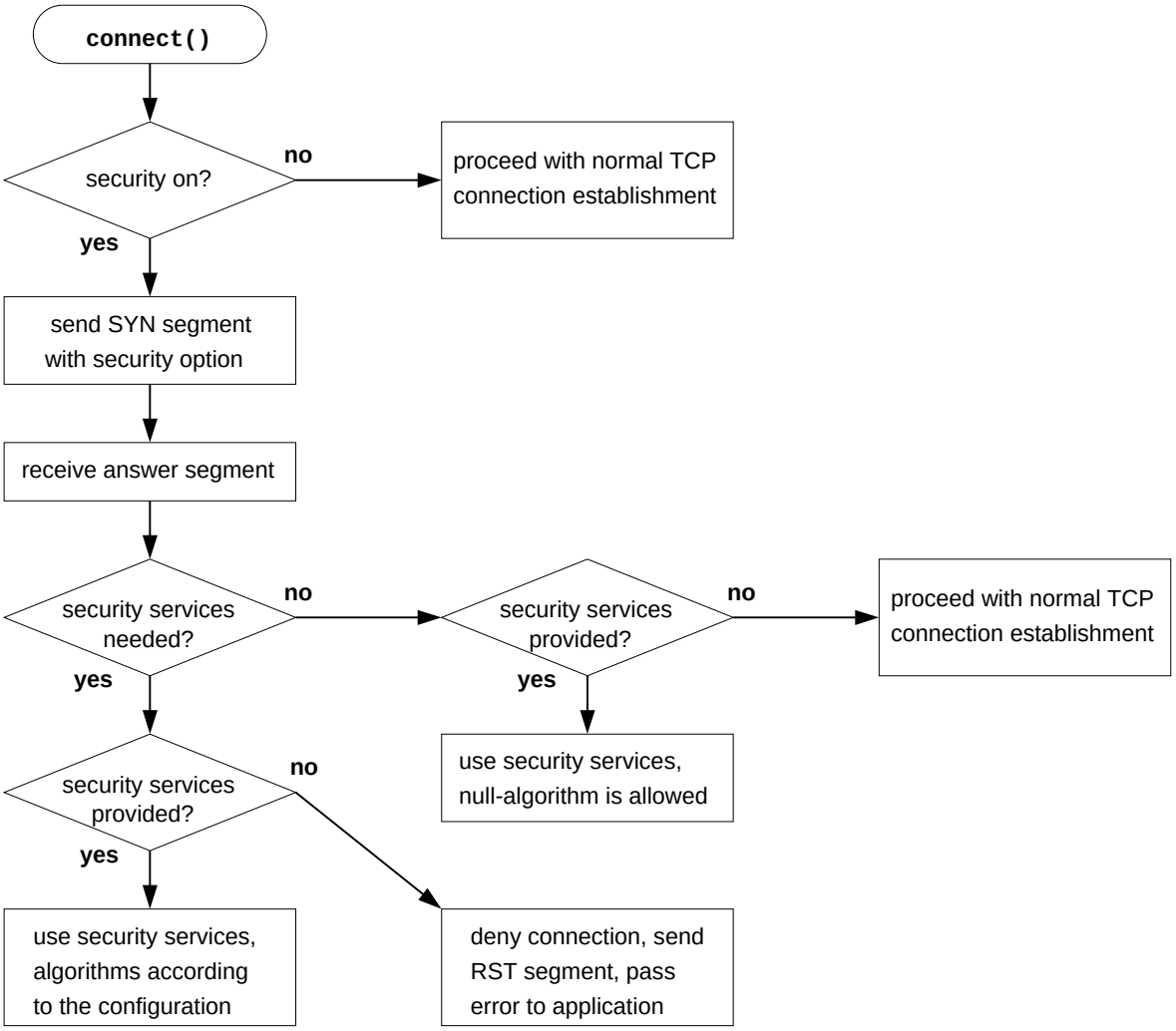


Abbildung 4.2: Ablauf der Entscheidung über den Einsatz von Sicherheitsdiensten auf der Seite des Clients

tion `tcps_set_incompar()` aufgerufen, die später einmal auf Grund der Konfigurationsdateien ermitteln wird, welche Sicherheitsfunktionen für diese Verbindung notwendig sind. Dies geschieht in Abhängigkeit von den IP-Adressen und Portnummern von Sender und Empfänger. Wenn Sicherheitsdienste auf Grund der lokalen Policy erforderlich sind, jedoch keine signalisierende TCP-Option mitgesendet wurde (siehe Abschnitt 3.3.3), wird die Verbindung sofort durch Senden eines RST-Segmentes abgebrochen. Die Funktion `tcps_set_incompar()` wurde zunächst in einer stark vereinfachten Variante erstellt. Sicherheitsdienste wurden vorläufig nur für eine feste Portnummer als notwendig vorausgesetzt. Dies ermöglichte bereits vor der Implementierung der Konfigurationsdateien das Testen der grundlegenden Signalisierung von Sicherheitsdiensten. Die Beschreibung der vollständigen Auswertung der notwendigen Sicherheitsparameter einer Verbindung erfolgt in Abschnitt 4.3.6. Abbildung 4.2 zeigt noch einmal den Ablauf der Entscheidungsfindung darüber, ob Sicherheitsdienste eingesetzt werden sollen.

Bei der Auswertung der Sicherheitsoption im ersten Segment ist im Falle von FreeBSD noch Sorge dafür zu tragen, daß für eine Verbindung nicht gleichzeitig sowohl der Einsatz von Sicherheitsdiensten als auch der Einsatz von Transaction-TCP (*T/TCP*) vereinbart werden kann. T/TCP nutzt ebenfalls die TCP-Optionen, um das Vorhandensein des entsprechenden Protokolls auf der Gegenseite festzustellen. Hierbei wird nach einem erfolgreichen Aufbau einer T/TCP-Verbindung in einem Cache vermerkt, daß der Kommunikationspartner über T/TCP verfügt. Beim nächsten Aufbau einer Verbindung zu diesem Rechner werden dann bereits im ersten SYN-Segment Anwendungsdaten übertragen. Natürlich dürfen beim Einsatz von Sicherheitsdiensten keine Daten angenommen und an die Anwendung weitergegeben werden, bevor der Aufbau einer gesicherten Verbindung erfolgt ist. Sonst könnten übertragene Daten bereits vor einer erfolgten Authentisierung die Anwendung erreichen. Sind in einem empfangenen SYN-Segment sowohl die Kennung für T/TCP als auch für die hier vorgestellten Sicherheitsdienste vorhanden, wird die T/TCP-Option ignoriert. Das entsprechende Antwortsegment enthält somit keine Kennung für T/TCP, es kommen nur die Sicherheitsdienste zum Einsatz. Bei ausgeschalteten Sicherheitsdiensten (siehe Abschnitt 4.3.1) wird mit der T/TCP-Kennung im nächsten Segment geantwortet, so daß bei weiteren Verbindungen der verkürzte Verbindungsaufbau von T/TCP stattfinden kann. Schaltet ein Administrator im laufenden Betrieb die Sicherheitsdienste von Rechner A zunächst aus und später wieder ein, kann es vorkommen, daß ein Kommunikationspartner B im Cache noch vermerkt hat, daß der Rechner A über T/TCP verfügt. Bei einem weiteren Verbindungsaufbau wird er also im SYN-Segment bereits Daten übertragen. Auch in diesem Falle müssen die Daten des SYN-Segmentes verworfen werden. Aus dem Antwortsegment von A kann B schließen, daß kein T/TCP vorhanden ist und den normalen Verbindungsaufbau fortführen. Die bereits gesendeten Anwendungsdaten müssen dann nach dem vollständigen Aufbau der Verbindung erneut übertragen werden.

4.3.3 Erweiterung des Protokollautomaten

Um die erforderlichen kryptographischen Informationen auszutauschen, muß der Verbindungsaufbau um einige Zustände erweitert werden (siehe Abschnitt 3.3.2). Hierzu wurde die Deklaration der Zustände im Modul `tcp_fsm.h` entsprechend erweitert. Im Modul `tcp_input.c` mußte die Funktion `tcp_input()`, die in Abhängigkeit vom empfangenen Segment und dem aktuellen Zustand der Verbindung den Folgezustand ermittelt, ergänzt werden, so daß beim Aufbau einer sicheren Verbindung die neue Zustandsfolge durchlaufen wird.

Für den Verbindungsaufbau existiert das Bit `TF_ACKNOW`, das in `tcp_input()` gesetzt wird, wenn ein Antwortsegment generiert werden muß. Dieses Bit wird beim folgenden Aufruf von `tcp_output()` ausgewertet. Auf diese Weise wird sichergestellt, daß beim Verbindungsaufbau auf ein empfangenes Segment auch das nächste Segment versendet wird, das für den Verbindungsaufbau notwendig ist. Dieses Bit muß also auch in allen neuen Zuständen gesetzt werden, die zum Verbindungsaufbau gehören.

Um die richtige Reihenfolge der ausgetauschten Segmente zu gewährleisten, wurde die Funktion `tcp_output()` so verändert, daß in allen Zuständen, die beim Verbindungsaufbau durchlaufen werden, die Sequenznummer des generierten Segmentes jeweils um

eins erhöht wird. Auf diese Weise werden die für den Verbindungsaufbau notwendigen Segmente durchnumeriert, so daß sich ein Verlust eines Segmentes feststellen läßt. Erhält ein Kommunikationspartner A auf sein an B gesendetes Segment bis zum Ablauf eines entsprechende Zeitgebers keine Antwort, sendet er dieses Segment erneut. Entweder ist bereits das erste Segment von A an B verlorengegangen oder aber die Antwort von B auf das erste Segment von A hat A nicht erreicht. Im ersten Fall erhält B im zweiten Anlauf das erwartete Segment und setzt den Verbindungsaufbau normal fort. Im zweiten Fall erhält B das Segment von A erneut und kann auf Grund der Sequenznummer feststellen, daß es sich um eine Wiederholung handelt. B wechselt daraufhin nicht in den Folgezustand, sondern ruft nur `tcp_output()` auf, um das offenbar verlorene Segment erneut zu senden. Dieses Vorgehen gilt bereits für den Auf- und Abbau von normalen TCP-Verbindungen, auf ein SYN-Segment mit der Sequenznummer n wird ebenfalls mit einem Bestätigungsnummer von $n+1$ geantwortet, obwohl das SYN-Segment kein Datenbyte enthielt.

Nach den bisherigen Änderungen signalisiert ein Kommunikationspartner A, der eine Verbindung initiiert, das lokale Vorhandensein der Sicherheitsdienste, wenn diese eingeschaltet sind. Der Kommunikationspartner B überprüft, ob das empfangene Segment die Kennung der Sicherheitsdienste enthält. Ist dies nicht der Fall, der Einsatz von Sicherheitsdiensten jedoch notwendig für diese Verbindung, wird der Verbindungsaufbau durch Senden eines RST-Segments abgebrochen. Wird von A kein Vorhandensein von Sicherheitsdiensten signalisiert, und sind diese aus Sicht von B auch nicht notwendig, so wird der normale TCP-Verbindungsaufbau durchlaufen (also die Zustände `SYN_SENT`, `SYN_RECEIVED` und `ESTABLISHED`). Sind Sicherheitsdienste auf beiden Seiten vorhanden, erfolgt ein erweiterter Verbindungsaufbau. Die dabei übertragenen Segmente sind jedoch zunächst noch leer, das Einfügen und Auswerten von kryptographischen Informationen ist noch nicht implementiert.

4.3.4 Einfügen und Auswerten der kryptographischen Informationen

In die zu versendenden Segmente müssen die benötigten kryptographischen Informationen eingetragen werden. Dies gilt sowohl für die Segmente des Verbindungsaufbaus, die die notwendigen Informationen zum Aushandeln von Algorithmen und Schlüsseln tragen, als auch für die Datensegmente einer etablierten Verbindung, die zusätzlich zu den Nutzdaten mit einer Signatur und mit Füllbytes versehen werden. Diese Informationen müssen auf der Empfängerseite wiederum ausgewertet werden. Es wurde ein neues Modul `tcp_secu.c` hinzugefügt, das die hierfür notwendigen Funktionen enthält.

Die Funktion `tcp_output()` bestimmt zu Beginn die Länge der Nutzdaten des zu versendenden Segmentes. Dies ist zunächst einmal die Anzahl der in der Sendewarteschlange befindlichen Anwendungsdaten. Diese Anzahl wird ggf. auf die Größe des aktuellen Sendefensters von TCP reduziert, um den Empfänger nicht zu überlasten (siehe [Stevens 1994, S. 275ff]). Weiterhin werden die Bytes abgezogen, die bereits gesendet, aber noch nicht vom Empfänger bestätigt wurden (siehe [Stevens 1995, S. 854ff]). Die berechnete Anzahl der zu sendenden Bytes wird noch einmal auf die maximale Länge der Nutzdaten eines Segmentes reduziert. Diese ist abhängig vom verwendeten Übertragungsmedium[2] und von der Einigung mit dem Kommunikationspartner über die ma-

[2]Die maximale Segmentlänge wird für ein verwendetes Übertragungsmedium i.d.R. fest vorgegeben, um

ximale Segmentgröße (*Maximum Segment Size* = MSS), die während des Verbindungsaufbaus durch eine TCP-Option durchgeführt werden kann.[3]

Werden nun kryptographische Informationen mit in das Segment integriert, muß die Anzahl der zu übertragenden Nutzdaten entsprechend reduziert werden, so daß sowohl Anwendungsdaten als auch kryptographische Daten in das Segment hineinpassen. Dazu wird zunächst in Abhängigkeit vom aktuellen Zustand der Verbindung die voraussichtliche Länge der kryptographischen Daten bestimmt. Diese ist auch abhängig von der Anzahl der zu übertragenden Nutzdaten, da die gesamte Segmentlänge durch Einfügen von Füllbytes (*Padding*) auf ein Vielfaches der Blocklänge des eingesetzten Verschlüsselungsverfahrens gebracht werden muß. Wird die Anzahl der zu übertragenden Bytes später verändert (z.B. weil Nutzdaten und kryptographische Daten die maximale Segmentlänge übersteigen), muß die Datenlänge jeweils so lange reduziert und die Länge der entsprechenden kryptographischen Informationen erneut berechnet werden, bis beide vollständig in das Segment hineinpassen.

Das Kopieren der Nutzdaten aus der Sendewarteschlange in einen neu allozierten Speicherbereich wird nun ersetzt durch einen Aufruf der neuen Funktion `tcp_createdata()`, die sich im Modul `tcp_secu.c` befindet. Hier werden zunächst die kryptographischen Informationen in den entsprechenden Speicherbereich eingefügt. Welche Informationen eingefügt werden müssen, wird aus dem aktuellen Zustand der Verbindung und den ausgehandelten Algorithmen bestimmt. Beispielsweise werden im Zustand `SYN_SECU_RECVD` die Listen der akzeptablen Verfahren für Schlüsselaustausch, Verschlüsselung und Signierung in das Segment eingetragen. Im Zustand `ESTABLISHED` erfolgt ein Einfügen der Nutzdaten mit einer entsprechenden Kennung, falls welche übertragen werden sollen. Weiterhin werden entsprechende Füllbytes hinzugefügt, wenn der verwendete Verschlüsselungsalgorithmus eine bestimmte Blocklänge der Daten erfordert. Dann wird die Signatur des Segmentes eingetragen, die über die alle Daten berechnet wurde. Das erzeugte Segment wird zuletzt mit dem gewählten Verschlüsselungsverfahren chiffriert. Die eingetragenen kryptographischen Informationen waren zunächst zu Testzwecken statisch vorgegebene Werte. Erst die Implementierung der Auswertung von Konfigurationsdateien ermöglichte, sinnvolle Werte zu ermitteln und in das Segment einzufügen (siehe Abschnitt 4.3.6).

Eine Schwierigkeit besteht in der Berechnung der Sequenz- und Bestätigungsnummern. Diese beziehen sich beim normalen TCP-Protokoll auf die übertragenen Nutzdaten. Sie werden auf der Empfängerseite benutzt, um die Daten eines neu empfangenen Segmentes an der richtigen Stelle in die Empfangswarteschlange mit den noch nicht an die Anwendung weitergereichten Daten einzusortieren. Da das Empfangen der Segmente nicht notwendigerweise in der richtigen Reihenfolge geschieht, kann es z.B. vorkommen, daß im Strom der empfangenen Daten eine Lücke besteht, weil ein Segment verloren gegangen ist. Die darauf folgenden Daten können erst an die Anwendung weitergereicht werden, wenn die fehlenden Daten empfangen wurden. Der Empfänger muß auch darauf vorbereitet sein, daß bei einem erneuten Übertragen die Segmente nicht identisch sind bezüglich der Daten, die sie enthalten. Z.B. können auf der Seite des Senders neue

zu verhindern, daß IP jedes Segment fragmentieren muß, damit die Länge der IP-Pakete in die Rahmen des physikalischen Übertragungsmediums passen. Dieses Vorgehen verhindert Geschwindigkeitseinbußen, die durch das Fragmentieren entstehen.

[3] Ohne diesen Einigungsprozeß beträgt die standardmäßige Segmentgröße 512 Bytes.

Anwendungsdaten eingetroffen sein, so daß bei der Wiederholung des Sendevorgangs in einem größeren Segment sowohl die bereits gesendeten als auch die neu hinzugekommenen Daten versendet werden. Die Datenbereiche der übertragenen Segmente können sich sogar überlappen. Geht beispielsweise eine Empfangsbestätigung verloren, werden die unbestätigten Daten erneut übertragen. Werden in diesem Segment weitere, noch nicht gesendete Daten übertragen, muß der Empfänger den ersten Teil der Daten verwerfen und nur die neuen Daten an die Anwendung weiterleiten. Ebenso kann es notwendig sein, einen Teil der Daten am Ende des Segmentes zu verwerfen, da sie nicht mehr in das Empfangsfenster passen (siehe [Stevens 1995, S. 957]).

Werden die übertragenen kryptographischen Daten mit in die Berechnung der Sequenz- und Bestätigungsnummern einbezogen, ergibt sich ein Problem bei der Zuordnung der Sequenznummern zu den übertragenen Nutzdaten. Bestätigt ein Empfänger mit der Bestätigungsnummer n den Empfang von Daten bis zur Sequenznummer $n-1$, muß der Sender aus dieser Information auf das nächste Datenbyte der Anwendungsdaten schließen können, das zu übertragen ist. Wenn die übertragenen kryptographischen Informationen in die Berechnung der Sequenznummer mit einbezogen werden, läßt sich nur schwerlich auf das entsprechende Datenbyte schließen. Die Länge der gesamten kryptographischen Informationen bis zu einem Datenbyte müßte mitprotokolliert werden. Dies ist um so schwieriger, da die Länge der kryptographischen Informationen in den Segmenten variabel ist. Sie schwankt z.B. in Abhängigkeit von den eingefügten Füllbytes für die Verschlüsselung. Somit müßte für jedes übertragene Segment vermerkt werden, wieviel kryptographische Information es beinhaltet hat, zumindest so lange, bis der Empfang bestätigt wurde. Bei Eingang einer Bestätigung müßte die Liste der vermerkten Segmentlängen durchlaufen werden, um die entsprechende Nummer der bestätigten Nutzdaten zu errechnen. Um eine entsprechend komplizierte Buchführung zu vermeiden, bietet es sich an, die kryptographischen Informationen nicht in die Berechnung der Sequenznummern einzubeziehen. Dies ermöglicht weiterhin die einfache Zuordnung von Sequenznummern zu den übertragenen Datenbytes. Nachteilig ist, daß auf diese Weise eine Datenflußanalyse vereinfacht wird, da sich die Menge der übertragenen Nutzdaten aus den im Klartext übertragenen Sequenznummern ergibt. Dies ließe sich jedoch einfach durch einen Offset verhindern, der in das Segment eingetragen und zu den übertragenen Sequenznummern addiert wird, um die wirkliche Sequenznummer dieses Segmentes zu erhalten. Bei verschlüsselten Segmenten wäre die Sequenznummer dann für einen Angreifer nicht mehr ersichtlich. Dieses Vorgehen wurde jedoch noch nicht implementiert.

Die übertragenen kryptographischen Informationen müssen vom Empfänger ausgewertet werden. Vor der normalen Verarbeitung eines TCP-Segmentes in der Funktion `tcp_input()` wurde deshalb ein Aufruf der neuen Funktion `tcp_process_secudata()` eingefügt, die sich im Modul `tcp_secu.c` befindet. Hier erfolgt zunächst die Entschlüsselung, falls diese vereinbart und die Verbindung bereits vollständig aufgebaut wurde. Darauf werden die Einträge im Segment sequentiell ausgewertet. Dabei wird für jeden Eintrag überprüft, ob er im jetzigen Zustand der Verbindung sinnvoll ist. Beispielsweise darf die Auswahl eines Verschlüsselungsalgorithmus nur in den Zuständen `LISTS_RCVD`, `SYN_SECU_RCVD` oder `CHOICES_SENT` erfolgen (siehe Abbildung 3.9). Andernfalls wird das empfangene Segment verworfen. Die Auswahl der kryptographischen Algorithmen beim Verbindungsaufbau wird in Abschnitt 4.3.6 detailliert beschrieben.

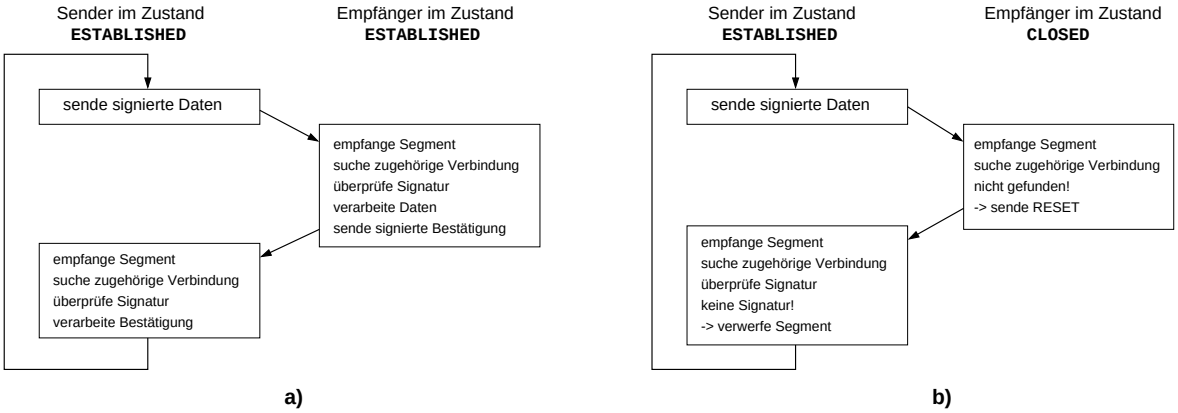


Abbildung 4.3: Schwierigkeiten von signierten Segmenten. In a) wird das normale Senden von Daten und Bestätigungen gezeigt. Beim plötzlichen Neustart eines Rechners befindet sich das TCP wieder im Zustand `CLOSED` (b). Die eingehenden Segmente werden durch ein RST-Segment beantwortet. Dieses wird jedoch vom Kommunikationspartner ignoriert, da es keine Signatur trägt.

Wenn beim erfolgten Verbindungsaufbau eine Signatur vereinbart wurde, wird diese überprüft. Ist sie nicht korrekt, wird das Segment ebenfalls ignoriert. Der Eintrag, der die Anwendungsdaten des Segmentes kennzeichnet, führt zu einem Bewegen dieser Daten an den Anfang des Segmentes (direkt hinter den TCP-Header), so daß ein gewöhnliches TCP-Segment entsteht, das in der normalen Art verarbeitet werden kann.

Wenn das Signieren der Segmente vereinbart wurde, werden Segmente mit ungültigen Signaturen verworfen. Dies kann allerdings zu unangenehmen Seiteneffekten führen. Wird z.B. der Kommunikationspartner A während einer bestehenden, signierten TCP-Verbindung plötzlich neu gestartet, versucht Partner B weiterhin, signierte Segmente zu versenden. Ist Rechner A wieder betriebsbereit, wird er auf diese Segmente von B mit einem RST-Segment antworten, da die zugehörige Verbindung für ihn nicht existiert. B muß jedoch diese RST-Segmente ignorieren, da sie nicht signiert und somit ungültig sind. Abbildung 4.3 stellt der normalen Verarbeitung den entstehenden Zyklus gegenüber. Es besteht daher die Notwendigkeit, auch unsignierte RST-Segmente zu verarbeiten.

Dieses Vorgehen ermöglicht jedoch einen Angriff auf die Verfügbarkeit der aufgebauten TCP-Verbindungen, ein Angreifer kann durch Versenden eines gefälschten, unsignierten RST-Segmentes den Abbau der Verbindung erreichen. Dies kann durch eine Heuristik unterbunden werden, indem z.B. eine gewisse Anzahl n an RST-Segmenten ignoriert wird. Wenn die Anzahl der RST-Segmente die Zahl n erreicht, erfolgt ein Abbruch der Verbindung. Wird jedoch vorher ein gültiges Segment empfangen, wird der RST-Zähler wieder mit Null initialisiert. Ein Angreifer kann die Verbindung also nur unterbrechen, wenn es ihm gelingt, zu verhindern, das ein gültiges Segment den Empfänger erreicht, bevor er n RST-Segmente gesendet hat. Wenn man diese Strategie noch mit einem entsprechenden Zeitgeber kombiniert, so daß eine Zeitspanne t auf den Eingang eines gültigen Segmentes gewartet wird, dann muß der Angreifer über diesen Zeitraum t verhindern, daß ein gültiges Segment zum angegriffenen Rechner gelangt.

Ein Angreifer, der über diese weitreichenden Möglichkeiten verfügt, ist ohnehin in der Lage, die Verbindung zu unterbrechen. Für einen weniger übermächtigen Angreifer stellt diese Strategie jedoch ein kaum zu überwindendes Hindernis dar. Diese Art von Denial-of-Service-Angriffen kann somit erkannt und verhindert werden. Diese Heuristik wurde in der Implementierung nicht umgesetzt, der Prototyp verarbeitet zunächst noch unsignierte RST-Segmente.

4.3.5 Integration neuer Socket-Optionen

Ein Vorteil der Integration von Sicherheitsdiensten in die TCP-Protokollschicht ist die direkte Kommunikation mit der Anwendung. Anwendungen erhalten die Möglichkeit einer Einflußnahme auf die Sicherheitsparameter. Dies geschieht durch Auslesen und Setzen von Socketoptionen mit den Systemaufrufen `getsockopt()` und `setsockopt()`. Im Modul `tcp_usrreq.c` sind die Optionen definiert, die für TCP-Sockets gelten. Diese wurden um die benötigten neuen Einträge erweitert. Die neuen Socketoptionen `TCP_KEYEXCHOICE`, `TCP_ENCINCHOICE`, `TCP_ENCOUTCHOICE`, `TCP_MACINCHOICE` und `TCP_MACOUTCHOICE`, die nach erfolgtem Verbindungsaufbau ein Auslesen der eingesetzten Algorithmen ermöglichen, können von der Anwendung nicht verändert werden. Bei einem Aufruf von `setsockopt` mit einem dieser Argumente wird der Fehler `EOPNOTSUPP` (Option nicht unterstützt) zurückgeliefert. Um die Auswahllisten der einzusetzenden Algorithmen anzupassen, wurden die Socketoptionen `TCP_KEYEXLIST`, `TCP_ENCRYPTLIST` und `TCP_SIGNLIST` hinzugefügt. Diese lassen sich sowohl auslesen als auch verändern.

4.3.6 Auswerten von Konfigurationsdateien

Für eingehende und ausgehende Verbindungen müssen Vorgaben existieren, die die notwendigen Sicherheitsdienste für diese Verbindung beschreiben. Diese Vorgaben können vom Administrator in Form einer Konfigurationsdatei erstellt werden. Da der Betriebssystemkern jedoch keine Möglichkeit besitzt, direkt auf eine Datei zuzugreifen, muß diese Konfiguration auf andere Weise an das Betriebssystem übertragen werden. Es wurde ein Binärformat gewählt, das die Einstellungen in kompakter Form wiedergibt. Ein Übersetzer `mk_db` transformiert eine textuelle Konfigurationsdatei in eine entsprechende Binärdatei. Eine solche Datei im Binärformat kann von dem neu erstellten Programm `set_db` eingelesen und über den Systemaufruf `sysctl()` an den Betriebssystemkern übermittelt werden. Ein Auslesen der aktuellen Konfiguration kann durch das Programm `get_db` erfolgen. Das in Abschnitt 3.4.2 beschriebene Format wurde hier umgesetzt.

Es wurden hierfür zwei neue Parameter für den Systemaufruf `sysctl()` definiert, jeweils einer für die Liste der eingehenden und einer für die Liste der ausgehenden Verbindungen. Eine mittels `sysctl()` übergebene Liste von Sicherheitseinstellungen wird in einen neu allozierten Speicherbereich kopiert. Im Betriebssystemkern wird ein Zeiger auf diesen Bereich gesetzt. `sysctl()` sorgt bei diesen Vorgängen durch Einsatz von Semaphoren selbständig dafür, daß keine nebenläufigen Änderungen derselben Variablen vorkommen.

Eine Verbindung wird gekennzeichnet durch die IP-Adressen und Portnummern von Sender und Empfänger. Da Wildcards in der Konfigurationsdatei erlaubt sind, kann es vorkommen, daß mehrere Einträge zu einer Verbindung passen. Beispielsweise überschneiden sich z.B. die Einträge `134.100.13.246` und `134.100.13.*`. Es muß also eine Reihenfolge der Regelanwendungen definiert werden, so daß eindeutig bestimmt ist, welcher Eintrag zu Anwendung kommt. In der durchgeführten Implementierung wurde als einziger Wildcard das Zeichen "*" eingeführt. Dieser wird intern als Zahl 256 repräsentiert. Somit besteht die Möglichkeit, die Einträge in aufsteigender Folge zu sortieren (dies wird vom Programm `mk_db` vorgenommen). Im obigen Beispiel wird demzufolge zunächst geprüft, ob der Eintrag `134.100.13.246` zu der Verbindung paßt. Ist dies nicht der Fall, wird mit dem Eintrag `134.100.13.*` fortgefahren.

Wenn auch Wertebereiche zugelassen werden (z.B. der Eintrag `134.100.13.100-135`), bietet es sich an, die Reihenfolge der Regeln in der Konfigurationsdatei als Entscheidungskriterium heranzuziehen (sog. *First Match*). Die IP-Adresse und Portnummer kommen als Sortierkriterium nicht mehr in Frage, da es schwer entscheidbar ist, welcher der Einträge `134.100.13.100-200`, `134.100.13.130-170` und `134.100.13.130-230` zuerst auf seine Anwendbarkeit geprüft werden soll.

Nach dem Systemstart ist zunächst keine Konfiguration der Sicherheitsdienste im Betriebssystemkern vorhanden. Dies bedeutet, daß weder eingehende noch ausgehende Verbindungen möglich sind. Zunächst kann mit dem Programm `set_db` jeweils eine Konfiguration für eingehende und für ausgehende Verbindungen aktiviert werden. Wird ein Serverprozeß gestartet, der mittels `listen()` und `accept()` auf eine eingehende Verbindung wartet, so wird eine Struktur vom Typ `tcpcb` (TCP-Control-Block) initialisiert. In diese können jedoch noch keine Listen von akzeptablen Algorithmen für die Verbindung eingetragen werden, da i.d.R. nur Port und IP-Adresse des Serverprozesses spezifiziert werden, nicht jedoch die des Clients. Der Serverprozeß kann jedoch über die Socketoptionen bereits eine Auswahl von Algorithmen für Verschlüsselung, Signatur und Schlüsselaustausch vorgeben, die für die Art der Anwendung in Frage kommen.

Versucht ein Clientprozeß, einen Verbindungsaufbau zu initiieren, kann auch er die akzeptablen Algorithmen durch die Socketoptionen vorgeben. Der Client kann versuchen, eine bestimmte eigene Portnummer und IP-Adresse zu erhalten (durch den Systemaufruf `bind()`). Wenn dies unterlassen wird, erfolgt die Zuteilung eines beliebigen verfügbaren SAPs durch das Betriebssystem beim Aufruf von `connect()`.[4] Die Funktion `connect()` erhält als Parameter den SAP des gewünschten Kommunikationspartners. Somit sind zu diesem Zeitpunkt die SAPs der Verbindung spezifiziert. Das einleitende SYN-Segment wird zunächst gesendet.

Das TCP des Serverprozesses erhält das SYN-Segment und kennt nun auch den SAP des Kommunikationspartners. Sind die Sicherheitsdienste aktiviert, wird lokal in der Konfiguration für eingehende Verbindungen nachgesehen, welche Sicherheitsdienste zum Einsatz kommen können. Wenn es keinen passenden Eintrag gibt, verbleibt die Verbindung im Zustand `LISTEN` und es wird ein RST-Segment an den Client gesendet, um das Ablehnen der Verbindung zu signalisieren. Aus dem Aufruf von `accept()` wird also die Kontrolle nicht zurück an den Serverprozeß zurückgegeben, so daß der Prozeß im Zustand `sleeping` verbleibt. Der Serverprozeß bemerkt auf diese Weise nicht den Versuch

[4] Natürlich ist nur die Portnummer beliebig, die IP-Adresse ist die des lokalen Rechners.

eines unerlaubten Verbindungsaufbaus. Findet sich jedoch ein zu der aufzubauenden Verbindung gehöriger Eintrag in der Konfiguration, werden die dort vorgegebenen Listen von Algorithmen mit den durch die Anwendung gesetzten Listen zusammengeführt (siehe Abschnitt 3.4.1). In das Antwortsegment werden die resultierenden Listen und die TCP-Option, die das Vorhandensein der Sicherheitsdienste anzeigt, eingetragen.

Auf der Seite des Clients wird bei Erhalt dieses Segments geprüft, ob diese Verbindung zustande kommen darf. Hierzu wird ein passender Eintrag in der Konfiguration für ausgehende Verbindungen gesucht. Die Verbindung wird auch hier mit einem RST-Segment abgebrochen, wenn kein entsprechender Eintrag existiert. In diesem Fall kehrt der Systemaufruf `connect()` mit dem Fehler `ECONNREFUSED` zurück. Die Anwendung stellt also fest, daß die Verbindung abgelehnt wurde. Dieses Vorgehen ist sinnvoll, da auf der Seite des Clients i.d.R. eine Anwendung läuft, die den Benutzer darüber informieren muß, daß die Verbindung nicht zustande gekommen ist. Auf der Serverseite ist es jedoch besser, den Serverprozeß, der zumeist ein Hintergrundprozeß (Dämon) ist, nicht über das Nichtzustandekommen der Verbindung zu informieren. Entsprechende Fehlermeldungen werden i.d.R. als Nichtverfügbarkeit des Netzwerkes interpretiert, so daß sich der Prozeß evtl. beenden würde. Stattdessen erfolgt für diese Ereignisse jedoch ein Eintrag in den Logdateien des Systems, damit Angriffsversuche vom Administrator oder entsprechenden Erkennungssystemen (*Intrusion Detection Systems*) bemerkt werden.

Existiert ein Eintrag in der Konfiguration, der zu der Verbindung gehört, werden auch hier die Daten der Anwendungsliste (aus den Socketoptionen) mit denen der Konfiguration zusammengeführt. Die erhaltenen Listen vom Serverprozeß werden nach Einträgen von Algorithmen durchsucht, die auch in der soeben generierten lokalen Liste vorhanden sind. In der bestehenden Implementation wird jeweils der erste Algorithmus aus der lokalen Liste gewählt, der auch Element der vom Kommunikationspartner empfangenen Liste ist. Die Auswahl kann jedoch auch nach anderen Kriterien erfolgen, es könnten z.B. Algorithmen bevorzugt werden, für die auch Unterstützung in Form eines Kryptoprozessors existiert. Der ausgewählte Algorithmus wird im Kontrollblock der Verbindung eingetragen und im nächsten Segment an den Kommunikationspartner übermittelt. Gibt es keinen Algorithmus, der für beide Seiten akzeptabel ist, wird als Auswahl des Algorithmus der Wert Null verwendet. Der Kommunikationspartner bricht bei Empfang dieser Auswahl die Verbindung mit einem RST-Segment ab, wenn Sicherheitsdienste für die Verbindung erforderlich sind. Nun werden die Auswahlen und die Listen der Clientseite übertragen. Der bisherige Vorgang des Zusammenführens der Algorithmenlisten und der Auswahl von Algorithmen wird in Abbildung 4.4 noch einmal dargestellt. Auf der Seite des Serverprozesses findet daraufhin derselbe Vorgang statt, jedoch diesmal unter Verwendung der lokalen Konfiguration für eingehende Verbindungen.

Ein Sonderfall ist das gleichzeitige Initiieren einer Verbindung von beiden Kommunikationspartnern (*Simultaneous Open*). Hierbei wird im Zustand `SYN_SECU_SENT` ein Segment empfangen, das keine Listen von Algorithmen und keine Bestätigungsnummer enthält. Es werden daraufhin die zu diesem Zweck eingeführten neuen Zustände durchlaufen (siehe Abbildung 3.10). Im Zustand `SYN_SECU_SENT` werden dabei bereits die lokalen Listen in den Kontrollblock der Verbindung eingetragen. Zu diesem Zeitpunkt wird schon überprüft, ob ein entsprechender Eintrag sowohl in der Konfiguration für eingehende als auch in der für ausgehende Verbindungen enthalten ist. Dieses Ver-

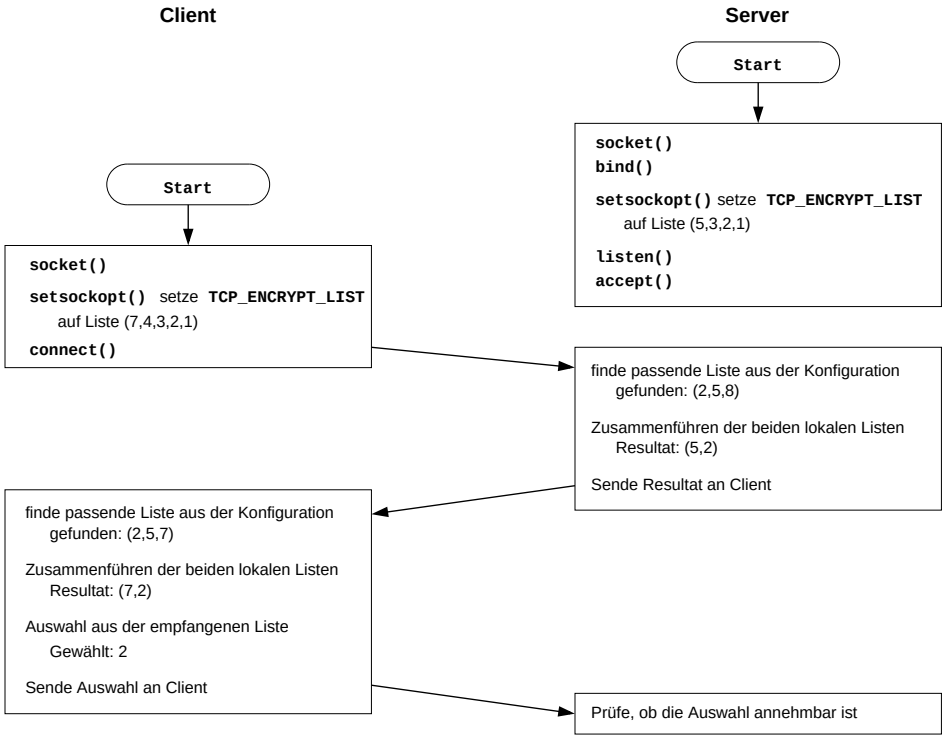


Abbildung 4.4: Auswahl der Algorithmen am Beispiel der Verschlüsselung vom Client zum Server. Der Server führt zunächst die Applikationsliste und die der Konfiguration entnommene Liste zusammen. Dabei entfallen die nicht konfigurierten Verfahren 1 und 3. Die von der Anwendung gewählte Reihenfolge (2 folgt nach 5) bleibt dabei erhalten. Der Client ermittelt ebenfalls das Resultat seiner beiden lokalen Listen. Aus der Schnittmenge dieses Resultats und der empfangenen Liste erfolgt die Auswahl des Algorithmus (in diesem Falle bleibt nur die 2).

wenden beider Konfigurationsdateien ist sinnvoll, da ein *Simultaneous Open* sowohl ein Initiieren als auch ein Entgegennehmen einer Verbindung darstellt. Ein Algorithmus kann in diesem Fall nur dann verwendet werden, wenn er in beiden Konfigurationsdateien für diese Verbindung eingetragen ist.

4.3.7 Integration der kryptographischen Algorithmen

Auf dem beschriebenen Stand der Implementierung werden nur Segmente generiert, deren kryptographische Informationen aus statischen Einträgen bestehen. Echte kryptographische Funktionalität fehlt bisher. In heutigen Betriebssystemkernen sind kryptographische Funktionen i.d.R. noch nicht vorhanden. Dies bedeutet, daß ein Einsatz von Kryptographie zunächst eine Integration von grundlegenden Funktionen voraussetzt. Beispiele hierfür sind z.B. das Generieren von großen Zufallszahlen, die kryptographischen Ansprüchen genügen, oder Funktionen zum Potenzieren von großen Zahlen. Anwendungen können diese Funktionen einfach durch Einsatz von bereits existierenden Bibliotheken nutzen. Im Falle einer Erweiterung des Kernels müssen diese jedoch vollständig neu implementiert werden, da der Kernel über keine Zugriffsmöglichkeit auf

Bibliotheken verfügt.

Aus diesem Grunde wurde in der durchgeführten Implementierung z.B. kein kryptographisches Schlüsselaustauschverfahren realisiert. Die Implementierung von Zertifikaten und Funktionen, die darauf aufbauend einen Schlüsselaustausch ausführen, hätte den Rahmen der vorliegenden Arbeit gesprengt. Für die Verwaltung von Zertifikaten wäre insbesondere eine Möglichkeit zu realisieren, wie das verschlüsselnde TCP an die geheimen Schlüssel der Benutzer gelangt, ohne daß diese offen in einigen Dateien verfügbar sein müssen. Eine Lösung dieses Problems wäre ein Dämonprozeß, der auf Anfrage des Betriebssystemkerns den entsprechenden Schlüssel liefert. Dieser Dämonprozeß kann bei einer Anfrage des Betriebssystems den Benutzer auffordern, ein Passwort einzugeben, mit dem der in einer Datei vorliegende geheime Schlüssel des Benutzer dechiffriert werden kann. Der Dämonprozeß speichert diesen Schlüssel im Hauptspeicher, um ihn auch bei späteren Anfragen des Kernels für weitere TCP-Verbindungen angeben zu können. Auf Anwendungsebene wird dieses Verfahren bereits in SSH eingesetzt, der entsprechende Dämonprozeß trägt den Namen `ssh-agent`. Die Interaktion mit dem Benutzer kann hierbei über die Textkonsole oder über ein X11-Fenster geschehen. Zum Einsatz von Zertifikaten wäre weiterhin eine *Public Key Infrastruktur* (PKI) notwendig, also mindestens ein Server, der auf Anfrage von Clients die Echtheit von übermittelten Zertifikaten überprüft.

In der Implementierung werden anstelle von Zertifikaten nur statische Einträge übertragen, die keine weitere Verwendung finden. Für den Schlüsselaustausch werden zufällige Werte gewählt. Jedoch kann mit diesen kein Diffie-Hellman-Schlüsselaustausch durchgeführt werden, da kein Austausch von Zertifikaten stattgefunden hat. Stattdessen werden die übertragenen Werte der Kommunikationspartner jeweils addiert, um den Schlüssel für die Verschlüsselung und die Signatur zu erhalten. Dies hat den Vorteil, daß die Berechnung des Schlüssels wie beim Diffie-Hellman-Verfahren jeweils von den Werten beider Kommunikationspartner abhängig ist. Wirkliche Sicherheit wird dadurch nicht erlangt, da die Werte im Klartext übertragen werden. Auf diese Weise läßt sich jedoch der Schlüsselaustausch simulieren und die weiteren kryptographischen Funktionen können somit getestet werden.

Die Länge der übertragenen Schlüsselaustauschwerte ist die maximale Schlüssellänge, die von einem der ausgehandelten Algorithmen benötigt wird. Durch Abschneiden der hinteren Bytes wird jeweils der Schlüssel für das Verfahren mit der kürzeren Schlüssellänge generiert. Kommen als Verschlüsselungsverfahren DES und als Signaturverfahren HMAC-MD5 zum Einsatz, wird ein 16 Byte langer Wert übertragen (16 Bytes = Schlüssellänge von HMAC-MD5). Der 8 Byte lange Schlüssel für DES besteht aus den ersten 8 Bytes dieses Wertes.

Zum Testen wurden jeweils nur ein Verschlüsselungs- und ein Signaturalgorithmus umgesetzt. Als Verschlüsselungsverfahren wurde DES im ECB-Modus gewählt, als Signaturverfahren HMAC-MD5. Diese lagen bereits im Quellcode vor und mußten nur noch geringfügig angepaßt werden. Der DES-Quellcode wurde `ssh` (Version 1.2.13) entnommen, hierfür wurde die Implementierung um die Module `tcp_des.c` und `tcp_des.h` erweitert. Für das Verfahren HMAC-MD5 findet sich eine Referenzimplementierung im zugehörigen RFC [Krawczyk 1997]. Die MD5-Hashfunktion ist bereits standardmäßig in den FreeBSD-Kernel integriert, so daß nur noch die Umsetzung als Signaturalgorithmus hinzugefügt werden mußte. Dies geschieht, indem der MD5-Algorithmus mehrfach

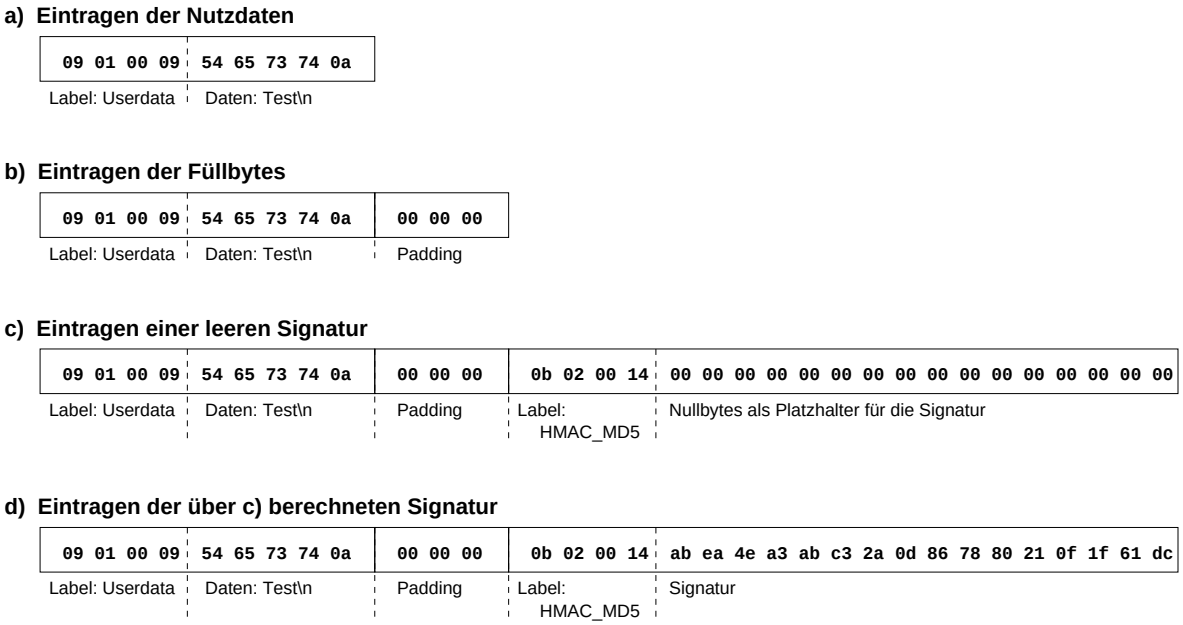


Abbildung 4.5: Die einzelnen Schritte des Signierens und Verschlüsselns. Zunächst werden Füllbytes angehängt, um die Gesamtlänge (einschließlich der noch nicht vorhandenen Signatur) auf ein Vielfaches der benötigten Blocklänge zu bringen. Dann wird ein Eintrag für die Signatur erzeugt, der zunächst mit Nullen gefüllt ist. Über diese Daten wird die Signatur gebildet und anstelle der Nullbytes eingetragen. Danach erfolgt die Verschlüsselung (nicht mehr dargestellt).

aufgerufen wird, wobei als Parameter sowohl der benutzte Schlüssel als auch der zu signierende Text verwendet wird. Der entsprechende Funktionsaufruf lautet *MD5(K XOR opad, MD5(K XOR ipad, Text))*, wobei K ein n-Byte Schlüssel und *Text* der zu signierende Text ist. Die Konstanten *ipad* und *opad* sind festgelegte Füllwerte. Diese Funktion wurde im neuen Modul `tcp_hmacmd5.c` implementiert, die zugehörigen Deklarationen befinden sich in `tcp_hmacmd5.h`.

Die Verschlüsselung durch den ausgewählten Algorithmus findet als letzter Schritt in der Funktion `tcp_createdata()` statt. Zuvor wird der Nutzdatenteil des zu versendenden TCP-Segmentes durch Nullbytes so aufgefüllt, daß die Länge ein Vielfaches der Blocklänge des Verschlüsselungsverfahrens beträgt. Wurde eine Signatur der Segmente vereinbart, wird zusätzlich die Kennung und die Längenangabe der Signatur in das Segment eingetragen. Der Wert der Signatur wird zunächst durch Nullbytes aufgefüllt. Über dieses Segment (mit den Füllbytes und dem leeren Eintrag für die Signatur) wird die Signatur berechnet und diese dann anstelle der Nullbytes eingetragen. Der Vorgang des Signierens und Verschlüsselns eines Segmentes ist in Abbildung 4.5 dargestellt.

In der bestehenden Implementierung wird die Signatur nur über die Nutzdaten des Segmentes berechnet. Durch Einsatz eines Pseudo-Headers können auch noch weitere Daten des TCP-Segmentes in die Signatur einbezogen werden (siehe 3.2.3). Dies könnten z.B. die IP-Adressen und Portnummern von Sender und Empfänger und die Sequenznummer sein.

Auf der Seite des Empfängers werden die erhaltenen Nutzdaten zunächst dechiffriert, falls eine Verschlüsselung vereinbart wurde. Die Einträge im Segment werden dann sequentiell verarbeitet. Beim Auffinden einer gültigen Signatur wird ein entsprechendes Bit gesetzt. Nur wenn dieses Bit nach der Verarbeitung des Segmentes gesetzt ist, werden die Anwendungsdaten in den zugehörigen Socket der Verbindung kopiert. Andernfalls wird das Segment verworfen. Um die Signatur zu überprüfen, wird der Signaturwert aus dem Segment in einen Zwischenspeicher kopiert. Der entsprechende Speicherbereich im empfangenen Segment wird durch Nullen überschrieben. Auf diese Weise entsteht dasselbe Segment, über das der Sender die Signatur berechnet hat, so daß eine Verifizierung der Signatur stattfinden kann.

4.4 Werkzeuge zum Testen der Implementierung

Um die Implementierung zu testen, wurde zunächst das Testprogramm `sock` aus [Stevens 1994] verwendet. Dieses kann als Serverprozeß gestartet werden und wartet dann zunächst auf eine eingehende Verbindung. Ein weiterer Aufruf des Programms als Client führt zu einem Verbindungsaufbau zum Serverprozeß. Nun werden weitere Tastatureingaben jeweils beim Zeilenabschluß an den Kommunikationspartner übertragen. Auf diese Weise kann ein TCP-Verbindungsaufbau und ein nachfolgendes Übertragen von Testdaten erfolgen.

Das Programm wurde zunächst um einen Kommandozeilenparameter erweitert, über den ein Auslesen und Setzen der neuen Socketoptionen bewirkt wird. Somit läßt sich testen, ob das Zusammenführen der vom Administrator konfigurierten Algorithmenlisten mit den von der Anwendung gesetzten funktioniert.

Es wurde ein weiterer Parameter hinzugefügt, der bewirkt, daß das Programm weder als Client noch als Server gestartet wird, sondern versucht, eine Verbindung mit sich selbst auf demselben SAP einzugehen. Hierzu wird zunächst der Socket mittels `socket()` und `bind()` an einen lokalen Port gebunden. Auf diesem Port wird jedoch kein `listen()` durchgeführt, sondern mit `connect()` eine Verbindung zu diesem Port selbst hergestellt. Der Socket ist nun mit sich selbst verbunden; Segmente, die auf diesem Socket gesendet werden, werden auch sofort auf diesem Socket empfangen. Durch dieses Vorgehen läßt sich das *Simultaneous Open* von TCP simulieren [Stevens 1994, S. 960ff]. Beim Verbindungsaufbau wird zunächst ein SYN-Segment gesendet. Dieses SYN-Segment wird sofort auf dem Socket empfangen, das TCP durchläuft daraufhin die Zustandsfolge für einen gleichzeitigen Verbindungsaufbau. Ein gleichzeitiges Initiieren einer Verbindung von unterschiedlichen Ports stellt ein nicht unerhebliches Problem dar, da eine sehr exakte Synchronisierung der entsprechenden Prozesse stattfinden muß.

Während das Programm `sock` verwendet wurde, um die entsprechenden Segmente zu generieren, wurde zugleich ein ausführliches Protokoll der vom TCP durchgeführten Aktionen erstellt. Auf diese Weise ließ sich das korrekte Verhalten des Prototyps überprüfen, auftretende Fehler konnten gefunden werden. Als zusätzliches Hilfsmittel kam das Programm `tcpdump` zum Einsatz. Dieses versetzt die Netzwerkkarte in den sog. *Promiscuous Mode*, so daß alle auf dem Medium übertragenen Rahmen aufgezeichnet werden können.[5] Durch Angabe von Filterregeln (z.B. Rechnername, Portnummer oder

[5] Gewöhnlich verarbeitet der Netzwerkadapter nur Pakete, die an seine eigene Adresse geschickt wur-

Protokoll) lassen sich die Pakete auswählen, die von `tcpdump` protokolliert werden sollen, alle anderen werden nicht beachtet. Auf diese Weise kann man eine Datei erhalten, die alle gesendeten und empfangenen Segmente einer TCP-Verbindung enthält. Dadurch können Fehler bei der Generierung von Segmenten gefunden werden. Eine Beschreibung der von `tcpdump` generierten Ausgabe bei einem erfolgten Verbindungsaufbau erfolgt in Abschnitt 5.5.

Da mit dem Programm `sock` über die Tastatureingabe nur kleine Datenmengen generiert werden, wurde zusätzlich das Programm `netcat` benutzt. Auch dieses kann sowohl als Server als auch als Client eingesetzt werden. Es überträgt Daten, die von der Standardeingabe des Clientprozesses gelesen wurden, der Server gibt diese wiederum auf seiner Standardausgabe aus. So können durch Umleiten der Ein- und Ausgabe auch große Dateien übertragen werden. Diese Datenübertragungen wurden ebenfalls zunächst mit `tcpdump` aufgezeichnet und dann auf Korrektheit überprüft.

Abschließend wurde das Meßwerkzeug `netperf`[6] benutzt, um den erreichbaren Datendurchsatz bei Einsatz von Verschlüsselung, Signierung oder beidem zu ermitteln. Das Meßwerkzeug besteht aus einem Serverprogramm `netserver`, der permanent auf eingehende Verbindungen wartet. Der Meßclient `netperf` öffnet zunächst eine Kontrollverbindung zum Server und teilt diesem mit, welche Art von Messung er durchführen möchte. Der Server führt daraufhin ein *Passive Open* auf einem neuen Port aus. Er überträgt die Nummer des verwendeten Ports über die bestehende Kontrollverbindung, so daß der Meßclient den Aufbau einer Datenverbindung initiieren kann. Bei Durchsatzmessungen werden nun über einen angegebenen Zeitraum möglichst viele Pakete an den Server übertragen. Die hierfür verwendeten Größen der Sende- und Empfangspuffer und der generierten Pakete können über Kommandozeilenargumente angegeben werden. Ist die Zeitspanne abgelaufen, wird die Datenverbindung geschlossen. Meßergebnisse, die nur dem Server bekannt sind (z.B. die Empfangsleistung bei Verwendung von UDP) werden über die Kontrollverbindung an den Client übertragen, so daß dieser die gesammelten Ergebnisse ausgeben kann. Auch ein wiederholtes Messen unter Berechnung des Mittelwerts, der Standardabweichung und der Fehlerintervalle kann mit diesem Meßwerkzeug durchgeführt werden. Entsprechende Messungen wurden unter Einsatz verschiedener Sicherheitsdienste durchgeführt. Zum Vergleich wurde auch der Durchsatz bei Einsatz der normalen TCP-Implementierung ermittelt. Die Ergebnisse der Messung werden in Abschnitt 6.3 vorgestellt.

---

den. Im *Promiscuous Mode* werden alle Pakete verarbeitet, die über das Medium übertragen werden, also auch die, die an andere Rechner gerichtet sind. Somit läßt sich der gesamte Netzwerkverkehr aufzeichnen.

[6] Das Programm `netperf` findet sich unter `http://www.netperf.org`, es wurde jedoch eine erweiterte Version eingesetzt, die mehrere Meßiterationen durchführt und die Standardabweichung und Fehlerintervalle berechnet. Diese Version wurde vom DFN Firewall Labor für Hochgeschwindigkeitsnetze (`http://www.fwl.dfn.de`) erstellt.

Kapitel 5

Arbeiten mit dem Prototypen

Dieses Kapitel soll den praktischen Einsatz des erstellten Prototypen beschreiben. Es stellt die Installation und die Konfiguration der Sicherheitsdienste anhand von Beispielen vor.

Die Sicherheitsdienste wurden im wesentlichen in den Betriebssystemkern (engl. *Kernel*) integriert. Daher ist die Installation eines Kernels nötig, der über die Sicherheitsdienste verfügt. Weiterhin müssen die Sicherheitsdienste den Anforderungen entsprechend konfiguriert werden. Hierzu bedarf es der notwendigen Programme und der Erstellung von Konfigurationsdateien.

5.1 Installation des Betriebssystemkerns

Die einfachste Lösung ist die Verwendung eines bereits vorkompilierten Betriebssystemkerns. Dieser kann in das Wurzelverzeichnis "`/`" kopiert werden. Beim Start des Systems muß dem *Bootloader* mitgeteilt werden, daß dieser Kernel geladen werden soll (standardmäßig wird der Kernel "`/kernel`" benutzt). Da in den Betriebssystemkern jedoch auch ein wesentlicher Teil der Gerätetreiber integriert ist, ist ein vorkompilierter Kernel nicht auf jedem Gerät einsetzbar. Es empfiehlt sich, die Quellen der Sicherheitsdienste in den Verzeichnisbaum der Kernelquellen zu kopieren. Ersetzt wird das Verzeichnis `netinet`, in dem sich die Implementierung der Internetprotokolle befindet, und die Datei `conf/files`, in der die benötigten Quelldateien für den Kernel angegeben sind. Für das Übersetzen des Betriebssystemkerns ergeben sich keine weiteren Veränderungen.

Wurde der neu generierte Betriebssystemkern mit dem Aufruf "`make install`" installiert, so befindet er sich unter dem Namen `kernel` im Wurzelverzeichnis, er wird bei einem Neustart des Betriebssystems automatisch geladen.

5.2 Konfiguration der Sicherheitsdienste

Nach einem Neustart verfügt der Betriebssystemkern über keine Konfiguration für die einzusetzenden Sicherheitsdienste. Dies bedeutet, daß keine ausgehenden und einge-

henden TCP-Verbindungen möglich sind. Die Sicherheitsdienste lassen sich durch Aufruf des Programms `sysctl` ein- und ausschalten. Der Aufruf

```
sysctl -w net.inet.tcp.tcpsec=0
```

schaltet die Sicherheitsdienste aus, durch Eingabe von

```
sysctl -w net.inet.tcp.tcpsec=1
```

werden sie wieder aktiviert.

Um eine Konfiguration für die Sicherheitsdienste zu generieren, müssen die hierfür benötigten Programme `mk_db`, `set_db` und `get_db` installiert sein. Zunächst muß jeweils eine Konfiguration für ausgehende und eine für eingehende Verbindungen erstellt werden. Für benötigte Dienste des Rechners muß angegeben werden, ob diese Sicherheitsdienste verwenden sollen und welche Algorithmen in diesem Fall zum Einsatz kommen dürfen. Verbindungen zu Rechnern, die keine Sicherheitsdienste verwenden, kommen nur zustande, wenn diese durch den Eintrag `nosecu` gekennzeichnet werden. Auf diese Weise kann der Einsatz von Sicherheitsdiensten innerhalb eines internen Netzes erfolgen, bei Verbindungen zu externen Netzwerken, die i.d.R. keine Sicherheitsdienste unterstützen, jedoch normales, ungesichertes TCP verwendet werden. Weiterhin kann auf diese Weise bei Anwendungen, die bereits von sich aus sichere Kommunikation ermöglichen (z.B. `ssh`), auf den Einsatz der Sicherheitsdienste verzichtet werden, um eine mehrfache Verschlüsselung zu vermeiden.

5.3 Beispiel einer Konfiguration

Anhand eines Beispiels soll die Konfiguration der Sicherheitsdienste schrittweise beschrieben werden. Der betreffende Rechner besitzt die IP-Adresse `134.100.13.244`. Er soll im wesentlichen als Client eingesetzt werden, es sollen keine eingehenden Verbindungen angenommen werden. Einzige Ausnahme bildet der Dienst `slogin`. Dieser Dienst verfügt ohnehin über ausreichende Sicherheitsmechanismen und muß nicht zusätzlich abgesichert werden. Der Dienst soll nur aus der Domain des Betreibers (in diesem Fall das Class-B-Netzwerk `134.100`) zugreifbar sein. Der zugehörige Server `sshd` wartet auf Port 22 auf eingehende Verbindungen. Die Konfigurationsdatei für eingehende Verbindungen sollte wie folgt aussehen:

```
134.100.13.244:22  134.100.*:*    nosecu exclist=01 enclist=01 maclist=01
```

Die Datei ist eine Textdatei mit einem beliebigen Namen. Die Einträge für die einzusetzenden Verschlüsselungsverfahren werden von dem Werkzeug, das die Konfigurationsdatei übersetzt, erwartet. Aus diesem Grund ist jeweils das Null-Verfahren für Verschlüsselung und Signatur angegeben. Für ausgehende Verbindungen kommt die folgende Konfigurationsdatei zum Einsatz:

```
134.100.13.244:*  134.100.*:23        exclist=01 enclist=02 maclist=02
134.100.13.244:*  193.13.178.10:23    exclist=01 enclist=02 maclist=02
134.100.13.244:*  *:22                nosecu exclist=01 enclist=01 maclist=01
134.100.13.244:*  134.100.9.80:80     nosecu exclist=01 enclist=01 maclist=01
```

Hier sollen ausgehende Verbindungen innerhalb des internen Netzwerkes immer Sicherheitsdienste einsetzen (mit Ausnahme der `ssh`-Verbindungen). Es werden nur einzelne Dienste explizit zugelassen, andere Verbindungen können nicht zustande kommen. Der Dienst `telnet` (Portnummer 23) ist nur innerhalb des internen Netzes

möglich. Ausnahme sind Verbindungen zum Rechner `193.13.178.10` (Zeile 2). In beiden Fällen muß das Verschlüsselungsverfahren DES (Nummer 02) und das Signaturverfahren HMAC-MD5 (Nummer 02) eingesetzt werden. Als Schlüsselaustauschverfahren kommt Diffie-Hellman (Nummer 01) zum Einsatz.[1] Der Dienst `ssh` wird in der dritten Zeile konfiguriert. Entsprechende Verbindungen können zu beliebigen Rechnern geöffnet werden, nicht nur innerhalb der eigenen Domain. Der WWW-Dienst `http` soll nur über einen Proxy im internen Netzwerk (IP-Adresse 134.100.9.80, Portnummer 80) möglich sein. Diese Verbindungen sind i.d.R. nicht vertraulich und erzeugen eine hohe Netzlast, so daß hierbei auf Verschlüsselung verzichtet werden kann. Der entsprechende Eintrag befindet sich in Zeile 4.

5.4 Einsatz der Konfiguration

Die erstellten Konfigurationsdateien müssen nun noch an den Betriebssystemkern übergeben werden. Hierzu werden sie zunächst durch den Übersetzer `mk_db` in ein Binärformat gewandelt. Um z.B. die Konfigurationsdatei `config_outgoing.txt` in eine Binärdatei mit Namen `config_outgoing.db` zu wandeln, ist der Aufruf

```
mk_db config_outgoing.txt config_outgoing.db
```

nötig. Diese Wandlung ist unabhängig davon, ob es sich um die Konfiguration für ausgehende oder für eingehende Verbindungen handelt. Mit dem Programm `set_db` kann eine solche Binärdatei an den Betriebssystemkern übergeben werden. Das Programm erhält einen Parameter, der angibt, ob die Datei für die ausgehenden oder die eingehenden Verbindungen verwendet werden soll. Als zweites Argument folgt der Name der Datei. Um also die Binärdatei `config_outgoing.db` für ausgehende und `config_incoming.db` für eingehende Verbindungen zu verwenden, wird `set_db` zweifach aufgerufen:

```
set_db out config_outgoing.db
set_db in config_incoming.db
```

Die somit übermittelten Konfigurationen werden nun vom Betriebssystemkern verwendet. Die aktuellen Konfigurationen des Kernels können in analoger Weise mit dem Programm `get_db` ausgegeben werden, der Aufruf

```
get_db in
```

führt zur Ausgabe der aktuellen Konfiguration für eingehende Verbindungen. Diese sieht im obigen Falle wie folgt aus:

```
/orion/home/olaf/tcps_config/get_db: Getting incoming db
Len of db: 76
Entry: oIP: 134.100.13.244, oPort: 22
       fIp: 134.100.256.256, fPort: 65536
       exclist: 1, 0, 0 enclist: 1, 0, 0 maclist: 1, 0, 0
       secu_flags: 1
```

[1] Es ist zu beachten, daß im vorliegenden Prototyp kein echtes Diffie-Hellman-Verfahren implementiert wurde. Durch den nur simulierten Schlüsselaustausch wird keine wirkliche Sicherheit erreicht.

Die Angaben `oIP` und `oPort` bezeichnen die lokale IP-Adresse und Portnummer (o = own), `fIP` und `fPort` jeweils IP-Adresse und Portnummer des entfernten Rechners. Der Wert 1 für die `secu_flags` ist die Repräsentation für den Eintrag `nosecu`, der anzeigt, daß Sicherheitsdienste nicht zum Einsatz kommen müssen. Die Zahl 256 innerhalb einer IP-Adresse und die Zahl 65535 als Portnummer sind die interne Repräsentation für den Platzhalter "*". Das Tool `mk_db` sortiert die Einträge in aufsteigender Reihenfolge. Es wird also zunächst für alle spezifischen Einträge überprüft, ob sie zu einer aufzubauenden Verbindung gehören. Die Platzhalter-Regel wird zuletzt überprüft. Die Reihenfolge `134.100.13.*:*` vor `134.100.13.244:22` würde sonst bedeuten, daß der zweite Eintrag nie erreicht wird, da bereits der erste Eintrag zu der Verbindung paßt.

5.5 Ablauf einer gesicherten TCP-Verbindung

In diesem Abschnitt wird der Ablauf einer TCP-Verbindung, bei deren Aufbau Sicherheitsdienste ausgehandelt werden, beschrieben. Der Ablauf des in Kapitel 3 entworfenen Protokolls läßt sich auf diese Weise anschaulich nachvollziehen. Tabelle 5.1 zeigt die übertragenen Segmente während der gesamten Dauer der Verbindung in ihrer zeitlichen Abfolge. Die ersten sechs Segmente dienen dabei dem Aufbau der Verbindung, die nachfolgenden zwei sind die übertragenen Daten und die zugehörige Empfangsbestätigung. Die letzten vier Segmente sind zum Abbau der Verbindung gehörig. Die Verbindung wird vom Rechner *cevap* initiiert, der empfangende Rechner trägt den Namen *cici*.

Zunächst wird von *cevap*, dem Initiator der Verbindung, ein SYN-Segment übertragen. Dieses enthält die TCP-Option 20 (hexadezimal 14), die das Vorhandensein der Sicherheitsdienste signalisiert. Der Rechner *cici* registriert das Vorhandensein von Sicherheitsdiensten und ermittelt die für diese Verbindung akzeptablen Algorithmen für Schlüsselaustausch, Verschlüsselung und Authentisierung. Es werden sowohl die signalisierende TCP-Option als auch die Listen der akzeptablen Algorithmen in dem in Abbildung 3.12 spezifizierten Format übermittelt.

Es folgt der Austausch der weiteren benötigten Informationen, wie z.B. Auswahl der Algorithmen, Zertifikat (in diesem Fall nur ein beliebiger Füllwert, siehe 4.3.7) und Schlüsselaustauschwert. Die in Abbildung 3.9 gezeigten Zustände werden dabei durchlaufen. Beim Konfigurieren der Sicherheitsdienste wurde keine Verschlüsselung gewählt, damit die Struktur der Nutzdaten der Segmente in der Tabelle weiterhin sichtbar ist. Als Signaturverfahren wird HMAC-MD5 gewählt (Auswahl 02 im Eintrag *Choices: Sign*).

Das siebte Segment enthält Anwendungsdaten, die nach dem erfolgten Verbindungsaufbau von *cevap* an *cici* übertragen werden. Die Nutzdaten sind mit einem entsprechenden Eintrag (hexadezimal 0901) gekennzeichnet, es folgt die Signatur des Segmentes (Eintrag 0b02). Das achte Segment ist die Bestätigung für die empfangenen Daten. Es enthält daher keine Anwendungsdaten, beinhaltet jedoch eine Signatur. Ein Vergleich mit anderen Segmenten, die keine weiteren Daten tragen (Segmente 6, 9, 10, 11 und 12), zeigt, daß die Signaturen jeweils identisch sind.

18:45:53.898160 cevap:1024 > cici:4000	cici → **SYN_SECU_RECEIVED**
S 51702532:51702532(0) <mss 1460,nop,nop,opt-20:>	
4500 0030 0000 4000 4006 1217 8664 0df4	
8664 0df5 0400 0fa0 0314 eb04 0000 0000	
7002 4000 08b5 0000 0204 05b4 0101 **14**02	Security-Option
18:45:53.902277 cici:4000 > cevap:1024	cevap → **LISTS_RCVD**
S 50666493:50666509(16) ack 51702533	
<mss 1460,nop,nop,opt-20:>	
4500 0040 0000 4000 4006 1207 8664 0df5	
8664 0df4 0fa0 0400 0305 1bfd 0314 eb05	
7012 4470 d60e 0000 0204 05b4 0101 **14**02	Security-Option
0101 0005 01**03** **01**00 0501 **0501** 0006 0202	Lists: KeyEx, Encrypt, Sign
18:45:53.906787 cevap:1024 > cici:4000	cici → **CHOICES_RCVD**
. 1:41(40) ack 1	
4500 0050 0001 4000 4006 11f6 8664 0df4	
8664 0df5 0400 0fa0 0314 eb05 0305 1bfe	
5010 4470 ffa9 0000 **0301** 0005 01**05** **01**00	Lists: Encrypt, Sign
0602 02**02** **02**00 0501 **0402** 0005 01**06** **02**00	Choices: KeyEx, Encrypt, Sign
0502 **0701** 000e fefd fefd fefd fefd fefd	Certificate
18:45:53.913012 cici:4000 > cevap:1024	cevap → **WAIT_ACK**
. 1:45(44) ack 2	
4500 0054 0001 4000 4006 11f2 8664 0df5	
8664 0df4 0fa0 0400 0305 1bfe 0314 eb06	
5010 4470 23a8 0000 **0402** 0005 01**06** **02**00	Choices: Encrypt, Sign
0502 **0701** 000e fefd fefd fefd fefd fefd	Certificate
0801 0014 fed1 f027 f35a c51f 028f f896	KeyExValue
2b67 19f6	
18:45:53.919178 cevap:1024 > cici:4000	cici → **ESTABLISHED**
. 2:22(20) ack 2	
4500 003c 0002 4000 4006 1209 8664 0df4	
8664 0df5 0400 0fa0 0314 eb06 0305 1bff	
5010 4470 c4da 0000 **0801** 0014 bbe8 22c6	KeyExValue
66f0 20ee 0510 6b49 b90a c4fe	
18:45:53.924178 cici:4000 > cevap:1024	cevap → **ESTABLISHED**
. 2:22(20) ack 3	
4500 003c 0002 4000 4006 1209 8664 0df5	
8664 0df4 0fa0 0400 0305 1bff 0314 eb07	
5010 4470 2203 0000 **0b02** 0014 ec14 a6f5	Signature
e4a3 57d7 1bb3 c428 0a92 3ad2	
18:45:57.190937 cevap:1024 > cici:4000	
P 3:32(29) ack 2	
4500 0045 0003 4000 4006 11ff 8664 0df4	
8664 0df5 0400 0fa0 0314 eb07 0305 1bff	

Tabelle 5.1: wird fortgesetzt

5018 4470 3c97 0000 **0901** 0009 5465 7374	UserData: Test\n
0a**0b** **02**00 14ab ea4e a3ab c32a 0d86 7880	Signature
210f 1f61 dc	
18:45:57.371750 cici:4000 > cevap:1024	
. 2:22(20) ack 8	
4500 003c 0003 4000 4006 1208 8664 0df5	
8664 0df4 0fa0 0400 0305 1bff 0314 eb0c	
5010 4470 21fe 0000 **0b02** 0014 ec14 a6f5	Signature
e4a3 57d7 1bb3 c428 0a92 3ad2	
18:46:00.991261 cevap:1024 > cici:4000	cici → **CLOSE_WAIT**
F 8:28(20) ack 2	
4500 003c 0004 4000 4006 1207 8664 0df4	
8664 0df5 0400 0fa0 0314 eb0c 0305 1bff	
5011 4470 21fd 0000 **0b02** 0014 ec14 a6f5	Signature
e4a3 57d7 1bb3 c428 0a92 3ad2	
18:46:00.995779 cici:4000 > cevap:1024	cevap → **FIN_WAIT2**
. 2:22(20) ack 9	
4500 003c 0004 4000 4006 1207 8664 0df5	
8664 0df4 0fa0 0400 0305 1bff 0314 eb0d	
5010 4470 21fd 0000 **0b02** 0014 ec14 a6f5	Signature
e4a3 57d7 1bb3 c428 0a92 3ad2	
18:46:00.999414 cici:4000 > cevap:1024	cevap → **TIME_WAIT**
F 2:22(20) ack 9	
4500 003c 0005 4000 4006 1206 8664 0df5	
8664 0df4 0fa0 0400 0305 1bff 0314 eb0d	
5011 4470 21fc 0000 **0b02** 0014 ec14 a6f5	Signature
e4a3 57d7 1bb3 c428 0a92 3ad2	
18:46:01.003591 cevap:1024 > cici:4000	cici → **CLOSED**
. 9:29(20) ack 3	
4500 003c 0005 4000 4006 1206 8664 0df4	
8664 0df5 0400 0fa0 0314 eb0d 0305 1c00	
5010 4470 21fc 0000 **0b02** 0014 ec14 a6f5	Signature
e4a3 57d7 1bb3 c428 0a92 3ad2	

Tabelle 5.1: Segmente einer TCP-Verbindung, die vom Rechner `cevap` zum Empfänger `cici` geöffnet wird. Die in Schreibmaschinenschrift gesetzten Texte im linken Teil der Tabelle sind die Ausgaben von `tcpdump`. Diese bestehen jeweils aus einer textuellen Darstellung der wichtigsten Informationen des TCP-Headers (die ersten zwei bis drei Zeilen zu jedem Segment), gefolgt vom Inhalt des gesamten Segmentes (in hexadezimaler Darstellung). Die von `tcpdump` ausgegebenen Informationen zu den einzelnen Segmenten wurden der Übersichtlichkeit halber leicht verkürzt. Rechts von der ersten Zeile zu jedem Segment ist der Zustand angegeben, in den das empfangende TCP auf das Segment hin wechselt. Innerhalb der Segmente sind die Kennungen der Sicherheitseinträge durch Fettdruck hervorgehoben (vgl. Abbildung 3.12). Rechts sind jeweils die Namen der Einträge angegeben, deren Kennungen in der Zeile stehen.

Die Berechnung der Signatur berücksichtigt bei der bestehenden Implementierung nur die Nutzdaten des Segmentes, die in diesem Falle nur aus der Kennzeichnung der Signatur und den Nullbytes bestehen, die später mit der Signatur überschrieben werden (vgl. Abschnitt 4.3.7). Würden zusätzliche Informationen des Segmentkopfes (also z.B. die Sequenznummer) mit in die Berechnung einbezogen, hätten Segmente mit gleichem Dateninhalt i.d.R. trotzdem unterschiedliche Signaturen.

Die folgenden vier Segmente gehören zum Abbau der Verbindung. Es wird hierbei die übliche Zustandsfolge durchlaufen wie dies auch beim normalem TCP der Fall ist. Die Segmente besitzen jedoch alle eine Signatur im Nutzdatenteil.

5.6 Ausgabe von Debugging-Informationen

Eine neu hinzugefügte Variable des Betriebssystemkerns, die über den Aufruf `sysctl()` verändert werden kann, ist der sog. *Loglevel*, der die Ausführlichkeit der Ausgaben der Sicherheitsdienste reguliert. Ein Loglevel von 3 führt zu sehr detaillierten, umfangreichen Ausgaben. Der Aufruf

```
sysctl -w net.inet.tcp.tcplog=1
```

setzt den Loglevel auf 1, so daß nur sicherheitsrelevante Meldungen (also z.B. der Empfang eines falsch signierten Segmentes) protokolliert werden. Diese Ausgaben dienen zunächst nur der Fehlersuche, da das Arbeiten mit einem Debugger keine Analyse des laufenden Betriebssystemkerns ermöglicht. Für die Messungen in Abschnitt 6.3 wurden diese Ausgaben vollständig ausgeschaltet (Loglevel = 0).

Im folgenden soll ein Teil des ausführlichen Protokolls der in Tabelle 5.1 gezeigten Verbindung gezeigt und kommentiert werden. Das gezeigte Protokoll wurde auf dem Rechner *cevap* erstellt, beschreibt also nur die Vorgänge auf diesem Rechner. Aus Platzgründen werden hier nur die Verarbeitungsschritte für die ersten drei Segmente aufgeführt.

Setzen der Socketoptionen

```
Jun  6 18:45:53 cevap /kernel: setsockopt: set exclist
Jun  6 18:45:53 cevap /kernel: setsockopt: now copying exclist...
Jun  6 18:45:53 cevap /kernel: setsockopt: set enclist
Jun  6 18:45:53 cevap /kernel: setsockopt: now copying enclist...
Jun  6 18:45:53 cevap /kernel: setsockopt: set maclist
Jun  6 18:45:53 cevap /kernel: setsockopt: now copying maclist...
```

Von der verwendeten Anwendung `sock` werden vor dem Aufbau der Verbindung zunächst die Socketoptionen gesetzt.

Einleitendes SYN-Segment

```
Jun  6 18:45:53 cevap /kernel: tcp_output: called. state = 3
Jun  6 18:45:53 cevap /kernel: snd_nxt: 314eb04, snd_max: 314eb04,
                               snd_una: 314eb04, rcv_nxt: 0
Jun  6 18:45:53 cevap /kernel: tcp_output: computing cryptolen.
```

```
Jun  6 18:45:53 cevap /kernel: datalen 0, off 0, win 0
Jun  6 18:45:53 cevap /kernel: tcp_output: computed cryptolen
Jun  6 18:45:53 cevap /kernel: tcp_get_cryptolen: encalg_len computed: 9
Jun  6 18:45:53 cevap /kernel: tcp_get_cryptolen: macalg_len computed: 9
Jun  6 18:45:53 cevap /kernel: tcp_get_cryptolen: macdata_len computed: 0
Jun  6 18:45:53 cevap /kernel: tcp_get_cryptolength: TCPS_SYN_SECU_SENT. 0
Jun  6 18:45:53 cevap /kernel: tcp_output: No CC Option!
Jun  6 18:45:53 cevap /kernel: tcp_output: window opened 2 segs. send
Jun  6 18:45:53 cevap /kernel: tcp_output: segment sent
Jun  6 18:45:53 cevap /kernel: snd_nxt: 314eb05, snd_max: 314eb05,
                               snd_una: 314eb04, rcv_nxt: 0
```

Durch den Systemaufruf `connect()` wird `tcp_output()` angestoßen. Dieses erhält eine initiale Sequenznummer. Es wird die Länge der im Socket vorhandenen Daten ermittelt. Diese ist zunächst Null. Der Offset *off* ist ebenfalls Null, er vermerkt die bereits gesendeten Daten. Die Funktion `get_cryptolength()` ermittelt die Länge der benötigten kryptographischen Daten, sie beträgt bei diesem Segment 0. Das generierte Segment wird abgesendet.

Antwortendes SYN-Segment

```
Jun  6 18:45:53 cevap /kernel: tcp_input: called.
Jun  6 18:45:53 cevap /kernel: tcp_input: len before processing: 16
Jun  6 18:45:53 cevap /kernel: tcp_input: secu needed!
Jun  6 18:45:53 cevap /kernel: tcp_process_secudata: 16 bytes in segment.
Jun  6 18:45:53 cevap /kernel: tcp_process_secudata: parsing exclist
Jun  6 18:45:53 cevap /kernel: tcp_process_exclist: listlen = 1
Jun  6 18:45:53 cevap /kernel: tcp_process_exclist: key exchange = 1
Jun  6 18:45:53 cevap /kernel: tcp_process_secudata: parsing enclist
Jun  6 18:45:54 cevap /kernel: tcp_process_enclist: listlen = 1
Jun  6 18:45:54 cevap /kernel: tcp_process_enclist: encryption out = 1
Jun  6 18:45:54 cevap /kernel: tcp_process_secudata: parsing maclist
Jun  6 18:45:54 cevap /kernel: tcp_process_maclist: listlen = 2
Jun  6 18:45:54 cevap /kernel: tcp_process_maclist: macoutchoice = 2
Jun  6 18:45:54 cevap /kernel: tcp_input: len after processing: 0
Jun  6 18:45:54 cevap /kernel: tcp_input: TCPS_SYN_SECU_SENT -> TCPS_LISTS_RCVD
Jun  6 18:45:54 cevap /kernel: tcp_input: all data acked: needoutput
Jun  6 18:45:54 cevap /kernel: tcp_input: update win: needoutput
Jun  6 18:45:54 cevap /kernel: tcp_input: needoutput. calling tcp_output
```

Das Antwortsegment vom Rechner *cici* wird von der Funktion `tcp_input()` verarbeitet. Es enthält 16 Datenbytes. Es wird festgestellt, daß Sicherheitsdienste für diese Verbindung eingesetzt werden sollen. Die enthaltenen Listen von möglichen Algorithmen werden nacheinander verarbeitet. Weitere Daten (also Nutzdaten der Anwendung) sind nicht vorhanden. Nachdem dies erfolgreich geschehen ist, wechselt die TCP-Verbindung in den Zustand `LISTS_RCVD`. Da der Aufbau der Verbindung noch nicht abgeschlossen ist, wird `tcp_output()` aufgerufen, um das nächste Segment zu generieren.

Senden des Algorithmenlisten und -auswahlen

```
Jun  6 18:45:54 cevap /kernel: tcp_output: called. state = 8
Jun  6 18:45:54 cevap /kernel: snd_nxt: 314eb05, snd_max: 314eb05,
```

```
                                snd_una: 314eb05, rcv_nxt: 3051bfe
Jun  6 18:45:54 cevap /kernel: tcp_output: computing cryptolen.
Jun  6 18:45:54 cevap /kernel: datalen 0, off 0, win 17520
Jun  6 18:45:54 cevap /kernel: tcp_get_cryptolen: encalg_len computed: 5
Jun  6 18:45:54 cevap /kernel: tcp_get_cryptolen: macalg_len computed: 6
Jun  6 18:45:54 cevap /kernel: tcp_get_cryptolen: macdata_len computed: 20
Jun  6 18:45:54 cevap /kernel: tcp_get_cryptolength: TCPS_LISTS_RCVD. 40
Jun  6 18:45:54 cevap /kernel: tcp_output: computed cryptolen
Jun  6 18:45:54 cevap /kernel: tcp_get_cryptolen: encalg_len computed: 5
Jun  6 18:45:54 cevap /kernel: tcp_get_cryptolen: macalg_len computed: 6
Jun  6 18:45:54 cevap /kernel: tcp_get_cryptolen: macdata_len computed: 20
Jun  6 18:45:54 cevap /kernel: tcp_get_cryptolength: TCPS_LISTS_RCVD. 40
Jun  6 18:45:54 cevap /kernel: tcp_output: t_flags 8001
Jun  6 18:45:54 cevap /kernel: tcp_output: owe an ack. sending
Jun  6 18:45:54 cevap /kernel: tcp_createdata: state 8, datalen: 0, cryptolen: 40
Jun  6 18:45:54 cevap /kernel: tcp_createdata: restmbuf: 40, skiplen 40
Jun  6 18:45:54 cevap /kernel: tcp_createdata: TCPS_LISTS_RCVD. 40 bytes
Jun  6 18:45:54 cevap /kernel: tcp_output: copying all data! old mbuf. 40 bytes.
Jun  6 18:45:54 cevap /kernel: tcp_output: created data successfully
Jun  6 18:45:54 cevap /kernel: tcp_output: segment sent
Jun  6 18:45:54 cevap /kernel: snd_nxt: 314eb06, snd_max: 314eb06,
                                snd_una: 314eb05, rcv_nxt: 3051bfe
```

Die Sequenznummer des zu versendenden Segmentes ist um eins erhöht worden (vgl. erstes gesendetes Segment). Es sind wiederum keine Anwendungsdaten vorhanden, durch den Austausch der ersten zwei Segmente wurde die Größe des Sendefensters bereits ausgehandelt. Wieder wird die Länge der benötigten kryptographischen Information ermittelt. Diese beträgt 40 Bytes (siehe Abbildung 5.1). Das Segment wird gesendet.

Kapitel 6

Ergebnisse und Ausblick

Der in Kapitel 3 vorgestellte Entwurf wurde weitestgehend in einer Implementierung umgesetzt. Die dabei gemachten Erfahrungen bilden den Inhalt des nächsten Abschnittes. Abschnitt 6.2 wird die bei der Implementierung erreichten Ziele bezüglich der Sicherheit beschreiben. Im dritten Abschnitt werden die Ergebnisse von Durchsatzmessungen, die unter Einsatz der Sicherheitsdienste durchgeführt wurden, vorgestellt. Diese bilden ein Maß für die Einsetzbarkeit der Sicherheitsdienste unter dem Gesichtspunkt der Verarbeitungsgeschwindigkeit. Abschließend folgt ein Ausblick auf notwendige weitere Schritte in der Entwicklung von Sicherheitsdiensten.

6.1 Erfahrungen bei der Implementierung

Die Implementierung von Sicherheitsdiensten innerhalb des Betriebssystemkerns erwies sich als komplexe Aufgabe. Zum einen ergaben sich die üblichen Schwierigkeiten, die bei Änderungen am Kernel auftreten:

- Der Einsatz von Debuggern ist i.d.R. nicht möglich, so daß das Erkennen und Beseitigen von Fehlern erschwert wird.
- Fehlerhafte Anweisungen führen oftmals zu einem sofortigen Systemabsturz, so daß der Fehler nur anhand der Protokollausgaben des Kernel auf der Systemkonsole nachvollziehbar sind.
- Das Testen von Änderungen kann nur nach einem Neustart des Systems durchgeführt werden, da der neu generierte Betriebssystemkern zunächst geladen werden muß.
- In heutigen Betriebssystemkernen herrscht nur eine geringe Vielfalt an verfügbaren Funktionen (die für die Anwendungsprogrammierung in umfangreichen Bibliotheken zur Verfügung stehen). Insbesondere sind i.d.R. noch keine Funktionen vorhanden, die den Einsatz kryptographischer Verfahren erleichtern. Dies sind z.B. Zufallszahlengeneratoren oder mathematische Funktionen zum Rechnen mit sehr großen Zahlen.

Zum anderen erwies sich das Erweitern eines Netzwerkprotokolls ebenfalls als eine Herausforderung. Ausführliche Dokumentation und weitreichende Erfahrungen in der Betriebssystemprogrammierung waren eine Grundvoraussetzung für die erfolgreiche Durchführung der Implementierung. Insbesondere die bisherigen Erweiterungen des klassischen TCP (als Beispiel sei *T/TCP* angeführt) erhöhen die Komplexität der Internetprotokolle erheblich, so daß das Verständnis der bestehenden Implementierung und das korrekte Einfügen von Änderungen erschwert wird.

Trotz dieser Umstände zeigte sich, daß die Integration komplexer Sicherheitsdienste möglich und sinnvoll ist. Die Verwendung von Sicherheitsdiensten, die bereits in den Betriebssystemkern integriert sind, ermöglicht den sicheren Einsatz von Anwendungen, die von sich aus keine geschützte Kommunikation benutzen (z.B. `telnet`, `ftp` oder `rsh`). Die Konfiguration kann durch den Systemverwalter für alle Dienste einheitlich erfolgen. Änderungen der Quellcodes oder ein zusätzliches Konfigurieren der Sicherheitsdienste einer jeden Anwendung sind somit nicht mehr nötig. Die Transparenz dieser Sicherheitsdienste ermöglicht einen komfortablen Einsatz von kryptographischen Verfahren, ohne daß dies vom Anwender bemerkt wird.

6.2 Erreichte Sicherheit

In die TCP-Protokollschicht wurden Sicherheitsdienste integriert, deren Parameter beim Aufbau jeder Verbindung zwischen den Kommunikationspartnern ausgehandelt werden. Hierdurch ist es möglich, eine Policy umzusetzen, die für einzelne Dienste Vorgaben bezüglich der notwendigen Sicherheitsdienste macht. Kann der Kommunikationspartner diesen Sicherheitsvorgaben nicht genügen, kommt keine Verbindung zu Stande.

Als Sicherheitsfunktionen kommen Verfahren zur Verschlüsselung und zur Signierung zum Einsatz, die die Vertraulichkeit, die Integrität und die Authentizität und Integrität der übertragenen Daten sicherstellen. Hierzu müssen zusätzlich zu den Anwendungsdaten die notwendigen kryptographischen Informationen (Schlüssel, Auswahlen von einzusetzenden Verfahren) ausgetauscht werden. Zur Integration dieser Informationen in die Segmente von TCP wird allen Einträgen in den Nutzdatenteil jeweils eine Kennung vorangestellt, die angibt, welcher Art die folgenden Daten sind. Somit können die übertragenen Anwendungsdaten und die verschiedenen kryptographischen Informationen unterschieden und entsprechend verarbeitet werden. Durch dieses Vorgehen wird eine hohe Flexibilität des Protokolls erreicht, da weitere Daten für zukünftige Erweiterungen mit Hilfe von neuen Kennungen in die Segmente integriert werden können. Durch diese Erweiterbarkeit lassen sich Schwachstellen in der erstellten Implementierung durch neue Ergänzungen beheben.

6.2.1 Vertraulichkeit

Als Verschlüsselungsverfahren kommt einfaches DES im ECB-Modus zum Einsatz. Die Verwendung des ECB-Modus vereinfacht die Implementierung, da jedes Segment unabhängig von allen anderen ver- und entschlüsselt werden kann. Diese Unabhängigkeit

vom Inhalt der bisher versandten bzw. empfangenen Daten ist wichtig, da die Empfangsreihenfolge der Segmente nicht der Sendereihenfolge entsprechen muß. Der ECB-Modus erreicht jedoch nicht die Sicherheit der anderen Modi (siehe Seite 10), die eine Rückkopplung mit vorher verschlüsselten Daten verwenden. Um eine höhere Sicherheit zu gewährleisten, empfiehlt sich der Einsatz dieser rückgekoppelten Modi. Um die Entschlüsselung von Segmenten unabhängig von ihrer Sendereihenfolge zu ermöglichen, muß mit jedem Segment der entsprechende Initialisierungsvektor übertragen werden. Die erstellte Implementierung macht diese Erweiterung jedoch einfach, da nur ein entsprechender neuer Eintrag spezifiziert werden muß, der den Initialisierungsvektor für das Segment aufnimmt.

Verschlüsselt wird nur der Nutzdatenteil der TCP-Segmente. Eine weitergehende Verschlüsselung der im Kopfteil enthaltenen Protokollinformationen sollte aus Gründen der Kompatibilität nicht erfolgen. Jedoch könnten einzelne Einträge durchaus verändert werden. Ein Beispiel hierfür sind die Sequenz- und Bestätigungsnummern, die einem Angreifer zumindest Auskunft über die übertragenen Datenmengen geben. Auch dieses Problem kann durch einen neuen Eintrag in die Nutzdaten gelöst werden, der einen Offset angibt. Dieser wird zu der übertragenen Sequenznummer hinzuaddiert, um die korrekte Sequenznummer zu erhalten. Ist für die Verbindung die Verschlüsselung vereinbart, kann ein Angreifer diesen Offset nicht bestimmen. Weiterhin kann die Segmentlänge durch Hinzufügen von Füllbytes verändert werden. Diese Ergänzungen wurden nicht implementiert, können jedoch leicht hinzugefügt werden.

6.2.2 Integrität

Zur Sicherung der Integrität der übertragenen Daten kommen Signaturverfahren zum Einsatz. Implementiert wurde das Verfahren HMAC-MD5, das auf einem 128 Bit langen Hashwert basiert. Das Verfahren selbst erfüllt die heutigen Anforderungen an kryptographische Verfahren. Die Signatur wurde in der bestehenden Implementierung nur über die Nutzdaten der TCP-Segmente berechnet, so daß ein Verändern der Segmentköpfe mit den Protokollinformationen weiterhin möglich ist. Ein Einbeziehen dieser Informationen in die Signaturberechnung läßt sich jedoch einfach zur Implementierung hinzufügen.

Der Verbindungsaufbau beinhaltet den notwendigen Austausch von kryptographischen Informationen. Da diese Informationen nicht a-priori vorhanden sind, können die übertragenen Segmente des Verbindungsaufbaus nicht gegen Veränderungen durch Angreifer geschützt werden. Der Austausch und das Überprüfen von ausgetauschten Zertifikaten kann jedoch sicherstellen, daß nur ein berechtigter Kommunikationspartner einen vollständigen Verbindungsaufbau durchführen kann. Nach wie vor sind jedoch Angriffe auf die Verfügbarkeit der Kommunikation (*Denial of Service*) möglich.

6.2.3 Authentizität

Das verwendete Signaturverfahren HMAC-MD5 stellt auch die Authentizität des Absender sicher. Da beim Verbindungsaufbau Zertifikate ausgetauscht werden, aus denen sich die einzusetzenden Schlüssel für die Signatur herleiten, können nur die Inhaber

der Zertifikate die übertragenen Segmente korrekt signieren. Ein Einbeziehen der Protokollinformationen aus dem Segmentkopf könnte zusätzlich die Portnummern und die Sequenznummern umfassen, so daß ein Angreifer nicht bereits signierte Segmente mit einem anderen Kopf versehen kann (und sie somit z.B. an einen anderen Port schicken kann). Auch weitere Informationen wie z.B. die IP-Adressen von Sender und Empfänger könnten in die Signatur aufgenommen werden. Dies wurde aus Zeitgründen nicht in den Prototyp integriert, läßt sich jedoch ohne großen Aufwand nachträglich einfügen.

6.2.4 Verfügbarkeit

Im Falle des Signierens der übertragenen Dateneinheiten kann ein Einschleusen verfälschter Segmente bemerkt werden. Angriffe, bei denen eine Verbindung durch Senden von RST-Segmenten unterbrochen werden soll, können folglich erkannt und verhindert werden. Da jedoch auch unsignierte RST-Segmente vearbeitet werden sollten, um den tatsächlichen Abbruch einer Verbindung im Fehlerfalle (z.B. beim Systemneustart) zu erkennen, muß eine entsprechende Heuristik verwendet werden, um über den Abbruch einer Verbindung zu entscheiden (siehe Abschnitt 4.3.4, Seite 88). Diese wurde jedoch im Prototyp noch nicht umgesetzt, entsprechend sind diese Angriffe auf die Verfügbarkeit nach wie vor möglich. Da die Signaturen wesentlicher Bestandteil solcher Heuristiken sind, lassen sich Angriffe auf die Verfügbarkeit während des Verbindungsaufbaus nicht ausschließen, da zu diesem Zeitpunkt noch keine Schlüssel für das Signieren ausgehandelt wurden.

6.2.5 Weitere Aspekte der Sicherheit

Ein wesentlicher Vorteil der Integration der Sicherheitsdienste in den Betriebssystemkern liegt darin, daß diese nicht vom Anwender umgangen werden können. Werden Sicherheitsfunktionen in Anwendungen integriert, kann der Benutzer diese leicht durch den Einsatz von herkömmlichen, ungesicherten Anwendungen (als z.B. `telnet` statt `ssh`) umgehen. Im erstellten Prototyp werden Sicherheitspräferenzen der Anwendung mit den Sicherheitsvorgaben des Systemverwalters zusammengeführt. Hierbei können Präferenzen der Anwendung nur dann zum Einsatz kommen, wenn diese auch den Sicherheitsvorgaben des Administrators genügen (vgl. Abschnitt 3.4.1 und 4.3.6). Dies bedeutet einen erheblichen Gewinn an Kontrolle durch den Systemverwalter. Das Einhalten einer Sicherheitspolicy kann somit für ein System garantiert werden. Zusätzlich können Anwendungen die Auswahl der einzusetzenden Sicherheitsdienste beeinflussen, um diese an eigene Anforderungen anzupassen.

Weiterhin erhalten die Endsysteme eines lokalen Netzwerkes auf diese Weise eine Firewall-Funktionalität. Die herkömmliche Absicherung des lokalen Netzwerkes durch eine zentrale Firewall bedeutet insbesondere beim zunehmenden Einsatz von Hochgeschwindigkeitsnetzen einen ernstzunehmenden Engpaß. Zudem ist eine zentrale Firewall eine Schwachstelle des Systems, da ein Ausfall der Firewall die gesamte Kommunikation mit externen Rechnern unmöglich macht (*Single Point of Failure*). Durch die auf jedem Endsystem vorhandenen Sicherheitsdienste wird zum einen erreicht, daß bereits die Kommunikation innerhalb des internen Netzwerkes abgesichert wird. Zum anderen

wird der Aufwand der einzusetzenden kryptographischen Verfahren und der Zugriffskontrolle auf die Endsysteme verteilt. Jedes Endsystem prüft selbst, ob eine Verbindung zustande kommen darf und führt aufwendige Funktionen wie z.B. Verschlüsselung eigenständig durch. Die dabei erreichte Performanz genügt durchaus den Ansprüchen heutiger lokaler Netzwerke (siehe Abschnitt 6.3).

6.3 Meßergebnisse

Zur Ermittlung der Effizienz der implementierten Sicherheitsdienste wurden Messungen durchgeführt. Bei diesen wurde in Abhängigkeit von der verwendeten Segmentgröße der erreichte Datendurchsatz gemessen. Die Rechner waren zwei PCs mit Intel Pentium 586/166MHz Prozessor und mit jeweils 64MB Hauptspeicher. Zur Datenübertragung wurde normales Ethernet (IEEE 802.3) benutzt, die dafür verwendeten Netzwerkschnittstellen waren Karten von 3Com, Typ 3C900 Combo. Die Verbindung erfolgte über ein Kabel mit gekreuzten Sende- und Empfangsleitungen (sog. Crossover-Kabel). Der theoretische Maximaldurchsatz über Ethernet beträgt 10Mbit/s. Der Datendurchsatz ist jedoch kleiner, da jede Protokollschicht jeweils ihre Protokolldaten mit überträgt, er erreicht in der Regel maximal etwa 9Mbit/s.

6.3.1 Datendurchsatz von normalem TCP

Zunächst wurde eine Vergleichsmessung unter Einsatz des gewöhnlichen FreeBSD-Kernels (Version 3.4) durchgeführt. Diese ermittelt den maximalen Datendurchsatz zwischen den beiden FreeBSD-Rechnern, der bei Einsatz von normalem TCP möglich ist. Abbildung 6.1 zeigt den gemessenen Durchsatz zwischen den beiden Rechnern `cevap` und `cici` in Abhängigkeit von der verwendeten Nachrichtenlänge.[1] Die Größe der Sende- und Empfangspuffer betrug bei allen gezeigten Messungen 8192 Bytes.

Bei kleinen Segmentgrößen wird der erreichte Durchsatz durch die hohe Zahl an kleinen Segmenten beschränkt. Für das Absenden eines jeden Segmentes ist das mehrfache Generieren von Segmentköpfen (IP-Header und TCP-Header) sowie die Übermittlung der Daten an die Netzwerkschnittstelle nötig. Auf Seiten des Empfängers werden die Daten von der Schnittstellenkarte an das Betriebssystem übergeben, wobei jeweils eine Unterbrechung des Prozessors erfolgt (sog. *Interrupt*). Auch müssen die Segmentköpfe durch die Protokollschichten verarbeitet und entfernt werden. Für jedes Segment ist also ein relativ hoher Rechenzeitaufwand nötig. Bei kleinen Segmentgrößen ist dieser Aufwand sehr groß im Verhältnis zu den übertragenen Datenmengen, der Datendurchsatz ist gering. Mit zunehmender Segmentgröße nimmt die Häufigkeit dieser zeitraubenden Operationen ab, den wesentlichen Engpaß bildet nun die Übertragungsrate des Übertragungsmediums. Der Einbruch zwischen den Segmentlängen von 1460 und 1470

[1] Die Nachrichtenlänge ist hierbei die Länge der Anwendungsdaten in jedem Segment. Bei Einsatz der Sicherheitsfunktionen wurden die in den Nutzdatenteil des Segmentes eingetragenen Kennzeichnungen der Daten (siehe Abbildung 3.12) nicht mitgezählt. Die Nachrichtenlänge wurde beim Aufruf von `netperf` angegeben, so daß in jedem Aufruf von `send()` jeweils eine entsprechende Datenmenge übergeben wurde. Um zu verhindern, daß TCP ein Zusammenfassen mehrerer solcher Datenblöcke zu einem großen Segment vornimmt (Blocking), wurde mittels `ifconfig` die maximale Länge der Dateneinheiten, die an die Netzwerkschnittstelle übergeben werden (*Maximum Transfer Unit* = MTU), entsprechend reduziert.

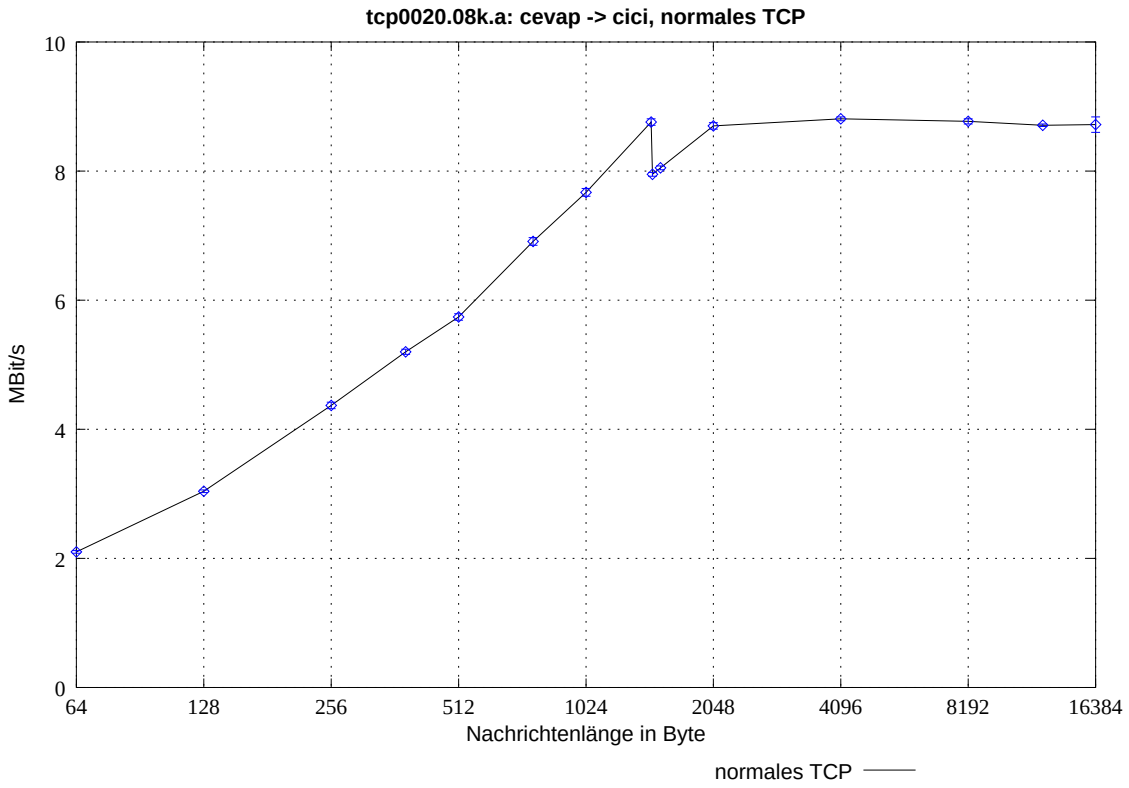


Abbildung 6.1: Datendurchsatz unter Verwendung des normalen TCP-Protokolls

Bytes kommt durch die notwendige Fragmentierung durch das Internet-Protokoll (IP) zustande. Bei 1460 Bytes Datenlänge wird inklusive IP-Header (20 Bytes) und TCP-Header (20 Bytes) die maximale Länge eines Ethernet-Rahmens erreicht (1500 Bytes). Wird das TCP-Segment länger, müssen die generierten Segmente von IP in mehrere kleinere Segmente zerlegt werden. Ein Segment mit 1470 Byte Datenlänge führt also zum Versenden von zwei IP-Paketen, von denen das erste die maximale Länge hat (1500 Bytes = 1460 Datenbytes + 20 Byte TCP-Header + 20 Bytes IP-Header), das zweite jedoch nur die Länge 30 (10 Datenbytes + 20 Byte IP-Header). Jedes zweite Segment ist also nur sehr klein. Bei stark ansteigender Segmentlänge erhöht sich der Anteil der IP-Pakete mit maximaler Länge wieder, bei 2940 Bytes werden z.B. wieder zwei Segmente maximaler Länge generiert. Entsprechend nähert sich der erreichte Datendurchsatz wieder dem Maximum an. Der erreichte maximale Datendurchsatz liegt nur wenig unterhalb des theoretischen Maximums, so daß davon ausgegangen werden kann, daß das Übertragungsmedium selbst den Engpaß bildet, der den Durchsatz beschränkt.

6.3.2 Leistung bei Einsatz der Sicherheitsmechanismen

Bei Einsatz der Sicherheitsdienste ist zu erwarten, daß nur noch ein geringerer Datendurchsatz erreicht werden kann. Hierfür ist der hohe Rechenaufwand bei der Verschlüsselung und der Signierung der Segmente verantwortlich. Weiterhin ist der Vorgang des Generierens und Auswertens der Segmente auf Grund der enthaltenen kryp-

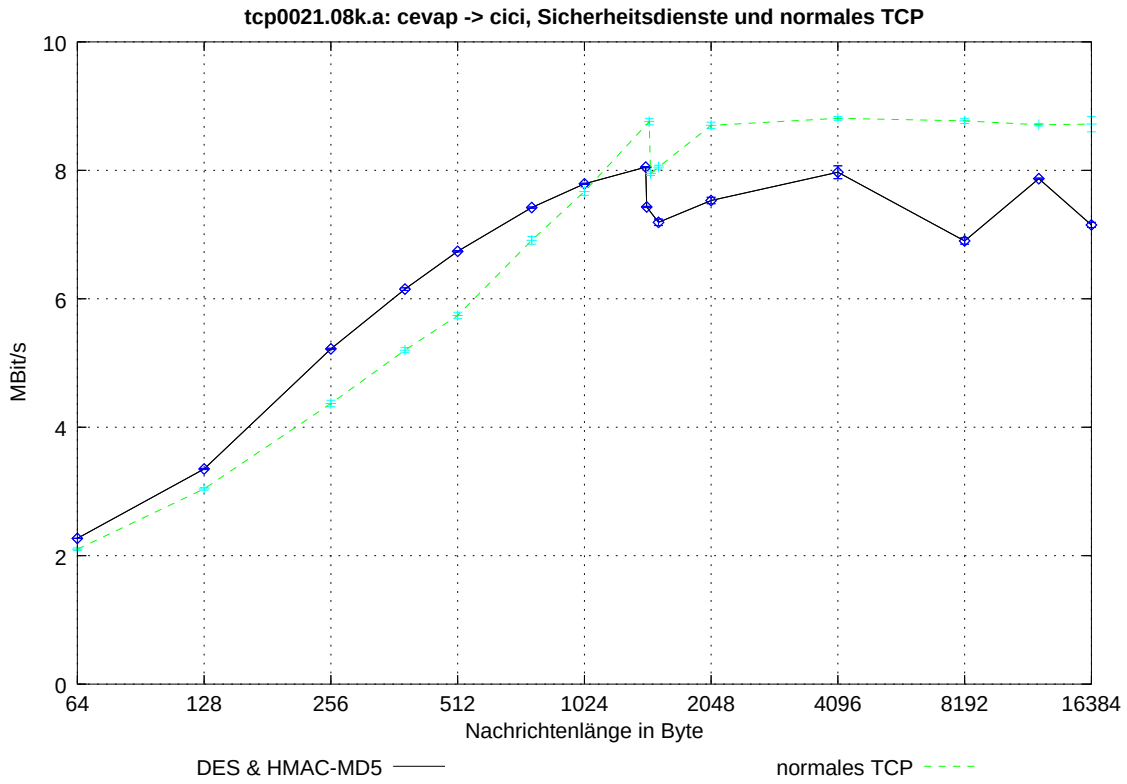


Abbildung 6.2: Datendurchsatz bei Verwendung von DES-Verschlüsselung und HMAC-MD5-Signaturen und bei normalem TCP.

tographischen Informationen aufwendiger. Dies sollte jedoch nur einen kleinen Anteil an der benötigten Rechenleistung ausmachen. Durch das Hinzufügen von kryptographischer Information zu jedem Segment wird der Nutzdatenanteil eines Segmentes geringer. Wird z.B. eine Signatur eingefügt, werden hierfür im Falle von HMAC-MD5 16 Byte für die Signatur selbst und 4 Bytes für deren Kennung benötigt. Ein Segment mit 40 Byte Nutzdaten, das bei normalem TCP zu einem 60 Byte langen Segment führen würde, erreicht somit also schon eine Länge von 80 Bytes. Bei sehr kleinen Segmenten ist dieser Anteil der kryptographischen Daten an der Gesamtlänge recht hoch, so daß ein geringerer Datendurchsatz erwartet werden kann. Da jedoch die notwendigen Unterbrechungen (*Interrupts*) für das Übermitteln der einzelnen Pakete zwischen der Netzwerkhardware und dem Betriebssystem den wesentlichen Aufwand darstellen, sollten sich diese Effekte bei etwas größeren Segmenten kaum bemerkbar machen. Bei großen Segmenten dürfte der Prozessor, der jeweils die Verschlüsselung und die Signierung der Daten vornehmen muß, einen Engpaß darstellen, der den maximalen Datendurchsatz begrenzt.

Abbildung 6.2 zeigt den gemessenen Datendurchsatz bei Einsatz von DES-Verschlüsselung und HMAC-MD5-Signaturen. Zum Vergleich ist die vorige Meßreihe mit normalem TCP mit eingezeichnet. Erwartungsgemäß liegt der Datendurchsatz bei der Verschlüsselung und Signierung von großen Segmenten unterhalb des sonst erreichbaren Wertes. Die Ursache für den Einbruch des Durchsatzes bei Nachrichtenlängen von 8192

Byte konnte nicht geklärt werden. Bei kleinen Nachrichtenlängen von bis zu 1024 Bytes wird ein höherer Durchsatz als beim normalen TCP beobachtet. Diese Leistungssteigerung trotz des hohen Aufwandes für Verschlüsselung und Signierung ist sehr überraschend. Dieses Phänomen soll im folgenden Abschnitt näher erläutert werden.

Höherer Datendurchsatz bei größeren Verzögerungszeiten

Aus Abbildung 6.2 ist ersichtlich, daß der Einsatz von Verschlüsselung und Signierung im Falle von geringen Nachrichtenlängen zu einer Verbesserung des Datendurchsatzes führt. Eine mögliche Ursache für dieses unerwartete Verhalten könnte in einem ausgeglicheneren Fluß von Datensegmenten und zugehörigen Bestätigungen liegen. Dauert z.B. die Verarbeitung von empfangenen Segmenten etwas länger, werden entsprechend später die zugehörigen Bestätigungssegmente generiert. Wenn vor dem Absenden der Bestätigung bereits ein neues Segment beim Empfänger ankommt, wird zunächst der Empfangspuffer gefüllt. Der Empfang mehrerer Segmente kann, wenn diese in der richtigen Reihenfolge ankommen, mit einem einzelnen ACK-Segment bestätigt werden. Ebenso kann der Inhalt mehrerer Segmente in einem Schritt an die Anwendung weitergereicht werden.

Ob genau dies die Ursache für das offenbar bessere Zeitverhalten bei höheren Verarbeitungszeiten ist, konnte nicht ausgemacht werden. Jedoch konnte gezeigt werden, daß die Ursache der Leistungssteigerung tatsächlich nur in den größeren Verzögerungszeiten durch die kryptographischen Verfahren begründet ist. Hierzu wurde ein Betriebssystemkern so modifiziert, daß bei ausgeschalteten Sicherheitsdiensten beim Empfangen und Versenden von Segmenten jeweils eine Warteschleife durchlaufen wird. Allein durch diese Verzögerung konnte in einer Messung der Datendurchsatz für kleine Nachrichtenlängen gesteigert werden. Abbildung 6.3 zeigt die Ergebnisse der entsprechenden Messungen.

Die Messung mit den integrierten Warteschleifen lieferte ein Extrem des zuvor beobachteten Verhaltens: Der Datendurchsatz steigt schneller an, als dies beim Einsatz von Signatur und Verschlüsselung der Fall ist. Er erreicht jedoch bei großen Segmenten nur noch einen geringeren Wert. Dies liegt wahrscheinlich daran, daß die gewählten Wartezeiten länger sind, als dies beim Einsatz der Sicherheitsdienste der Fall ist. Bei großen Segmenten wird oftmals der erreichbare Durchsatz durch das Übertragungsmedium terminiert. Jedoch kann im Falle einer starken Beanspruchung von Rechenleistung der Prozessor einen Engpaß darstellen. Dies scheint hier (stärker noch als bei der Verwendung der Sicherheitsmechanismen) der Fall zu sein, da ein Großteil der Prozessorleistung im Durchlaufen der Warteschleifen gebunden wird. Das Variieren der Wartedauer sowohl beim Empfänger als auch beim Sender zeigte, daß im wesentlichen die Zeitverzögerung auf Seiten des Empfängers die Ursache für dieses Phänomen ist.[2]

Weitere Messungen

Werden die zu übertragenden Daten nur signiert, kommt es zu den im vorigen Abschnitt beschriebenen Effekten, jedoch in einem geringeren Maße. Die durch die Signierung be-

[2]Die entsprechenden Messungen wurden nur als Abschätzung bei der Nachrichtenlänge 512 Bytes bestimmt, die Ergebnisse sind nicht in der vorliegenden Arbeit aufgeführt.

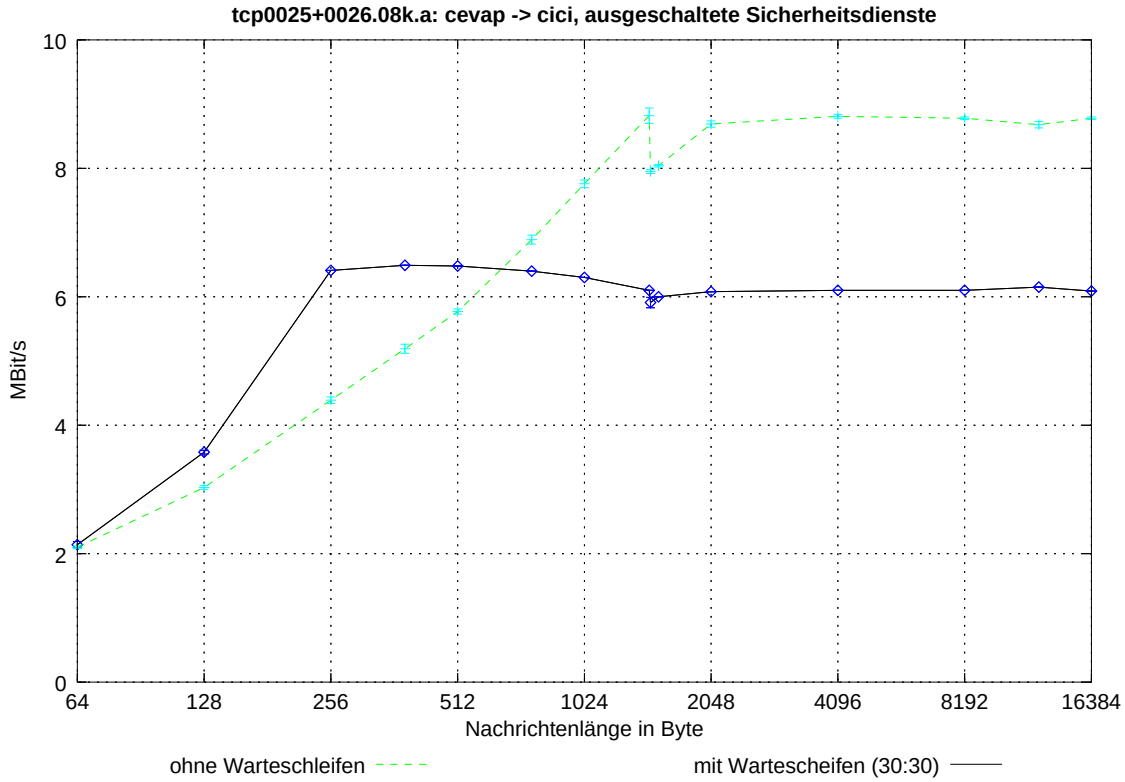


Abbildung 6.3: Datendurchsatz bei ausgeschalteten Sicherheitsmechanismen. Kommen keine Sicherheitsmechanismen zum Einsatz, entspricht der Datendurchsatz dem des normalen TCP (vgl. Abbildung 6.1). Werden Warteschleifen für Empfänger und Sender eingesetzt, um die Zeitverzögerung für die Verschlüsselung zu simulieren, wird ein höherer Datendurchsatz für kleine Nachrichtenlängen gemessen.

dingte Zeitverzögerung alleine ist hierfür offenbar nicht groß genug, sie macht sich nicht so stark bemerkbar wie der gemeinsame Einsatz von Signatur und Verschlüsselung. Erfolgt nur eine DES-Verschlüsselung der Daten, bleibt der Datendurchsatz gemäß der ursprünglichen Erwartung zurück. Offenbar reicht die Verzögerungszeit durch die Verschlüsselung nicht aus, um eine positive Wirkung auf den Datendurchsatz bei kleinen Segmenten zu haben. Zusätzlich zu dieser Verzögerungszeit hat auch der geringere Nutzdatenanteil der Segmente (die Anwendungsdaten werden mit einer Kennung und einer Längenangabe versehen) einen hemmenden Einfluß auf den erreichten Durchsatz. Abbildung 6.4 zeigt die Ergebnisse dieser beiden Messungen.

6.3.3 Zusammenfassung

Es wurde gezeigt, daß im Falle eines Ethernets (IEEE 802.3) die eingesetzten Algorithmen für Verschlüsselung und Signierung nur einen geringen Einfluß auf die Übertragungsleistung der Endsysteme haben. Selbst bei Einsatz von vergleichsweise langsamen Prozessoren (Intel 586/166MHz) war die Prozessorleistung durchaus ausreichend, um die Verschlüsselung und das Signieren von Kommunikationskanälen durch-

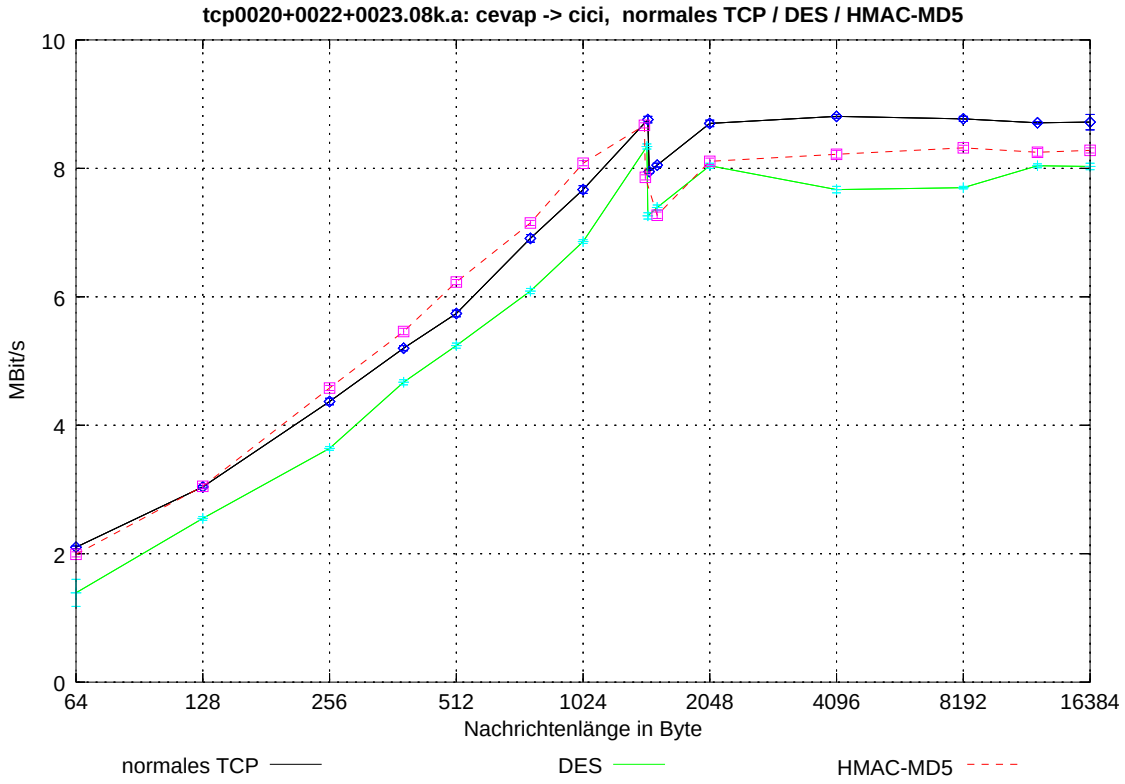


Abbildung 6.4: Datendurchsatz bei Signierung und bei Verschlüsselung der Segmente. Zum Vergleich wurde die Kurve aus Abbildung 6.1 mit eingezeichnet, die den Durchsatz bei unverändertem TCP zeigt.

zuführen. Der erreichte Datendurchsatz von mehr als 7 MBit/s ist auch der guten Performanz des DES-Algorithmus zu verdanken. Neuere Rechner besitzen jedoch ohnehin schon eine wesentlich höhere Prozessorleistung, so daß auch komplexere Algorithmen benutzt werden können. Ebenfalls können auf diese Weise auch schnellere Netzwerktechnologien Verwendung finden. Ggf. können hierbei die Sicherheitsdienste so konfiguriert werden, daß sie nur für einzelne, sicherheitsrelevante Anwendungen eingesetzt werden, um den Prozessor nicht allzusehr zu belasten.

6.4 Ausblick

Die Implementierung von Sicherheitsmechanismen in die TCP-Schicht erfüllt bereits viele der gestellten Sicherheitsanforderungen. Sie ist kompatibel zum normalen TCP, so daß der Einsatz in einer gewöhnlichen Internetumgebung problemlos ist. Wenn möglich finden jedoch die Sicherheitsdienste Verwendung. Der Entwurf stellt sicher, daß die Integration weiterer Algorithmen und Funktionen möglich ist. Der Prototyp verwendet noch keine Zertifikate und kein kryptographisches Schlüsselaustauschverfahren. Diese Erweiterungen sind zunächst die wichtigsten, um wirkliche Sicherheit zu garantieren. Weitere Algorithmen sollten noch integriert werden, um eine höhere Flexibilität zu

erreichen. Das verwendete Verschlüsselungsverfahren DES sollte durch bessere Algorithmen ergänzt werden, ebenfalls sollte nicht der ECB-Modus verwendet werden. Eine Erweiterung des Protokolls um einen Eintrag für einen Initialisierungsvektor würde es leicht ermöglichen, andere Modi einzusetzen. Die Verschlüsselungsfunktionen für DES und Triple-DES im CBC-Modus sind bereits in der Implementierung vorhanden, so daß der Verwendung dieser Verfahren nicht mehr viel im Wege steht. Die spezifizierten Kennungen der Algorithmen (siehe Abbildung 3.13) müßten entsprechend um neue ergänzt werden. Um die Verbreitung der vorgestellten Sicherheitsdienste zu ermöglichen, sind Implementierungen auf weiteren Plattformen notwendig. Eine Portierung auf andere BSD-Derivate oder Linux sollte relativ einfach möglich sein, wenn auch die notwendigen Änderungen an den jeweiligen Betriebssystemkernen nicht trivial sind. Andere Portierungen könnten jedoch eine fast vollständige Neuimplementierung erfordern.

Allgemein zeigt der erstellte Prototyp, daß die Integration von Sicherheitsmechanismen in die Transportschicht geeignet ist, anwendungsspezifische Sicherheit zu gewährleisten. Dies kann völlig transparent für die Anwendungen geschehen, diese können jedoch auch Einfluß auf die Auswahl der eingesetzten Algorithmen nehmen. Im Gegensatz zum Ansatz von IPsec, wo bereits in der Vermittlungsschicht eine anwendungsspezifische Konfiguration erfolgen kann und somit die Schichtstruktur des Protokollstapels durchbrochen wird, stellt die Transportschicht in heutigen Systemen ohnehin den Übergang zum Anwendungssystem dar. Die Implementierung paßt sich somit viel natürlicher in die Systemumgebung ein. Weitere Bestrebungen sollten sich also eher auf die Transportschicht konzentrieren.

Die für Sicherheitsmechanismen notwendigen kryptographischen Funktionen stehen mittlerweile in einer Vielzahl von Implementierungen für Anwendungen zur Verfügung. Dies geschieht gewöhnlich in Form von Bibliotheken, die beim Start von Anwendungen dynamisch geladen werden. Betriebssystemkerne haben jedoch i.d.R. keinen Zugriff auf solche Funktionsbibliotheken, was die Integration von Sicherheitsmechanismen in den Kernel erschwert. Die Erkenntnis, daß diese Verfahren zunehmend wichtiger werden und deshalb bereits in den Betriebssystemkern integriert werden müssen, setzt sich erst allmählich durch. Im Betriebssystem OpenBSD wurden bereits einige dieser Mechanismen (Hash-Funktionen, Zufallszahlengenerator und Treiber für Kryptoprozessoren) in den Kernel integriert. Andere Betriebssysteme werden diesem Beispiel folgen, so daß die Integration von Sicherheitsdiensten zunehmend einfacher wird. Insbesondere wird die zunehmende Verbreitung von Kryptoprozessoren die Absicherung von Kommunikationsverbindungen zur Normalität werden lassen, da auf diese Weise auch aufwendige kryptographische Verfahren in Netzwerken mit hohen Übertragungsraten eingesetzt werden können.

Literaturverzeichnis

[Abadí 1994] Martin Abadí, Roger Needham. *Prudent Engineering Practice for Cryptographic Protocols*. In: Proceedings of the 1994 IEEE Symposium on Security and Privacy, IEEE Comput. Soc. Press, May 1994. Ebenfalls erschienen als SRC Research Report 125. `http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-125.html`

[Adleman 1982] Leonard M. Adleman. *On Breaking The Iterated Merkle-Hellman Public-Key Cryptosystem*. Advances in Cryptology, Crypto '82 Proceedings, Springer Verlag, Berlin, 1982.

[Atkinson 1995] R. Atkinson. *IP Encapsulating Security Payload (ESP)*. RFC 1827.

[Aziz 1997] Ashar Aziz, Tom Markson, Hemma Prafullchandra. *Simple Key-Management for Internet Protocols (SKIP)* Technical Specification, 24.04.1997. `http://www.skip-vpn.org/spec/SKIP.html`

[Bellovin 1989] Steven M. Bellovin. *Security Problems in the TCP/IP Protocol Suite*. Computer Communication Review, Vol. 19, Nr. 2, 1989. `http://www.deter.com/unix/papers/tcpip_problems_bellovin.ps.gz`

[Bellovin 1996] Steven M. Bellovin. *Problem Areas for the IP Security Protocols*. Proceedings: Sixth USENIX Security Symposium, USENIX Association, Berkeley, CA, USA, 1996. `ftp://ftp.research.att.com/dist/smb/badesp.ps`

[Benecke 1998] Carsten Benecke, Uwe Ellermann. *Nutzung von Kryptographie im Zusammenhang mit Firewalls*. DFN-Bericht Nr. 86, Verein zur Förderung eines Deutschen Forschungsnetzes e.V., Berlin, 1998.

[Bogen 1998] Manfred Bogen, Michael Lenz, Andreas Reichpietsch, Peter Simons. *Securing Ordinary TCP Services Through Tunnels*. Proceedings: INET'98, The Internet Society, Genf, Schweiz, 1998. `http://www.isoc.org/inet98/proceedings/6h/6h_1.htm`

[Borman 1993] D. Borman. *Telnet Authentication Option*. RFC 1416.

[Braden 1989] R. Braden (Editor). *Requirements for Internet Hosts – Communication Layers*. RFC 1122, Oktober 1989.

[Braden 1994] R. Braden. *T/TCP – TCP Extensions for Transactions. Functional Specification*. RFC 1644.

[Brickell 1984] Ernest F. Brickell. *Breaking Iterated Knapsacks*. Advances in Cryptology, Crypto '84 Proceedings, Springer Verlag, Berlin, 1984.

[Caronni 1996] Germano Caronni, Hannes Lubich, Ashar Aziz, Tom Markson, Rich Skrenta. *SKIP - Securing the Internet*. Proceedings: 5th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises 1996 (WET ICE '96), Stanford, CA. IEEE Comput. Soc. Press, 1996, S. 62-67.

[CCITT 1991] CCITT. *Recommendation X800: Security Architecture for Open Systems Interconnection for CCITT Applications*. ITU, International Telegraph and Telephone Consultative Committee, Genf, 1991.

[CCITT 1996] CCITT. *Recommendation X800, Amendment 1: Layer Two Security Service and Mechanisms for LANs*. ITU, International Telegraph and Telephone Consultative Committee, 1996.

[CCITT 1997] CCITT. *Recommendation X509: The Directory: Authentication framework*. ITU, International Telegraph and Telephone Consultative Committee, 1997.

[Chapman 1995] D. Brent Chapman, Elizabeth D. Zwicky. *Building Internet Firewalls*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1995.

[Cheswick 1996] William R. Cheswick, Steven M. Bellovin. *Firewalls und Sicherheit im Internet*. Addison Wesley, Bonn, 1996.

[Davies 1982] Donald W. Davies, Graeme I.P. Parkin. *The average cycle size of the key stream in output feedback encipherment*. Cryptography, Eurocrypt '82 Proceedings, Springer Verlag, Berlin, 1982.

[Dierks 1999] T. Dierks, C. Allen. *The TLS Protocol, Version 1.0*. RFC 2246.

[Diffie 1976] Whitfield Diffie, M. E. Hellman. *New Directions in Public Key Cryptography*. IEEE Transactions on Information Theory, Vol. IT-22, Nr. 6, November 1976, S. 644 - 654.

[Diffie 1985] Whitfield Diffie. *Security for the DoD Transmisssion Control Protocol*. Advances in Cryptology, Crypto'85 Proceedings, Springer Verlag, Berlin, 1985.

[Diffie 1992] Whitfield Diffie. *The First Ten Years of Public Key Cryptography*. In: [IEEE 1992].

[Estrin 1991] Deborah Estrin, Gene Tsudik. *An End-to-End Argument for Network Layer, Inter-Domain Access Controls*. in: Ïnternetworking: Research and Experience", Vol. 2, Nr. 2, S. 71-85 (Juni 1991). `ftp://ftp.cert.dfn.de/pub/docs/net/e2e.ps.gz`

[Fielding 1999] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, Juni 1999.

[Freier 1996] Alan O. Freier, Philip Karlton, Paul C. Kocher. *The SSL Protocol - Version 3.0*. Internet Draft, März 1996. `ftp://ftp.netscape.com/pub/review/ssl-spec.tar.Z`

[Fujie 1998] Masanori Fujie, Jun-ichiro Itoh, Takashi Tochihara, Hiroo Shirasaki, Kazumasa Utashiro. *Study from Hybrid Implementation of SwIPe and IPsec*. in: Proceedings INET'98, The Internet Society, Genf, Schweiz, 1998 `http://www.isoc.org/inet98/proceedings/6h/6h_2.htm`

[Goldschlag 1999] David M. Goldschlag, Michael G. Reed, Paul F. Syverson. *Onion Routing for Anonymous and Private Internet Connections*. Communications of the ACM, Vol. 42, Nr. 2, Februar 1999. `http://www.onion-router.net/Publications/CACM-1999.ps.gz`

[Großklaus 1999] Axel Großklaus. *Policy, Vorfallsbearbeitung, Schwachstellenanalyse*. In: [Mück 2000].

[Heffernan 1998] A. Heffernan. *Protection of BGP Sessions via the TCP MD5 Signature Option*. RFC 2385.

[Huitema 1996] Christian Huitema. *IPv6: The new Internet Protocol*. Prentice Hall, 1996.

[Hunt 1992] Craig Hunt. *TCP/IP Network Administration*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1992.

[IEEE 1992] Gustavus J. Simmons (Hrsg). *Contemporary Cryptology*. Institute of Electrical and Electronics Engineers, Inc., New York, NY, 1992.

[Ioannides 1993] John Ioannides, Matt Blaze. *The Architecture and Implementation of Network-Layer-Security under Unix*. Proceedings: Fourth USENIX Security Symposium, USENIX Association, Berkeley, CA, USA, 1993. `ftp://ftp.research.att.com/dist/mab/swipeusenix.ps`

[Jacobson 1992] V. Jacoboson, R. Braden, D. Borman. *TCP Extensions for High Performance*. RFC 1323.

[Joncheray 1995] Laurent Joncheray. *A Simple Active Attack Against TCP*. Proceedings: Fifth USENIX Security Symposium, USENIX Association, Berkeley, CA, USA, 1995. `http://www.deter.com/unix/papers/tcp_attack.ps.gz`

[Jordan 1993] Francisco Jordan, Manuel Medina. *A Complete Secure Transport Service in the Internet*. Technical Report UPC-DAC-93-22, Departament Arquitectura de Computadors, Universitat Polytècnica de Catalunya, 1993. `ftp://ftp.ac.upc.es/pub/reports/DAC/1993/UPC-DAC-93-22.ps.Z`

[Jueneman 1982] Robert R. Jueneman. *Analysis of certain aspects of output feedback mode*. Advances in Cryptology, Crypto '82 Proceedings, Springer Verlag, Berlin, 1982.

[Kaliski 1992] B. Kaliski. *The MD2 Message-Digest Algorithm*. RFC 1319.

[Kaufman 1995] Charlie Kaufman, Radia Perlman, Mike Speciner. *Network Security - Private Communication in a Public World*. Prentice Hall Inc., Englewood Cliffs, New Jersey, USA, 1995.

[Kelsey 1997] John Kelsey, Bruce Schneier, David Wagner. *Protocol Interactions and the Chosen Protocol Attack*. Security Protocols, 5th International Workshop April 1997 Proceedings. Springer-Verlag, 1998, S. 91-104. `http://www.counterpane.com/chosen_protocol.html`

[Kent 1998a] Stephen T. Kent, R. Atkinson. *Security Architecture for the Internet Protocol*. RFC 2401.

[Kent 1998b] Stephen T. Kent, R. Atkinson. *IP Authentication Header*. RFC 2402.

[Kent 1998c] Stephen T. Kent, R. Atkinson. *IP Encapsulating Security Payload (ESP)*. RFC 2406.

[Kerner 1992] Helmut Kerner. *Rechnernetze nach OSI*. Addison Wesley, Bonn, 1992.

[King 1990] Greg King. *A Survey of Commercially Available Secure LAN Products*. Proceedings: Fifth Annual Computer Security Applications Conference 1989, Tucson, AZ. IEEE Comput. Soc. Press, 1990

[Krawczyk 1997] H. Krawczyk, M. Bellare, R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104.

[Kyas 1998] Othmar Kyas. *Sicherheit im Internet*. International Thomson Publishing, Bonn, 1998.

[Liedtke 2000] Britta Liedtke, Ingmar Camphausen. *Public-Key-Zertifizierung und Public-Key-Infrastrukturen*. In: [Mück 2000].

[Lunt 1991] S. Lunt. *FTP Security Extensions*. RFC 2228.

[McCurley 1996] Kevin S. McCurley. *Cryptography and the Internet: Lessons and Challenges*. Advances in Cryptology, ASIACRYPT '96 Proceedings, Springer Verlag, Berlin, 1996.

[Morris 1985] R. Morris *A Weakness in the 4.2BSD UNIX TCP/IP Software*. Computing Science Technical Report No 117, AT&T Bell Laboratories, 1985. `http://www.deter.com/unix/papers/bsd_tcpip_weakness_morris.ps.gz`

[O'Malley 1991] S. O'Malley, L. Peterson. *TCP Extensions considered harmful*. RFC 1263.

[Matthis 1996] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow. *TCP Selective Acknowledgement Options*. RFC 2018.

[Maughan 1998] D. Maughan, M. Schertler, M. Schneider, J. Turner. *Internet Security Association and Key Management Protocol (ISAKMP)*. RFC 2408.

[Mitchell 1992] C.J. Mitchell, F. Piper, P. Wild. *Digital Signatures*. In: [IEEE 1992].

[Moy 1998] J. Moy. *OSPF Version 2*. RFC 2328.

[Mück 2000] Hans-Joachim Mück, Carsten Benecke, Stefan Kelm (Hrsg). *Sicherheit in vernetzten Systemen*. Bericht 224, Fachbereich Informatik, Universität Hamburg. Hamburg, 2000.

[Nechvatal 1992] James Nechvatal. *Public Key Cryptography*. In: [IEEE 1992]

[Nelson 1990] Ruth Nelson. *End-to-End Encryption at the Network Layer*. Proceedings: Fifth Annual Computer Security Applications Conference 1989, Tucson, AZ. IEEE Comput. Soc. Press, 1990

[Newman 1999] C. Newman. *Using TLS with IMAP, POP3 and ACAP*. RFC 2595.

[NIST 1995] National Institute of Standards and Technology (NIST). *FIPS-180-1 'Secure Hash Standard'*. U.S. Department Of Commerce, USA, 1995. `http://csrc.nist.gov/fips/fip180-1.pdf`

[Parker 1990] Richard L. Parker II. *Layer 2 security services for local area networks*. Proceedings: 13th National Computer Security Conference. Information Systems Security. Standards - the Key to the Future. Washington, DC, USA, 1990. National Institute of Standards, 1990. Vol. 1, S. 296-306.

[Poon 1990] F.S.F. Poon, M.S. Iqbal. *A Physical Layer System for IEEE 802.3 / (ISO 8802/3) Protocol Based Local Area Networks*. Proceedings: Bilkent International Conference on New Trends in Communiation, Control and Signal Processing, 1990, Ankara, Türkei. Elsevier, Niederlande, 1990, Vol. 2.

[Poon 1992] F.S.F. Poon, M.S. Iqbal. *Design of a physical layer security mechanism for CSMA/CD networks*. IEEE Proceedings I (Communications, Speech and Vision) 1992, Vol. 139, No. 1, S. 103-112.

[Postel 1981] Jon Postel. *Transmission Control Protocol*. RFC 793.

[Ramaswamy 1989] Raju Ramaswamy. *Security Architecture for Data Transfer through TCP/IP Protocols*. Computers & Security, Vol. 8. Elsevier Science Publishers Ltd., Amsterdam, 1989.

[Ramaswamy 1990a] Raju Ramaswamy. *A Security Architecture and Mechanism for Data Confidentiality in TCP Protocols*. Proceedings: IEEE Computer Society Symposium on Research in Security and Privacy 1990, Oakland, CA, 1990.

[Ramaswamy 1990b] Raju Ramaswamy. *Traffic Flow Confidentiality Security Service in OSI Computer Network Architecture*. Proceedings: IEEE Tencon'90, Hong Kong. Vol. 2, S. 649-652.

[Reed 1998] Michael G. Reed, Paul F. Syverson, David M. Goldschlag. *Anonymous Connections and Onion Routing*. IEEE Journal on Selected Areas in Communication, Special Issue on Copyright and Privacy Protection, 1998. `http://www.onion-router.net/Publications/JSAC-1998.ps.gz`

[Reinolds 1994] J. Reynolds, Jon Postel. *Assigned Numbers*. RFC 1700. Siehe auch: `http://www.iana.org/numbers.html`

[Rivest 1992a] Ron L. Rivest. *The MD4 Message-Digest Algorithm*. RFC 1320.

[Rivest 1992b] Ron L. Rivest. *The MD5 Message-Digest Algorithm*. RFC 1321.

[Rueppel 1992] Rainer A. Rueppel. *Stream Ciphers*. In: [IEEE 1992].

[Saltzer 1984] J. H. Saltzer, D. P. Reed, D. D. Clark. *End-To-End Arguments in System Design*. ACM Transactions on Computer Systems, 1984, Vol. 2, Nr. 4, S. 277-288.

[Schneider 1991] Hans-Jochen Schneider (Hrsg.). *Lexikon der Informatik und Datenverarbeitung*. Oldenbourg Verlag, München, 1991.

[Schneier 1996] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Viley & Sons, New York, 2nd Edition, 1996.

[Schneier 1998] Bruce Schneier. *Cryptographic Design Vulnerabilities*. IEEE Computer, Vol. 31, Nr. 9, Sep 1998, S. 29-33. `http://www.counterpane.com/design-vulnerabilities.pdf`

[Schuba 1997] Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram, Diego Zamboni. *Analysis of a Denial of Service Attack on TCP*. In: Proceedings of the 1997 IEEE Symposium on Security and Privacy, S. 208-223, IEEE Comput. Soc. Press, Mai 1997.

[Schuchmann 1982] H.R. Schuchmann. *Enigma Variations*. Advances in Cryptology, Eurocrypt '82 Proceedings, Springer Verlag, Berlin, 1982.

[Simmons 1992] Gustavus J. Simmons. *A Survey of Information Authentication*. In: [IEEE 1992].

[Smith 1997] Richard E. Smith. *Internet Cryptography*. Addison Wesley, Reading, Massachussets, USA, 1997.

[Stallings 1995] William Stallings. *Sicherheit im Datennetz*. Prentice Hall, München, 1995.

[Stallings 1999] William Stallings. *Cryptography and Network Security*. Prentice Hall, New Jersey, 1999.

[Stevens 1994] W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, Reading, Massachusetts, 1994.

[Stevens 1995] W. Richard Stevens, Gary R. Wright. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison Wesley, Reading, Massachusetts, 1995.

[Tanenbaum 1990] Andrew S. Tanenbaum. *Computer Netzwerke*. 2. Auflage, Wolfram's Fachverlag, Deutschland, 1990.

[Terplan 1992] Kornel Terplan. *Communication Networks Management*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1992.

[Tsutsumi 1995] Toshiyuki Tsutsumi, Suguru Yamaguchi. *Secure TCP – Providing Security Functions in TCP Layer*. Proceedings: Inet '95 Conference, Internet Society, Reston, VA, USA, 1995. `http://www.isoc.org/HMP/PAPER/144/`

[Voydock 1983] Victor L. Voydock, Stephen T. Kent. *Security Mechanisms in High-Level Network Protocols*. Computing Surveys, Vol. 15, Nr. 2, 1983.

[Voydock 1985] Victor L. Voydock, Stephen T. Kent. *Security Mechanisms in a Transport Layer Protocol*. Computers & Security, Vol. 4, Elsevier Science Publishers B.V., Amsterdam, 1985.

[Wiener 1994] M. J. Wiener. *Efficient DES Key Search*. Technical Report TR-244, School of Computer Science, Carleton University, Ottawa, Canada, Mai 1994. `ftp://ftp.ox.ac.uk:/pub/crypto/cryptanalysis/des_key_search.ps.gz`

[Ylönen 1996] Tatu Ylönen. *SSH - Secure Login Connections over the Internet*. Proceedings: Sixth USENIX Security Symposium, USENIX Association, Berkeley, CA, USA, 1996. `ftp://cag.lcs.mit.edu/pub/dm/papers/ylonen:ssh-usenix.ps.gz`