Rheinische
Friedrich-Wilhelms-Universität
Bonn

Diplomarbeit

Sicherer Dokumentenaustausch auf Basis des World-Wide-Web

von
Georg Bißeling

Institut für Informatik
Abteilung III
Betreuer: Prof. Dr. A. B. Cremers

Bonn, im Juli 1996

Danksagung

Meinen Eltern und meiner Tante möchte ich für ihre Unterstützung meines Studiums bis ins hohe Alter danken.

Diplom-Informatiker Elmar Haneke und Diplom-Informatiker Alexander Horz danke ich für ihre Zeit, ihre Geduld und viele hilfreiche Hinweise.

Prof. Dr. Cremers danke ich für seine Bereitschaft, meinen Themenwunsch aufzugreifen und meine Arbeit zu betreuen.

Karen danke ich auch für das Korrekturlesen, mehr noch für ihr Verständnis und ihre moralische Unterstützung.

Inhaltsverzeichnis

Einleitung

Bislang wird Dokumentenaustausch in organisatorisch oder geographisch verteilten Arbeitsgruppen in der Regel mittels Elektronischer Post durchgeführt. Erfahrungsgemäß ist diese Form des Dokumentenaustausches keine angemessene Lösung. Die Größe einzelner Elektronischer Nachrichten und der erlaubte Zeichensatz sind oft auf 64 Kilobyte bzw. 7 Bit eingeschränkt. Um beliebige Dokumente über Elektronische Post auszutauschen, ist man deshalb gezwungen, Programme wie `uuencode`, `split` und `merge`[1] zu benutzen. Diese Programme stehen nicht unter einer ergonomischen Oberfläche zur Verfügung und sind nur von versierten Benutzern beherrschbar.

Alternativ wird auch `ftp` (File Transfer Protocol) zum Dokumentenaustausch benutzt. Ergonomisch stellt diese Lösung keinen Fortschritt dar. Hier liegt der entscheidende Mangel insbesondere in der fehlenden Übersicht über Veränderungen an den gemeinsam genutzten Dokumenten.

Dieser Mangel bewirkt unter Umständen das gleichzeitige Bearbeiten eines Dokuments durch zwei oder mehrere Personen. Sobald mehrere Bearbeiter jeweils ihre Aktualisierung des Dokuments unkoordiniert zurückschreiben (engl.: *race condition*), gehen alle Änderungen, bis auf die zuletzt geschriebenen, verloren.

BSCW (Basic Support for Cooperative Work[2]) stellt unter einer einheitlichen Oberfläche eine komfortable Unterstützung für einen gemeinsamen Dokumenten-Pool, den sog. *Workspace*, zur Verfügung. BSCW bietet u.a. folgende Grundfunktionalität: das *Hinzufügen* eines Dokuments, das *Löschen* eines Dokuments, das *Ändern* eines Dokuments mit Verwaltung der bisherigen Versionen sowie die *Verwaltung* der Teilnehmer einer Arbeitsgruppe.

BSCW ist auf der Basis des World-Wide-Web (WWW oder W3) realisiert. Der Benutzer interagiert mit dem Workspace über einen sogenannten *WWW-Browser*. Geeignete WWW-Browser verschiedener Anbieter stehen für nahezu alle denkbaren Arbeitsplatzrechner, Betriebssysteme und Benutzeroberflächen zur Verfügung (Windows (3.x, 95, NT), OS/2 mit Workplace Shell, Apple Macintosh mit Finder, Unix Varianten mit X11). Die verschiedenen Implementierungen von WWW-Browsern entsprechen der Bedienphilosophie der jeweiligen graphischen Oberfläche. BSCW selbst ist *plattformunabhängig*. Die Abbildung 1.1 und die folgenden ab Seite 8 zeigen den WWW-Browser *Netscape Navigator* als Zugang zu BSCW.

BSCW stellt die benötigte Grundfunktionalität für den Dokumentenaustausch unter einer einfach bedienbaren, plattformunabhängigen Oberfläche zur Verfügung. Möglich wird dies durch Rückgriff auf Standardschnittstellen und -software.

Ziel der Diplomarbeit

BSCW ist unsicher. Ziel der Diplomarbeit ist es, einen Sicherheitsmechanismus für BSCW zu entwickeln.

Die verwendeten WWW-Browser und HTTP-Server[3] bieten einen zu einfachen Passwortmechanismus. Der unauthorisierten Einsichtnahme und Manipulation wird dadurch kaum

[1] Beschreibungen der erwähnten Programme finden sich in praktisch jedem Unix Manual, siehe z.B. [1].

[2] BSCW wurde beim Forschungszentrum Informationstechnik GmbH (GMD) entwickelt.

[3] HTTP: HyperText Transfer Protocol, das für die Kommunikation zwischen Server und WWW-Browser verwandte Protokoll [2].

entgegengewirkt. Der Transfer der Dokumente erfolgt unverschlüsselt. Benutzernamen und Passworte werden ebenfalls unverschlüsselt über das Netz gesandt[4].

Die Passworte werden lediglich *uuencoded*. Dieses Verfahren ist strenggenommen nicht als Verschlüsselung zu betrachten, da der Klartext leicht wiedergewonnen werden kann. Davon abgesehen ist die stets gleiche „Verschlüsselung" der Passworte auch deshalb nutzlos, weil ein Angreifer die verschlüsselte Form abhören und benutzen könnte, um sich Zutritt zu verschaffen (*replay attack*).

Der Diplomarbeit liegt folgende Hypothese zugrunde:

> Mit einem *Public Key*-Kryptosystem kann man BSCW gegen nicht authorisierte Einsichtnahme und Manipulation absichern.

Aus den derzeit zur Verfügung stehenden Verschlüsselungsverfahren und Protokollen werden geeignete ausgewählt und kombiniert. Ziel ist ein möglichst einfacher, portabler und erweiterungsfähiger Sicherheitsmechanismus. Ergebnis ist ein Entwurf, der sich durch die Benutzung von vorhandenen Standardkomponenten realisieren läßt.

Textübersicht

Kapitel 1

Die vorliegende Arbeit besteht aus vier Kapiteln. Im ersten Kapitel wird BSCW vorgestellt und kurz in seine Bedienung eingeführt, um ein intuitives Verständnis des BSCW-Systems zu vermitteln.

Dann werden einige Kriterien angeführt, mit deren Hilfe sich Software für die Unterstützung der Gruppenarbeit charakterisieren läßt. BSCW wird mit Hilfe dieser Kriterien im Groupware-Kontext positioniert. Dieser Abschnitt orientiert sich an der in [7] entwickelten Systematik.

Schließlich werden die Sicherheitsanforderungen von BSCW unter Berücksichtigung des Einsatzzwecks und der verwendeten Technologie untersucht und die Verwendung kryptographischer Methoden zu ihrer Erfüllung wird motiviert.

Kapitel 2

Das gesamte zweite Kapitel dient der Vorbereitung auf die Aufgabe der Beschreibung und Bewertung verschiedener Protokollvorschläge für die Verschlüsselung elektronischer Kommunikation.

Hierzu werden zunächst die Grundbausteine für diese Protokolle vorgestellt. Den Anfang machen die symmetrischen Verfahren. Es werden nur drei prominente Vertreter dieser Verfahrensklasse skizziert. Aus den grundsätzlichen Schwächen symmetrischer Verfahren wird die Suche nach den asymmetrischen oder *Public Key*-Verfahren begründet. Auch hier werden neben dem RSA-Verfahren nur drei andere Vertreter dieser Verfahrensklasse beschrieben.

Sichere Hashfunktionen sind der letzte algorithmische Baustein, der für ein kryptographisches Protokoll benötigt wird. Nun werden digitale Unterschriften, Zertifikate und Zertifizierungshierarchien eingeführt.

Mit diesen Verfahren und Begriffen wird ein Prototyp eines kryptographischen Protokolls konstruiert, der in ähnlicher Form den meisten Protokollvorschlägen zugrundeliegt.

Der letzte Vorbereitungsschritt besteht in der Bewertung verschiedener Möglichkeiten zur Positionierung kryptographischer Protokolle im sogenannten Protokoll-Stack der Netzwerkkommunikation.

Nachdem der Vorbereitung soviel Raum gegeben wurde, kann die eigentliche Beschreibung der Protokollvorschläge relativ kompakt erfolgen. Für jeden Vorschlag wird angegeben, welche Verfahren benutzt werden, welche Abweichungen vom Prototyp bestehen und wo der

[4] Verwendet wird das im Internet benutzte TCP/IP Protokoll, das keinerlei Schutz gegen Lauschangriffe und Datenmanipulation bietet [3], [6].

Vorschlag im Protokoll-Stack positioniert ist. Das Kapitel endet in einer tabellarischen Aufstellung der wesentlichen Merkmale der beschriebenen Protokollvorschläge.

Kapitel 3

Das dritte Kapitel beschreibt den Entwurf und Implementierung, sowie die Auswahl des verwendeten Protokolls. Der erste Abschnitt zeigt die Entwicklung eines Entwurfs, der sehr stark an den einfach zugänglichen Erweiterungsmöglichkeiten der Web-Browser orientiert ist. Dieser Entwurf wurde im Laufe der Arbeit verworfen, da seine Implementierung sich als unelegant und wenig leistungsfähig erwies.

Der im nächsten Abschnitt beschriebene, zweite Entwurf führte zu einer von ihrer Sicherheit her akzeptablen Implementierung. Im Gegensatz zum ersten Entwurf kann dieser allein unter Verwendung von standardisierten Komponenten und Schnittstellen realisiert werden. Leider kann die Übertragungsleistung der ersten Implementierung nicht überzeugen.

Im dritten Abschnitt werden die Ursachen für die mangelnde Leistung gesucht und analysiert. Eine verbesserte Implementierung wird entworfen und realisiert. Der Abschnitt endet mit einer Beschreibung der verbesserten Implementierung und einem Erfolgsnachweis.

Kapitel 4

Das letzte Kapitel faßt noch einmal die geleistete Arbeit und die erreichten Ziele unter Berücksichtigung der im ersten Kapitel entwickelten Sicherheitsanforderungen zusammen.

Der erste Abschnitt beschreibt mögliche Weiterentwicklungen des zweiten Entwurfs und seiner Implementierung, die seinen Komfort, seine Geschwindigkeit und seine Sicherheit verbessern könnten.

Im letzten Abschnitt werden einige Gedanken zur Zukunft kryptographischer Verfahren in der elektronischen Kommunikation entwickelt. Hier bleibt genug Raum für Spekulationen. Die Arbeit klingt aus mit einer optimistischen Einschätzung von Philip Zimmermann, dem Autor des Programms *pgp*, die den kryptographischen Verfahren eine unaufhaltsame Verbreitung prophezeit.

Kapitel 1

BSCW (Basic Support for Cooperative Work)

Dieses Kapitel führt in seinem ersten Abschnitt kurz in die Bedienung des BSCW-Systems ein, um ein intuitives Verständnis der Möglichkeiten von BSCW zu ermöglichen.

Der zweite Abschnitt setzt BSCW zu den Begriffen Groupware, Workflow und zum Forschungsgebiet *computer support for cooperative work* (CSCW) in Beziehung. Hier wird eine grobe Orientierung über die verschiedenen in [7] vorgeschlagenen Systemklassen gegeben.

Im letzten Abschnitt des Kapitels werden die Sicherheitsanforderungen von BSCW untersucht, wie sie sich aus dem Einsatzzweck und der von BSCW benutzten Technologie ergeben. Ausgehend von den Sicherheitsanforderungen wird der Einsatz kryptographischer Verfahren motiviert.

1.1 Leistungen und Bedienung von BSCW

BSCW befindet sich in ständiger Weiterentwicklung: die derzeit[1] neueste Version 2.0 Beta 1 zeigt schon die ersten Änderungen an der Modellierung von Gruppen und Zugriffrechten in BSCW. Bei der Frage nach den Sicherheitsanforderungen von BSCW sollten diese Entwicklungsmöglichkeiten berücksichtigt werden.

Dieser Abschnitt bietet eine kurze Einführung in die Bedienung von BSCW Version 1.0 im Stil einer *Guided Tour* durch die wichtigsten Tätigkeiten, die in BSCW möglich sind. Die dargestellten Funktionen sind:

1. Das Speichern von Dokumenten.
2. Das Abrufen von Dokumenten.
3. Visuelles Ereignisprotokoll.
4. Versionsverwaltung.
5. Teilnehmerverwaltung.

[1] Stand 13.06.1996

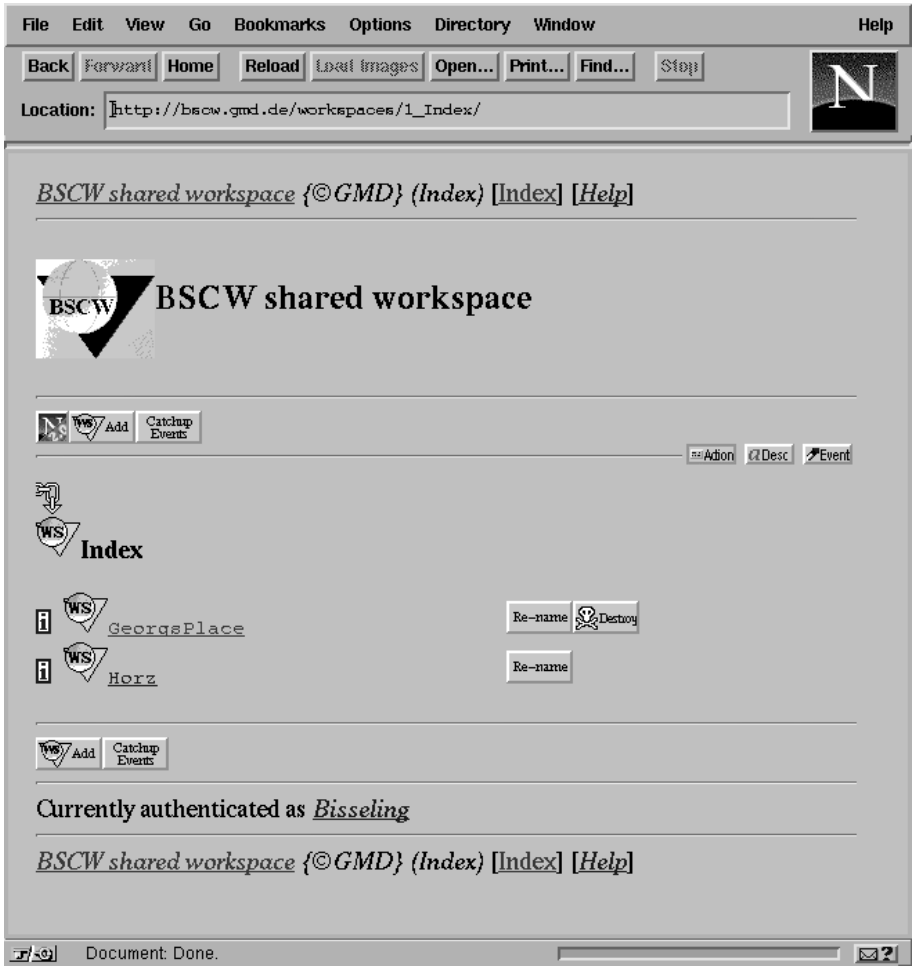

Abbildung 1.1: Übersicht der für den Benutzer Bisseling zugänglichen Arbeitsbereiche

Ein Benutzer, der in seinem Web-Browser die URL (Uniform Resource Locator, siehe [8]) `http://bscw.gmd.de/workspaces` wählt, findet sich (nachdem er sich über den üblichen Mechanismus authentisiert hat) auf der in Abbildung 1.1 gezeigten Seite wieder.

Durch Anklicken des Arbeitsbereiches `GeorgsPlace` gelangt man zur Übersicht über die in diesem Arbeitsbereich gespeicherten Dokumente, siehe Abbildung 1.2.

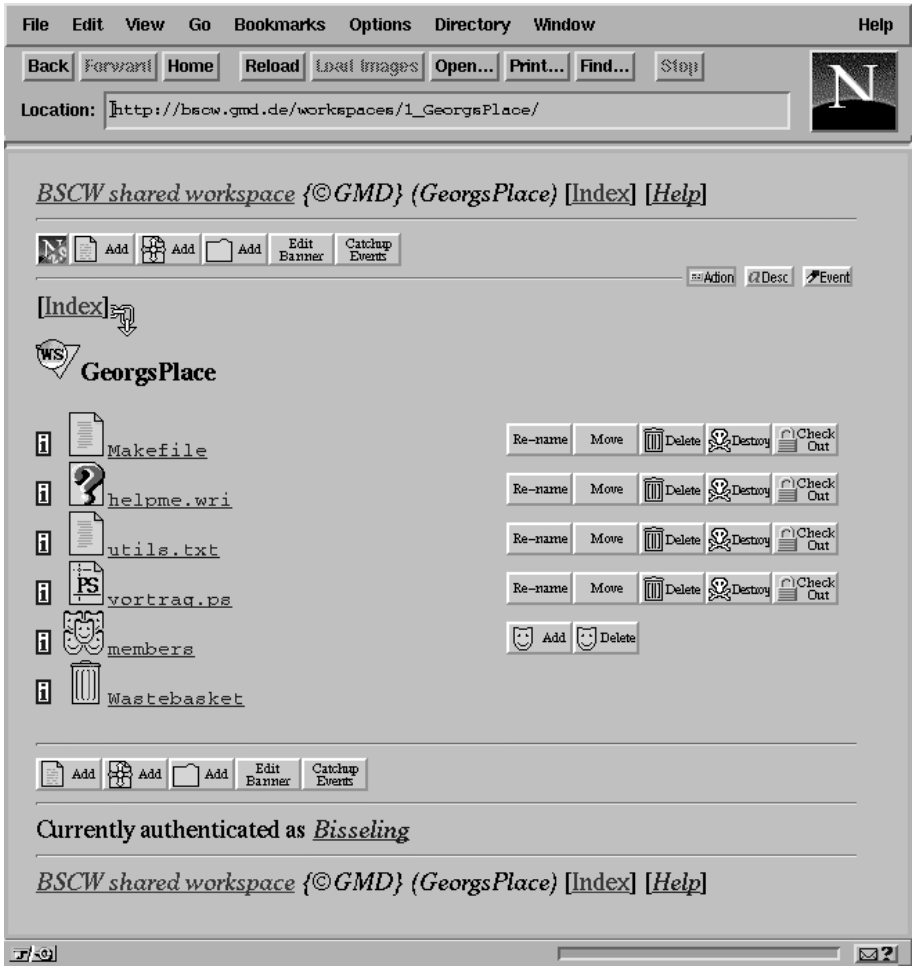

Abbildung 1.2: Der Arbeitsbereich `GeorgsPlace` in der Ansicht „Action View“.

Der Arbeitsbereich ist hier in der Ansicht „Action View“ dargestellt, die sich durch die Buttons neben den Dokumentnamen auszeichnet. Sie ermöglichen die üblichen Aktionen, wie man sie auch von Dateisystemen kennt. Die Unterscheidung zwischen Delete und Destroy bedeutet das Weiche Löschen durch Verschieben in den Papierkorb und das Harte Löschen eines Dokuments. Letzteres ist nur dem Besitzer des Dokuments möglich. Der Papierkorb ist global für den gesamten Arbeitsbereich, obwohl eine hierarchische Strukturierung des Arbeitsbereiches mit Hilfe von Verzeichnissen möglich ist. Bedauerlicherweise werden Informationen wie der Besitzer eines Dokuments sowie seine Größe und der Zeitpunkt seiner Erzeugung aus Platzgründen im „Action View“ nicht angezeigt. Auf den Button „Check out“ wird im Zusammenhang mit der Versionsverwaltung noch eingegangen.

Klickt man auf einen Dokumentnamen, so wird das Dokument mit dem dazugehörigen *MIME*-Typ[2] an den Web-Browser gesandt. Dessen Reaktion hängt davon ab, ob für diesen Dokumenttyp ein *external viewer* oder eine *helper application* vorgesehen ist. Für ein Postscript-Dokument wird z.B. im allgemeinen das Programm `ghostview` aufgerufen. Dokumente unbekannten Typs werden im lokalen Dateisystem gespeichert.

Klickt man den Button „Add Document“ 

an, so gelangt man zu der Seite aus Abbildung 1.3.

[2]Zu den Multipurpose Internet Mail Extensions siehe [9] und [10]. Zur Verwendung der *MIME*-Typen in HTTP siehe [2].

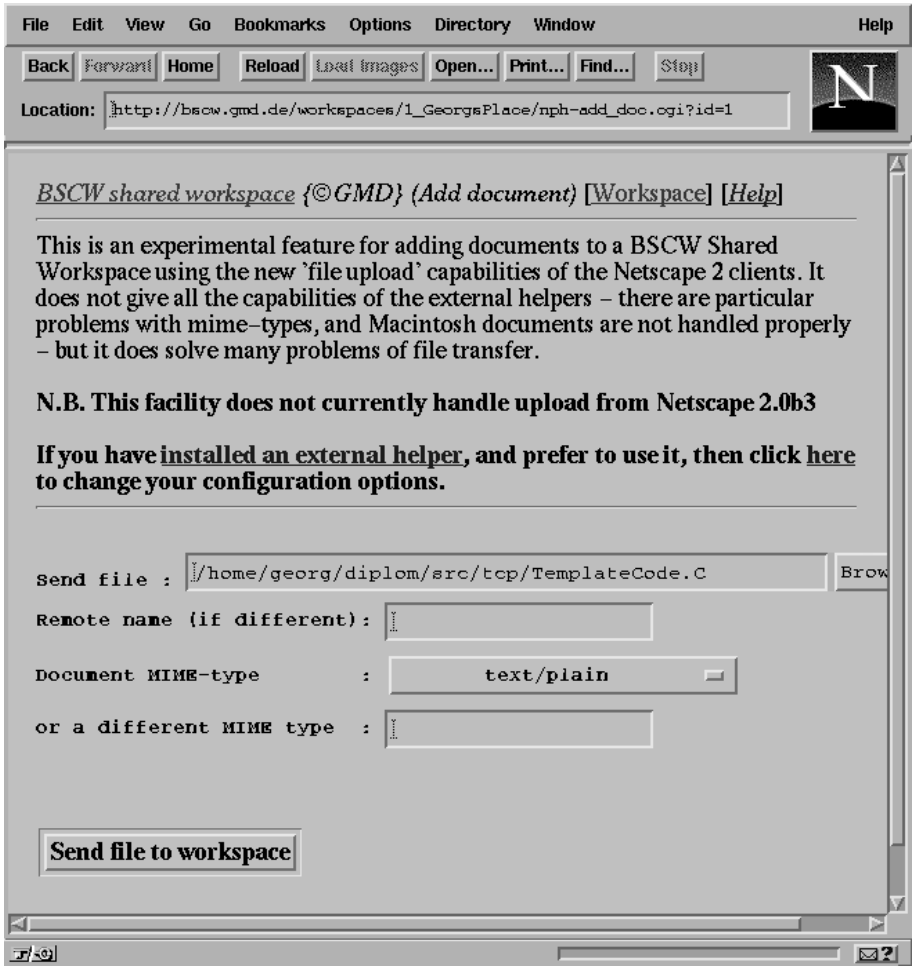

Abbildung 1.3: Formular für das „Upload“ eines Dokuments aus dem lokalen Dateisystem in den Arbeitsbereich.

Hier kann man über den Button „Browse“ rechts neben dem mit `Send file :` beschriebenen Textfeld ein Dokument aus dem lokalen Dateisystem auswählen und unter evtl. geändertem Namen zusammen mit der Information über den Dokumententyp an BSCW senden.

Das „Upload“ kann auch durch eine eigens dafür vorgesehene Hilfsanwendung geschehen. Diese ermöglicht auch das gleichzeitige Senden mehrerer Dateien. Die Möglichkeit, eine Dateiauswahl für das lokale Dateisystem in ein Formular einzubetten, ist bisher nur in Web-Browsern der Firma Netscape integriert.

Nach dem Upload des C++-Quelltextes `TemplateCode.C` (siehe Abbildung) stellt sich der Arbeitsbereich im „Event View“ wie in Abbildung 1.4 dar.

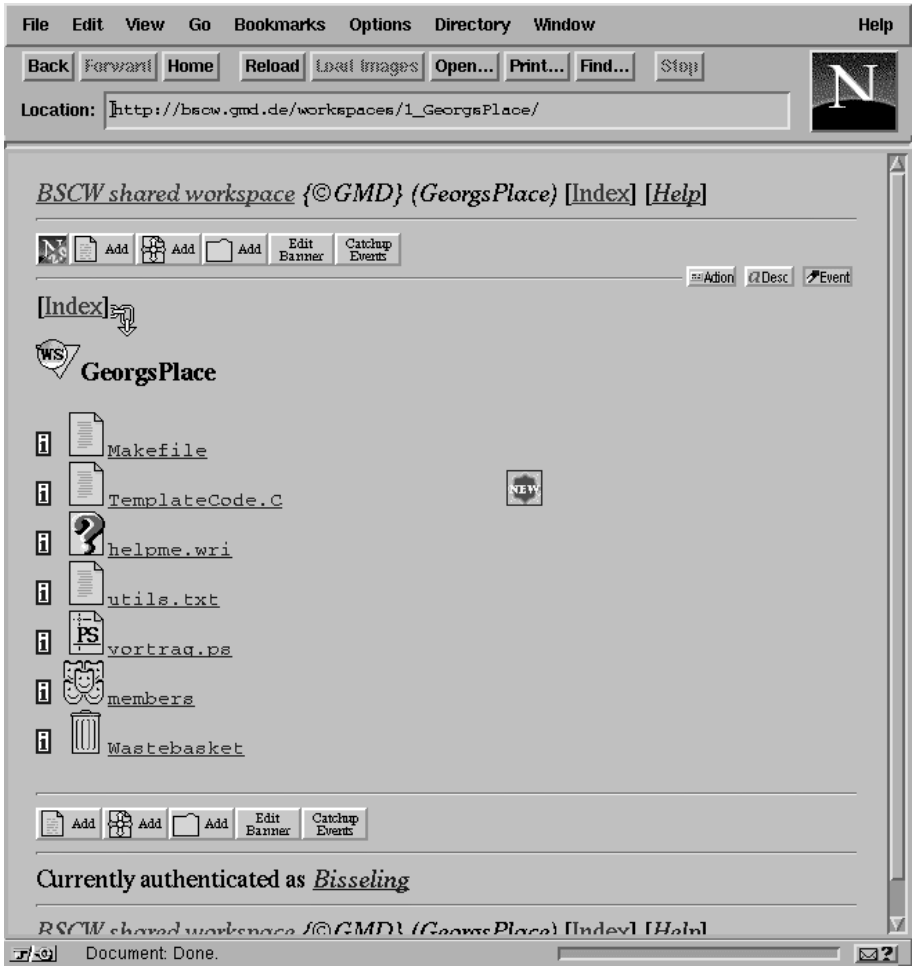

Abbildung 1.4: Ein neues Dokument wird durch das „New“-Icon angezeigt.

Im „Event View“ werden Ereignisse wie das Erzeugen, Überschreiben und Lesen eines Dokuments angezeigt. Drückt man den Button „Catchup Events“ so werden alle Events gelöscht, sodaß jeweils nur Änderungen sichtbar sind, die nach dem letzten „Catchup“ stattgefunden haben.

Lädt man noch ein Dokument des gleichen Namens in den Workspace, so wird dieses neue Dokument als aktualisierte Version des nun veralteten Dokuments betrachtet, wie in Abbildung 1.5 zu sehen ist.

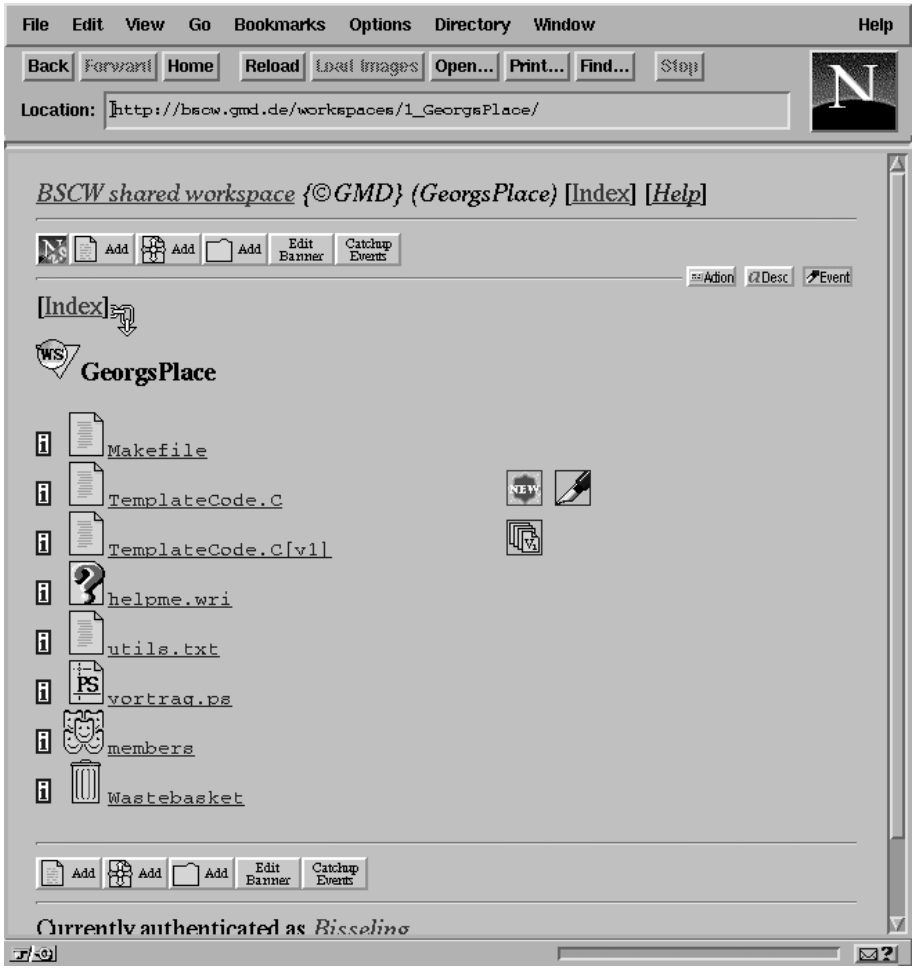

Abbildung 1.5: Eine neue Version von `TemplateCode.C` wurde im Arbeitsbereich gespeichert. Gezeigt ist die Darstellungsart „Event View“.

Arbeiten mehrere Benutzer in einem Arbeitsbereich zusammen, so kann es durch die Verwaltung beliebig vieler Versionen eines Dokuments nicht mehr zu sogenannten *race conditions* kommen. In Dateisystemen ohne Versionsverwaltung geschieht es häufig, daß zwei Benutzer gleichzeitig ein Dokument editieren. Es „gewinnt“ derjenige Benutzer das „Rennen“, der das Dokument als letzter speichert. Alle anderen Änderungen gehen verloren.

Mit einer Versionsverwaltung wie in BSCW würden die Änderungen nicht verloren gehen. Trotzdem wäre auch hier ein solches Szenario ärgerlich, da man die verschiedenen gleichzeitig gemachten Änderungen noch in einer gemeinsamen Version zusammenführen müßte.

Der in Abbildung 1.2 zu sehende Button „Check Out“ ermöglicht es ähnlich wie in RCS (Revision Control System, siehe [11] oder die Manual Page „rcsintro“ eines UNIX-Systems) ein Dokument auf das eigene lokale Dateisystem zu laden und für das Editieren durch andere zu sperren. In BSCW wird ein Benutzer nur gewarnt, wenn er ein „gesperrtes“ Dokument laden oder speichern möchte. Auch hier wird das Prinzip Kontrolle hintangestellt.

Klickt man auf das kleine Icon mit dem Buchstaben „i“ wie Information neben dem Namen `TemplateCode.C` so gelangt man zu den Detailinformationen zum jeweiligen Dokument, wie in den Abbildungen 1.6 und 1.7 zu sehen.

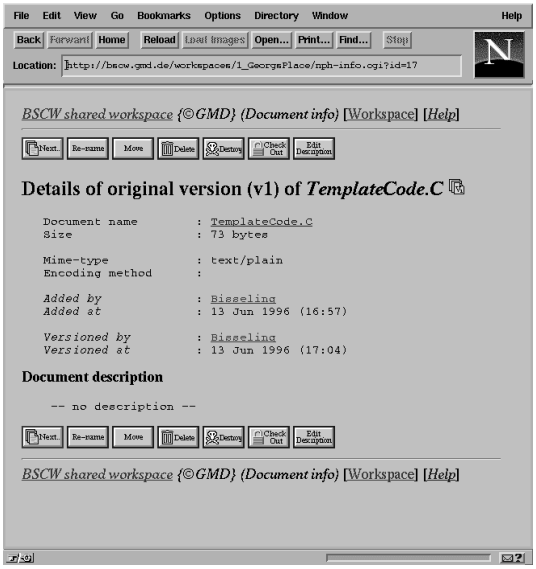

Abbildung 1.6: Detailinformationen zur ersten Version von `TemplateCode.C`.

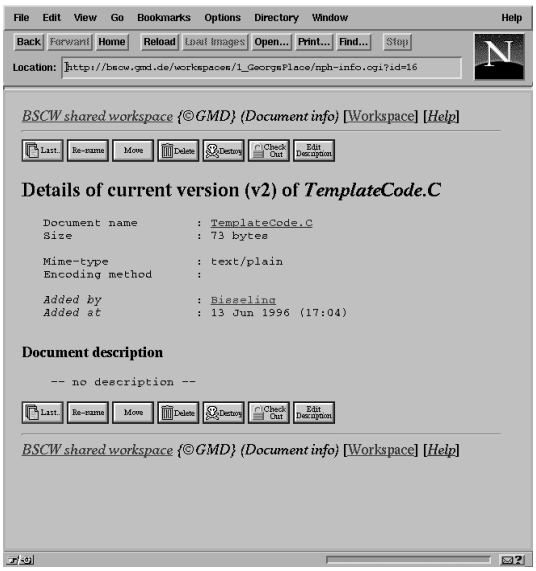

Abbildung 1.7: Und zur zweiten Version von `TemplateCode.C`.

Wir springen nun zurück zum „Action View“ auf den um zwei Dokumentversionen bereicherten Arbeitsbereich, siehe Abbildung 1.8.

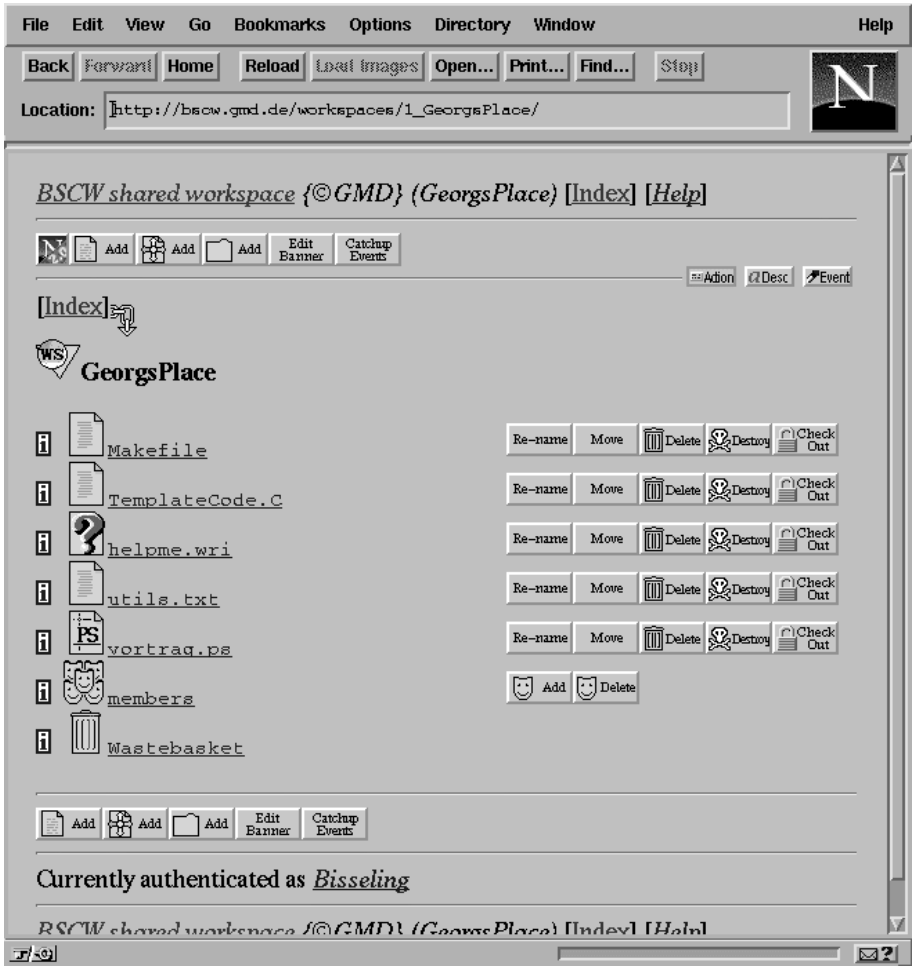

Abbildung 1.8: „Action View“ auf den Arbeitsbereich `GeorgsPlace` mit zwei Versionen des Dokuments `TemplateCode.C`.

Wie man sieht, sieht man nichts! Im Action View erhält man keinerlei Hinweis darüber, ob und wieviele vorangegangene Versionen eines Dokuments im Arbeitsbereich gespeichert sind. D.h., man muß explizit zur Darstellungsart „Event View“ wechseln, um einen Überblick über die Tätigkeit anderer Mitglieder des Arbeitsbereiches zu bekommen. Abgesehen davon muß man hierzu auch alle Verzeichnisse im Arbeitsbereich „durchwandern“.

Klickt man auf den Button „Member Add“ 

, so kann man mit einem in Abbildung 1.9 gezeigten Formular dem Arbeitsbereich neue Mitglieder hinzufügen.

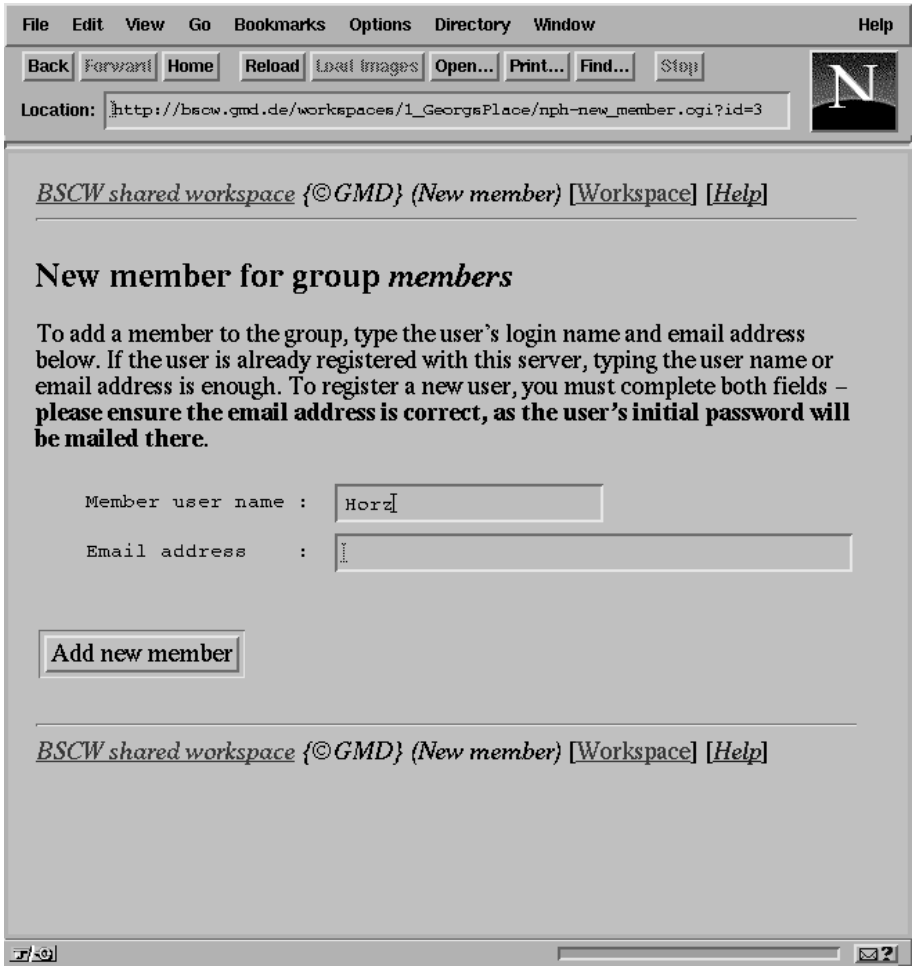

Abbildung 1.9: Der Arbeitsbereich `GeorgsPlace` bekommt ein neues Mitglied „Horz“.

Hier wird dem Arbeitsbereich GeorgsPlace der Benutzer Horz als neues Mitglied hinzugefügt. Hier sei angenommen, daß der Benutzer Horz dem BSCW-System schon bekannt ist[3].

In Abbildung 1.10 sieht man, welche Informationen BSCW für einen Benutzer speichert. Diese Informationen für den Benutzer Bisseling sind für alle anderen Benutzer sichtbar, die Mitglied eines gemeinsamen Arbeitsbereiches mit dem Benutzer Bisseling sind.

[3] Alle Benutzer aller Arbeitsbereiche des BSCW-Systems werden in einem Namensraum verwaltet. Wir nehmen also an, der Benutzer Horz ist schon Mitglied eines anderen Arbeitsbereiches, nämlich des Arbeitsbereiches `Horz`. Siehe auch Abbildung 1.1.

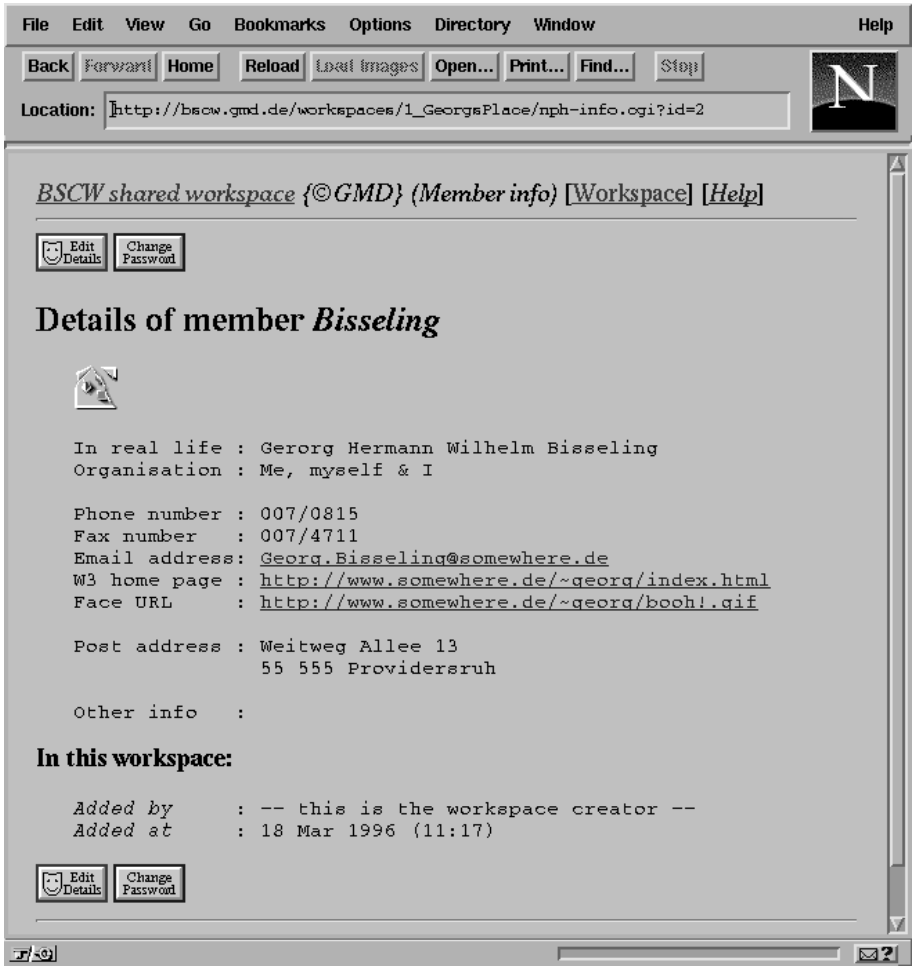

Abbildung 1.10: Detailinformattionen für den Benutzer Bisseling.

Dem interessierten Leser sei empfohlen, sich als Benutzer für BSCW eintragen zu lassen. Die erforderlichen Schritte sind unter der URL `http://bscw.gmd.de` beschrieben und die Prozedur ist je nach Geschwindigkeit der *email*-Verbindung in wenigen Sekunden bis Minuten durchlaufen. Unter der gleichen URL findet man auch die Vorgehensweise, Teilnehmer bei BSCW Version 2.0 Beta 1 zu werden.

1.2 Einordnung von BSCW im Groupware-Kontext

Um zu klären, was Groupware eigentlich ist und nicht ist, wird in [7] zunächst das Forschungsgebiet *computer support for cooperative work* (CSCW) umrissen, indem Begriffe wie Gruppe und Gruppenarbeit durch Definitionen präzisiert werden. Der Begriff Groupware ist annähernd so stark für die Vermarktung von mehr oder weniger neuen Ideen und Produkten abgenutzt worden wie die Begriffe „objektorientiert“ oder „Internet“. In [7, S. 21] wird der Begriff Groupware folgendermaßen definiert:

> Software, welche im Rahmen von CSCW entsteht, wird als *Groupware* bzw. *CSCW-Applikation* bezeichnet. Neben CSCW selbst ist Groupware der bekannteste Begriff im Gebiet der computerunterstützten Gruppenarbeit. Groupware ist vor allem im kommerziellen Bereich zu einem Schlagwort avanciert, das ähnlich wie der Begriff Multimedia einfach dazugehört, soll ein Produkt auf dem Markt Erfolg haben. Beispielsweise Büro-Informationssysteme, welche oft schon seit Jahren auf dem Markt sind, werden nachträglich als Groupware bezeichnet und können damit neu vermarktet werden, ohne im Sinne der CSCW-Forschung die Gruppenarbeit zu unterstützen.

Für die Beschreibung von CSCW-Applikationen werden Kriterien angeboten, die eine Anwendung zwar nicht erschöpfend, so doch vorläufig charakterisieren können. So stellt man z.B. die Frage, ob Kommunikation in einer Groupware-Applikation eher implizit oder explizit stattfindet. Wendet man diese Kriterien auf BSCW an, so erhält man folgende Beschreibung des Systems: in BSCW werden *beliebige Datentypen* über *räumliche Entfernungen asynchron mehreren* Kommunikationspartnern in einem gemeinsamen Arbeitsbereich zur Verfügung gestellt; also findet *(implizite)* Kommunikation statt.

Mit diesen einfachen Kriterien läßt sich die Vielfalt an verschiedenen Anwendungen, die als Groupware-Applikation angesehen werden, nicht zufriedenstellend ordnen. Die Autoren versuchen, CSCW-Applikationen in vier sogenannte Systemklassen zu kategorisieren:

Systemklasse Kommunikation In diese Klasse fallen elektronische Post-Systeme sowie Systeme für Audio-, Video- und Desktopkonferenzen.

Systemklasse Gemeinsame Informationsräume In diese Klasse fallen Bulletin-Board-Systeme, spezielle Datenbanken und verteilte Hypertext-Systeme.

Systemklasse Workflow Management In diese Klasse fallen Applikationen für die Analyse, Modellierung, Simulation, Ausführung und Steuerung von Workflows benutzt werden.

Systemklasse Workgroup Computing In diese Klasse fallen Planungssysteme, Entscheidungs- und Sitzungsunterstützungssysteme sowie Gruppeneditoren.

Bevor BSCW in eine dieser Klassen eingeordnet werden kann, müssen einige der verwendeten Begriffe zumindest grob erläutert werden: Workflow, „spezielle“ Datenbank und Entscheidungsunterstützungssystem.

1.2.1 Der Begriff Workflow

Leider wird der Begriff Workflow in [7] nicht definiert, aber es findet sich in [12, Abschnitt 1.2] eine Definition des Begriffs Workflow-Management:

> Workflow management software is a proactive computer system which manages the flow of work among participants, according to a defined procedure consisting of a number of tasks. It co-ordinates user and system participants, together with the appropriate data resources, which may be accessible directly by the system or

> off-line, to achieve defined objectives by set deadlines. The co-ordination involves passing tasks from participant to participant in correct sequence, ensuring that all fulfil their required contributions, taking default actions when necessary.

Auf den Punkt gebracht, ist der Workflow für den Arbeitsablauf in einer Organisation das, was eine Prozedur für ein strukturiertes Programm ist. Dieses Bild trägt auch dann weiter, wenn man an verschachtelte Aufrufe, Iteration und Rekursion denkt.

1.2.2 Der Begriff „Spezielle“ Datenbank

Zur Charakterisierung von „speziellen“ oder Non-Standard-Datenbanken werden folgende Eigenschaften genannt [7, S. 174]:

1. Die Beschreibung der zu modellierenden Informationsobjekte ist nicht mehr einfach durch eine Reihe von Attributen möglich, da diese aus zahlreichen verschiedenartigen Bausteinen zusammengestetzt sind. Nur die elementaren Bausteine können durch Attribute beschrieben werden. Diese komplexen Objekte müssen in herkömmlichen Datenbanksystemen durch zahlreiche Sätze unterschiedlichen Typs abgebildet werden. Zur Laufzeit müssen durch aufwendige Verbundoperationen die komplexen Objekte rekonstruiert werden.

2. Die Konsistenz der Abbildung von komplexen Informationsobjekten kann nur durch umfangreiche Integritätsbedingungen definiert werden, deren Einhaltung das Datenbanksystem überwachen muß. Dazu sind die Ausdrucksmittel in herkömmlichen Datenbanksystemen zu schwach.

3. Transaktionen, d.h. Folgen von Änderungen, welche den Datenbestand in einen neuen, konsistenten Zustand überführen sollen, betreffen nicht mehr nur einige wenige Sätze der Datenbank. Speziell in gemeinsamen Informationsräumen kann es Tage oder Wochen dauern, bis ein Informationsobjekt sich in einem Zustand befindet, in dem es anderen Benutzern zugänglich gemacht werden kann.

4. Die Verwaltung und Integration neuer Daten- bzw. Objekttypen mit speziellen Operatoren ist möglich, z.B. eines „Punktes im Dreidimensionalen Raum“.

5. Die Verwaltung von Versionen wird unterstützt. Speziell bei der kooperativen Erstellung von Informationsobjekten ist es erforderlich, neben der aktuellen Version auch Zugriff auf ältere Versionen zu besitzen.

Um sich eine Vostellung von den Systemen zu machen, die in die Klasse Gemeinsame Informationsräume fallen, genügt es in diesem Zusammenhang, sich Datenbanken vorzustellen, die *Objekte* verwalten, die nicht mehr durch einfache Aufzählung von Attribut-Wert-Paaren beschrieben werden können.

1.2.3 Der Begriff Entscheidungsunterstützungssystem

Zuletzt sei noch versucht, den Begriff Entscheidungsunterstützungssystem zu klären. In [7, S. 227] werden *Entscheidungsunterstützungssysteme für Gruppen* wiefolgt definiert:

> Entscheidungsunterstützungssysteme sind Systeme, welche die Effektivität von Entscheidungsprozessen von Gruppen im Rahmen teilweise strukturierbarer Aufgaben unterstützen. Ein Entscheidungsunterstützungssystem für Gruppen muß zwingend von mehreren Personen benutzbar sein bzw. benutzt werden.

Man darf sich also Systeme vorstellen, die menschliche Interaktion wie Argumentation, Erörterung, Diskurs, Rede und Gegenrede und mehrheitliche Abstimmung auf geeignete Weise moderieren helfen.

1.2.4 Einordnung von BSCW in die Systemklassen

Mit diesen Begriffsklärungen sollte eine grobe Vorstellung von der oben angegebenen Klasseneinteilung möglich sein. Nun läßt sich die Frage stellen, in welche Klasse das BSCW-System fällt.

Drei der vier Systemklassen, nämlich Kommunikation, Workflow Management und Workgroup Computing, können schnell ausgeschlossen werden, da BSCW weder die direkte Kommunikation mit einem ausgewählten Adressaten unterstützt noch die Strukturierung von zeitliche Abläufen oder Entscheidungsprozessen.

In der Klasse Gemeinsame Informationsräume fällt BSCW unter die Rubrik Bulletin-Board-Systeme, wo es sich direkt neben *Lotus Notes* einreiht. Die Ähnlichkeiten zwischen diesen beiden Systemen sind recht groß, wobei die Möglichkeiten von *Lotus Notes* über die Funktionalität des BSCW-Systems hinausgehen.

In [7, S. 158] wird sogar eingeräumt, daß man *Lotus Notes* bei Ausnutzung all seiner Möglichkeiten (Zusatzwerkzeuge und Makroprogrammiersprache) auch als Entwicklungsumgebung für Workflow Management-Systeme bezeichnen darf.

Damit wäre die Frage, ob BSCW zur Groupware „im Sinne der CSCW-Forschung“ gehört und in welche Systemklasse BSCW fällt, positiv beantwortet. Zudem ist klar, daß BSCW im Vergleich mit anderen in [7] oder [12] beschriebenen Systemen eher einfach, klein und wenig mächtig ist.

Diese Tatsache muß nicht unbedingt ein Nachteil sein, da BSCW durch seine Einfachheit schnell zu durchschauen ist. Sein Einsatz erfordert es nicht, Arbeitsabläufe, Gruppenzugehörigkeiten, Kompetenzen etc. formal zu spezifizieren bzw. einem bestimmten Paradigma anzupassen. Natürlich wird dadurch auch die Chance vertan, durch die Analyse von Arbeitsabläufen organisatorische Schwächen aufzudecken.

Man sieht es dem BSCW-System an, daß es als Forschungsobjekt und -vehikel von Wissenschaftlern entwickelt wurde, die dem Ideal freier, selbstbestimmter Arbeit anhängen.

1.3 Die Sicherheitsanforderungen von BSCW

Der wesentliche Sicherheitsmangel von BSCW ist die ungeschützte Übertragung aller Daten über TCP/IP-Verbindungen. Dabei lassen sich zwei Klassen von Daten unterscheiden. Es sind dies erstens die HTML-Seiten, die quasi die graphische Benutzeroberfläche des BSCW-Systems ausmachen, und zweitens die verwalteten Dokumente.

Die HTML-Seiten enthalten im wesentlichen sogenannte Verkehrsinformationen, die es erlauben festzustellen, wer wann mit wem in welchem Zusammenhang kommuniziert hat. All dies läßt sich allein aus Benutzernamen, Dokumentnamen und den Namen von Arbeitsbereichen ersehen, die z.B. Angeboten, Aufträgen, Projekten oder Abteilungen entsprechen können. Mit etwas Geschick läßt sich über die statistische Auswertung solcher Daten sehr viel Information über die Struktur einer Organisation gewinnen.

So gewonnene Informationen können genutzt werden, um Vertrautheit mit Personen oder der Organisation allgemein vorzutäuschen und sich so weitere Möglichkeiten zu erschleichen (*social hacking*, [13, S. 30]). Aus diesem Grund ist es wünschenswert, nicht nur die übertragenen Dokumente, sondern auch sämtliche andere Daten vor der unbefugten Einsichtnahme zu schützen.

Dieser Aspekt wird hier besonders betont, da die Struktur der Implementierung von BSCW einen bestimmten Entwurf für einen Sicherheitsmechanismus geradezu aufdrängt, der die Absicherung dieser Verkehrsinformationen nicht oder nur schwerlich leisten kann (siehe Abschnitt 3.1).

Die Forderung nach der Absicherung der Datenübertragung bildet den Ausgangspunkt für die Überlegungen zur Absicherung des BSCW-Systems. Die Sicherheit einer Datenübertragung erfordert drei Eigenschaften: die Abhörsicherheit, die Authentizität und die Integrität.

Abhörsicherheit bedeutet, daß ein passiver Angriff auf den Übertragungsweg unmöglich ist. Sie läßt sich durch die Verwendung von Verschlüsselungstechniken erreichen, aber auch durch Absicherung des Übertragungsmediums, man denke nur an Erdkabel. Das Internet bietet ideale Voraussetzungen für einen passiven Angriff, da die übermittelten Pakete unter Umständen über eine große Zahl nicht kontrollierbarer Rechner weitergegeben werden. Eine Absicherung der Verbindungswege scheint deshalb kaum möglich.

Die Authentizität einer Nachricht gewährleistet, daß der behauptete und der tatsächliche Absender einer Nachricht übereinstimmen. Die Internet Protokolle übermitteln nur die IP-Adresse des absendenden Rechners und diese läßt sich leicht fälschen, da sie sich an jedem Rechner frei konfigurieren läßt.

Authentizität elektronischer Nachrichten kann durch die Verwendung von Verschlüsselung oder sogenannter Digitaler Unterschriften erreicht werden. Dabei ist immer die Frage zu beantworten, wessen Identität authentisiert werden soll: die eines Rechners (oder einer Programmlizenz etc.) oder die einer Person.

Die Integrität einer Nachricht ist die Eigenschaft, auf dem Weg vom Sender zum Empfänger nicht durch Dritte verändert worden zu sein. Es geht also nicht um Veränderungen der Nachricht durch Störungen des Übertragungskanals: aus der Perspektive eines kryptographischen Protokolls sind alle Störungen als Angriffe Dritter zu werten. Die Unterscheidung zwischen Authentizität und Integrität ist zwar üblich, erscheint mir aber künstlich, da eine manipulierte Nachricht gleichzeitig auch nicht authentisch ist und umgekehrt.

Für das BSCW-System ist Abhörsicherheit nur durch die Verwendung von Verschlüsselung zu erreichen, da die Übertragungswege nicht abgesichert werden können.

BSCW regelt die Zugriffsrechte für verschiedene Workspaces auf Basis der Identität des jeweiligen Benutzers. BSCW macht keine Annahmen über die Rechner, mit denen das System kommuniziert. Das bedeutet, ein Anwender sollte den Sicherheitsmechanismus zum Zugriff auf BSCW auf beliebig vielen Rechnern installieren können, ohne diese Installationen zentral anmelden zu müssen. Es kommt also nur eine Authentisierung von Personen, nicht von Rechnern oder IP-Adressen infrage.

Die Authentizität einer Nachricht sollte aber nicht nur für eine Kommunikationsrichtung

sichergestellt sein. Ein Benutzer von BSCW muß sicherstellen können, mit dem richtigen HTTP-Server verbunden zu sein. Auch hier sollte es keine Rolle spielen, auf welchem Rechner bzw. auf welcher Kombination aus Rechnername und IP-Adresse der HTTP-Server läuft. Nur so ist eine reibungslose Administration von BSCW möglich, ohne z.B. bei Änderung der IP-Adresse alle Benutzer des Systems (auf sicherem Wege!) benachrichtigen zu müssen.

Die Maximalanforderungen an einen Sicherheitsmechanismus für BSCW lauten also:

Authentizität und Integrität Sowohl der Benutzer als auch der BSCW-Server müssen ihre Identität der Gegenseite beweisen. Die Identitäten der Benutzer sollten persistent sein unter Wechsel des Aufenthaltsortes oder Arbeitsplatzrechners. Ebenso sollte die Identität des Server unabhänging von der Installation auf einem bestimmten Rechner oder auf einer bestimmten Programmlizenz sein.

Abhörsicherheit Die *gesamte* Kommunikation zwischen dem Client-Rechner und dem BSCW-Server muß gegen das Abhören geschützt sein.

Komfort Die Verwendung des Sicherheitsmechanismus sollte so komfortabel sein, daß beim Benutzer nicht der Wunsch entsteht, ihn zu unterlaufen.

Offenheit Die Funktionsweise des Sicherheitsmechanismus soll vollkommen offengelegt werden (können), ohne seine Wirksamkeit zu beeinträchtigen.

Zu Beginn der Arbeit bestanden Zweifel daran, ob es möglich ist, die Gesamtheit aller übertragenen Daten im BSCW-System abzusichern. Zweifel bestanden insbesondere daran, ob die entwickelte Lösung so komfortabel sein würde, daß die Benutzung von BSCW zusammen mit dem Sicherheitsmechanismus mindestens zumutbar bleibt.

Die Sicherheitsanforderungen von BSCW sind nach heutigem Erkenntnisstand nur durch kryptographische Verfahren zu erfüllen. So schreiben Chapman und Zwicky in [14, S. 285]:

> Better yet, don't do any authentication or authorization based solely on hostname or even on IP address; there is no way to be sure that a packet comes from the IP address it claimes to come from, unless there is some kind of cryptographic authentication within the packet that only the true source could have generated.

Das folgende Kapitel gibt einen Überblick über gängige kryptographische Verfahren, ihre Arbeitsweise, ihre Leistungen und ihre Defizite beim Einsatz in Protokollen für elektronische Datenübertragung.

Kapitel 2

Kryptographische Verfahren

Der Wissenschaftszweig der Kryptologie teilt sich auf in die Kryptographie, die Lehre des Entwurfs von Chiffrierverfahren, und die Kryptoanalyse, die das „Brechen“ von Chiffren zum Ziel hat. Erfahrung auf dem Gebiet der Kryptoanalyse wird als Voraussetzung für erfolgreiches Arbeiten eines Kryptologen auf dem Gebiet der Kryptographie gesehen [15, S. 170]. Trotzdem besteht für den Kryptologen allzu leicht die Gefahr, die eigenen Entwicklungen zu unkritisch zu betrachten ([15, S. 21 u. 23].

Die Kryptologie wurde erst lange nach dem zweiten Weltkrieg Anfang der siebziger Jahre zu einem öffentlich zugänglichen Forschungsgebiet. Bis dahin unterlagen die meisten Forschungsergebnisse militärischer Geheimhaltung. Das amerikanische *National Bureau of Standards* (NBS) forderte 1973 öffentlich zur Vorlage von Vorschlägen für ein Verfahren zur Verschlüsselung der Kommunikation zwischen Behörden, Banken und anderen privaten Institutionen auf. Schneier [16, S. 265] schreibt dazu:

> In the early 1970s, nonmilitary cryptographic research was haphazard. Almost no research papers were published in the field. Most people knew that the military used special encoding equipment to communicate, but few understood the science of cryptography. The National Security Agency (NSA) had considerable knowledge, but they did not even publicly admit their own existence[1].
>
> Buyers didn't know what they were buying. Several small companies made and sold cryptographic equipment, primarily to overseas governments. The equipment was all different and couldn't interoperate. No one really knew if any of it was secure; there was no independent body to certify the security.

Ergebnis dieses Aufrufs des NBS ist der im nachfolgenden Abschnitt dargestellte *Data Encryption Standard* (DES), der auch heute noch meistverwendete symmetrische Verschlüsselungsalgorithmus. Die Darstellung des DES nimmt im folgenden nicht nur wegen seiner großen kommerziellen Bedeutung breiten Raum ein, sondern auch, weil die Beurteilung und Analyse des DES der Kryptologie entscheidende Impulse gegeben hat [17, S. 218]. Zudem wird am Beispiel des DES deutlich, wie weit der militärische und der zivile Kenntnisstand auf dem Gebiet der Kryptologie in den siebziger Jahren auseinanderklaffte und dies mit hoher Wahrscheinlichkeit heute noch tut.

Bauer [15] und Schneier [16] grenzen in ihren einführenden Kapiteln die Kryptographie von der Steganographie ab. Beide Begriffe bedeuten in der wörtlichen Übersetzung verdecktes oder verborgenes Schreiben. Im Unterschied zur Kryptographie, die das Vorhandensein einer chiffrierten Nachricht nicht verschleiert, entwickelt die Steganographie Methoden, mit denen geheimzuhaltende Nachrichten in unverfänglichen Fülldaten (Rauschen) versteckt werden können, sodaß abgefangene Nachrichten (oder Gegenstände) keinen Verdacht erregen. Bilderrätsel, Mikrographie, Geheimtinten und hohle Absätze fallen also in das Ressort

[1] Was ihr auch die Namen „No Such Agency“ oder „Never Say Anything“ eintrug.

der Steganographie. In [15, Kapitel 1] finden sich weitere Beispiele für steganographische Methoden.

Selbstverständlich ist die „Verschachtelung“ kryptographischer und steganographischer Methoden möglich und erschwert das unbefugte Mitlesen der Nachrichten sehr. Verständlicherweise können Methoden der Steganographie nicht in öffentlich bekanntzumachenden Protokollen verwendet werden, da ihre öffentliche Darstellung sie wertlos machen würde. Aus diesem Grund wird im folgenden nicht mehr auf steganographische Methoden eingegangen.

Verschiedene klassische kryptographische Verfahren werden in [15] und [17] in einer fein gegliederten Taxonomie geordnet, auf die hier verzichtet werden kann. Im folgenden wird hauptsächlich die Unterscheidung zwischen den symmetrischen oder *Private Key*-Verfahren und den asymmetrischen oder *Public Key*-Verfahren eine Rolle spielen. Im Zusammenhang mit den verschiedenen Betriebsarten für die DES-Chiffre wird noch kurz auf die Unterscheidung zwischen Block- und Stromchiffren einzugehen sein.

2.1 Aufgaben kryptographischer Verfahren

Kryptographische Verfahren dienen dem Erreichen folgender Ziele:

1. Vertraulichkeit (*Secrecy*)
2. Integrität (*Integrity*)
3. Authentizität (*Authenticity*)
4. Nichtabstreitbarkeit (*Nonrepudiability*)

Das klassische Ziel der Kryptographie, die *Vertraulichkeit*, ist erreicht, wenn nur die vom Sender intendierten Empfänger eine verschlüsselte Nachricht entschlüsseln können. Hier gibt es zwei interessante Extremfälle. Zum einen können Sender und Empfänger identisch sein, z.B. bei der verschlüsselten Speicherung von Daten. Zum anderen ist es möglich, daß der Sender selbst nicht in der Lage ist, die Nachricht zu entschlüsseln[2].

Integrität ist erreicht, wenn eine Veränderung der Nachricht auf dem Übertragungsweg sicher (oder mit hoher Wahrscheinlichkeit) erkannt wird. Dabei muß man unterscheiden zwischen Veränderungen durch Störungen der Übertragung und Manipulation durch Dritte. Erstere sollten von herkömmlichen Protokollen entdeckt werden. Alle Veränderungen, die von einem kryptographischen Verfahren aufgedeckt werden, sind als Angriff zu bewerten.

Authentizität ist erreicht, wenn sich Sender und Empfänger gegenseitig ihre Identität beweisen können (und müssen).

Nichtabstreitbarkeit ist erreicht, wenn der Sender das Erstellen und Versenden einer Nachricht nicht verleugnen kann, wenn sie einmal versandt worden ist. Dieser Punkt wird im Zusammenhang mit Vertragsabschlüssen und finanziellen Transaktionen über digitale Kanäle immer bedeutender.

Von allen vier Zielen ist die Authentizität das wichtigste und Voraussetzung für das Erreichen der anderen Ziele. Vertraulichkeit bedingt das Wissen, mit wem eine Nachricht, ein Geheimnis geteilt wird. Integrität setzt voraus, daß die empfangene Nachricht in genau dieser Form vom vermuteten Sender erzeugt wurde, und die Nichtabstreitbarkeit setzt die Möglichkeit voraus, aus einer erhaltenen Nachricht *beweisbar* auf die Identität des Senders schließen zu können.

So unterschiedlich wie die bei der Anwendung kryptographischer Methoden verfolgten Ziele sind die Angriffsarten der „feindlichen“ Kryptoanalytiker, im folgenden kurz Angreifer genannt. Die Angriffsarten unterscheiden sich stark für symmetrische und asymmetrische Verfahren und werden aus diesem Grund erst später nach der Einführung in die jeweilige Verfahrensklasse dargestellt.

[2] Ein Fall, der erst bei der Verwendung asymmetrischer Verfahren auftreten kann.

2.2 Grenzen kryptographischer Verfahren

Wichtiger als die verschiedenen Angriffsarten und das Abschätzen der Leistungsfähigkeit verschiedener Algorithmenklassen und Protokolle ist das Bewußtwerden über die Ziele, die durch die Verwendung kryptographischer Verfahren nicht erreicht werden können[3]:

- Kryptographie schützt nicht die unverschlüsselten, lokal gespeicherten Kopien der verschlüsselten Daten. Insbesondere das nicht wirklich destruktive Löschen von Dateien wird unter vielen Betriebssystemen zum Problem: die momentan unbenutzten Blöcke eines Dateisystems enthalten noch die vollständigen Daten ihrer letzten Benutzung.
- Kryptographie schützt nicht mehr, wenn die Schlüssel „gestohlen“ werden. Da die Schlüsselinformation nicht physikalisch gestohlen werden muß, kann dieser Fall sehr lange unbemerkt bleiben. Dieser Punkt spricht für das (verschlüsselte) Abspeichern von Schlüsseln auf Chip- oder Magnetkarten, deren Verlust oder Manipulation eher bemerkt wird.
- Kryptographie schützt nicht vor *denial of service attacks*, dem Überfluten von Rechnern oder Teilnetzen mit bedeutungslosen Nachrichten ([14, S. 7], [19, S. 165]).
- Kryptographie schützt nicht bei manipulierter Kryptosoftware. Auch Kryptohardware ist kein grundsätzlicher Schutz, da jede Hardware nur mit Software betrieben werden kann.
- Kryptographie schützt nicht vor „Verrat“ im eigenen Hause, gleichgültig ob Bestechung, Erpressung oder nur Fahrlässigkeit im Spiel sind.
- Kryptographie verschleiert nicht die Tatsache der Kommunikation zwischen zwei Parteien. Dieser Punkt ist insbesondere im Zusammenhang mit kryptographischen Protokollen wichtig: welche sogenannten Verkehrsinformationen kann ein Angreifer aus dem sichtbaren Datenstrom ablesen?

Erst wenn genau klar ist, welche Ziele erreicht werden können und sollen und, welche protokollarischen Änderungen im Arbeitsablauf nötig sind, kann der Einsatz von Kryptographie optimale Ergebnisse erbringen.

[3] Nach [18, S. 54]

2.3 Symmetrische Verfahren

Ein symmetrisches Verfahren ermöglicht zwei Kommunikationspartnern, die traditionellerweise **A**lice und **B**ob heißen, die sichere Kommunikation über einen unsicheren Kanal. Der Name „symmetrisch“ bedeutet hier, daß beide Partner das gleiche Verfahren und vorallem den *gleichen* Schlüssel verwenden.

Das Alphabet für den Klartext und den Chiffretext ist im allgemeinen durch den Übertragungskanal oder das zugrundeliegende Protokoll vorgegeben. Aus diesem Grund reicht es, Verfahren zu betrachten, die für Klartext und Chiffretext dasselbe Alphabet Σ verwenden. Eine symmetrische Chiffre ist also im wesentlichen eine Abbildung F mit:

$$\begin{array}{rccc} F: & \mathcal{K} \times \Sigma & \rightarrow & \Sigma \\ & (k, v) & \rightarrow & F_k(v) \end{array}$$

Dabei ist $k \in \mathcal{K}$ ein *Schlüssel* aus dem *Schlüsselraum* $\mathcal{K}$, $v \in \Sigma$ ein Alphabetzeichen aus dem Klartext und $F_k(v) \in \Sigma$ das korrespondierende Alphabetzeichen des Chiffretextes.

Nehmen wir nun an, Alice und Bob kennen einen gemeinsamen Schlüssel k und Alice verschlüsselt v zu $F_k(v)$. Damit Bob die Nachricht entschlüsseln kann muß F bezüglich des Alphabets Σ umkehrbar sein und die Umkehrung F_k^{-1} muß leicht berechenbar sein:

$$\bigwedge_{v \in \Sigma} \bigwedge_{k \in \mathcal{K}} F_k^{-1}(F_k(v)) = v$$

Bei der Beurteilung moderner Kryptosysteme geht man von der sogenannten Kerckhoff'schen Maxime [15, S. 149] aus, die lautet: „Der Feind kennt das benutzte System“. Ein Angreifer, der den Klartext aus dem Chiffretext gewinnen will, sollte also keinen Vorteil davon haben, die Abbildung F und die Vorgehensweise zu ihrer Invertierung zu kennen, wenn er nicht auch den gewählten Schlüsel k kennt.

Die Kerckhoff'sche Maxime ist von der Verwendung kryptographischer Verfahren in Kriegszeiten geprägt, in denen die Geheimhaltung eines Verfahrens oft fehlschlägt. Heute sollen Kryptoverfahren in öffentlich bekannten Protokollen eingesetzt werden, sodaß Geheimhaltung prinzipiell unmöglich ist.

Die „effiziente Berechenbarkeit“ von F und F^{-1} bedeutet in diesem Zusammenhang nicht abstrakt „mit polynomiellen Aufwand“, sondern Mindestdurchsatz in Kilobit pro Sekunde bei vertretbaren Beschaffungskosten für Hard- und Software in DM.

Ebenso bedeutet „der Angreifer kann den Chiffretext *nicht* entschlüsseln“ hier „*nicht mit für den Angreifer vertretbarem Aufwand an Zeit und Geld*“. Damit hängt die Sicherheit von den Möglichkeiten und der Motivation des Angreifers ab, sowie vom (vermutlichen) Wert der Nachricht.

Damit fließen in die Beurteilung der Sicherheit einer Chiffre sowohl komplexitätstheoretische Überlegungen ein (Berechnung unterer Schranken für Algorithmen zum „Brechen“ einer Chiffre) als auch Abschätzungen der Entwicklung des Preis-Leistungsverhältnisses von Rechenanlagen oder des Fortschritts in der Kryptoanalyse ([17, Abschnitt 2.2], [16, S. 167] und im Zusammenhang mit RSA [18, S. 361]).

2.3.1 Leistungsfähigkeit symmetrischer Chiffren

Wie erreichen symmetrische Algorithmen prinzipiell die vier gesteckten Ziele für allgemeine kryptographische Verfahren?

Vertraulichkeit ist gewährleistet solange es Alice und Bob gelingt, ihren Schlüssel geheimzuhalten. Notwendigerweise müssen sich Alice und Bob in Bezug auf die Weitergabe des Schlüssels gegenseitig ihrer Diskretion sicher sein, sonst würden sie sich keine vertraulichen Nachrichten senden.

Integrität wird wie üblich durch Prüfsummen erreicht, die mit der Nachricht verschlüsselt werden und so vor Manipulation sicher sind.

Authentizität ist gegeben solange der Schlüssel geheim bleibt.

Die Nichtabstreitbarkeit ist mit symmetrischen Verfahren prinzipbedingt nicht zu erreichen, da Alice und Bob exakt das gleiche Verfahren und den gleichen Schlüssel zur Erzeugung von Chiffretext benutzen. Dadurch ist mit dem Chiffretext keine Entscheidung möglich, ob Alice oder Bob der Absender ist[4].

2.3.2 Angriffe auf symmetrische Chiffren

2.3.2.1 Szenarien

Der klassische Angriff auf ein Verschlüsselungsverfahren ist der **Chiffretextangriff** (*ciphertext only attack*), bei dem der Kryptoanalyse ausschließlich der Chiffretext zur Verfügung steht. Ziel des Angriffes ist zumindest das Entschlüsseln einer Nachricht im idealen Fall das Berechnen des benutzten Schlüssels.

Alle nachfolgend beschriebenen Verfahren sind nach bisherigem Kenntnisstand sicher gegenüber diesem Angriff: es ist kein Verfahren zum Entschlüsseln der Nachricht bekannt, das weniger Aufwand erfordert als die Erschöpfende Suche (*exhaustive search* oder *brute force attack*) im Schlüsselraum. Die Sicherheit gegenüber einem solchen Angriff ist zwingend notwendig.

Der **Angriff mit bekanntem Klartext** (*known plaintext attack*) ist ein Szenario, in dem der Kryptoanalyse eine Anzahl von Klartext-Chiffretextpaaren zur Verfügung steht, die mit dem gleichen Schlüssel verschlüsselt wurden. Ziel ist die Berechnung dieses Schlüssels, um weitere Nachrichten entschlüsseln oder selbst erzeugen zu können.

Dieses Szenario ist nicht unrealistisch, da in der Praxis oft gegen einfache Grundregeln für die Verwendung kryptographischer Methoden und Protokolle verstoßen wird, sodaß eine Nachricht, die als Klartext über einen Kanal gesandt wurde, ohne syntaktische Veränderung auf einem verschlüsselten Kanal weitergesandt wird. Kann man beide Kanäle abhören und die Paare zuordnen, ist ein Angriff mit bekanntem Klartext möglich. Von modernen Chiffren wird auch Sicherheit gegenüber einem solchen Angriff gefordert [17, S. 99].

Die dritte klassische Angriffsvariante ist der **Angriff mit wählbarem Klartext** (*chosen plaintext attack*). Dem Angreifer steht die Möglichkeit offen, einen beliebigen Klartext verschlüsseln zu lassen und den zugehörigen Chiffretext abzufangen. Ziel ist die Berechnung des verwendeten Schlüssels.

Auch dieses Szenario ist nicht völlig unrealistisch: man muß den Betreiber eines sicheren Kanals nur dazu bringen, eine bestimmte Nachricht übertragen zu wollen [15, S. 140].

Ein relativ praxisnahes Beispiel wäre der Diebstahl einer *black box*, die zwar nach einem bekannten Verfahren, aber mit einem unbekannten Schlüssel arbeitet und die gegen das Auslesen des Schlüssels gesichert ist.

Nachfolgend werden noch andere Angriffsarten dargestellt, die die Eigenschaften bestimmter Chiffren ausnutzen. Grundsätzlich bleibt aber für alle Angriffsarten und alle[5] Chiffren das Problem der Beweisbarkeit ungelöst: es läßt sich unter Umständen nachweisen, daß eine Chiffre unsicher gegenüber einem Angriff ist. Sicherheit gegenüber einem Angriff ist jedoch nicht beweisbar [17, S. 101].

2.3.2.2 Techniken

Die Verfahren der Kryptoanalyse sind vielfältig und können hier nicht systematisch vorgestellt werden. Hier werden nur kurz die im nachfolgenden Text auftretenden Begriffe

[4] Dies ist eine Schwäche, die erst mit der Verwendung asymmetrischer Verschlüsselungsverfahren, digitaler Unterschriften und der dazugehörigen Protokolle behoben wird. Auf weitere Schwächen symmetrischer Verfahren wird in Abschnitt 2.3.6 eingegangen.

[5] Zumindest für alle praktikablen Chiffren, nicht für das *one-time-pad*. Siehe z.B. [15, S. 117] und [16, S. 15].

erläutert. Häufig dienen die Analyseverfahren nicht direkt dazu, in einem der oben dargestellten Szenarien das gesteckte Ziel zu erreichen, sondern „nur“, um strukturelle Schwächen in Algorithmen zu finden.

Zu den Analyseverfahren von eher theoretischem Interesse gehören z.B. die Differentielle Kryptoanalyse ([16, S. 284], [17, S. 205]) und die Lineare Kryptoanalyse ([16, S. 348], [17, S. 247]). Bei schwachen Chiffren können sie jedoch von praktischem Wert sein [17, S. 249].

Die meistbeachtetste „Technik“ mit praktischer Bedeutung ist die Erschöpfende Suche (*exhaustive search*, *brute force attack*): man durchsucht im Worst Case den gesamten Schlüsselraum, indem man jeden möglichen Schlüssel am Chiffretext ausprobiert und prüft, ob der erhaltene Klartext sinnvoll scheint. Für einen n Bit langen Schlüssel benötigt man also im Mittel etwa 2^{n-1} Versuche.

2.3.3 DES (Data Encryption Standard)

Das amerikanische National Bureau of Standards (NBS, heute National Institute of Standards and Technology (NIST)) forderte am 15. Mai 1973 auf, Vorschläge für ein öffentlich verfügbares Kryptoverfahren einzureichen ([16, S. 265], [18, S. 64], [20, S. 179], [21, S. 47]). Folgende, etwas vage Entwurfskriterien wurden in dieser Aufforderung vorgeschrieben:

1. Das Verfahren soll sehr sicher sein.
2. Das Verfahren soll vollständig spezifiziert werden und leicht zu verstehen sein.
3. Die Sicherheit des Verfahrens muß allein durch die Geheimhaltung der Schlüssel gewährleistet sein, die Geheimhaltung des Verfahrens darf für die Sicherheit keine Bedeutung haben[6].
4. Das Verfahren soll jedem Anwender zur Verfügung stehen.
5. Das Verfahren soll an diverse Anwendungen anpaßbar sein.
6. Das Verfahren soll zu geringen Kosten in Hardware realisierbar sein.
7. Das Verfahren soll effizient sein.
8. Das Verfahren soll validierbar sein.
9. Der Export des Verfahrens soll möglich sein.

Punkt zwei bedeutet nicht etwa, daß sämtliche Details des Verfahrens veröffentlicht werden sollten. Laut Schneier hat die amerikanische National Security Agency (NSA) niemals zugestimmt, soviele Informationen über DES zu veröffentlichen, daß der Algorithmus von jedermann in Software realisiert werden kann [16, S. 267]. In diesem Zusammenhang ist wohl auch Punkt neun so zu verstehen, daß Realisierungen in Hardware exportfähig sein sollten. Der Export von Hard- und Softwarerealisierungen des DES-Algorithmus aus den USA ist bis heute unverständlicherweise nicht gestattet, obwohl der DES-Algorithmus schon lange in Hard- und Software außerhalb der USA implementiert wird.

Trotz Vorbehalten war das Vorgehen des NBS „revolutionär“: hier sollte Kerckhoffs Maxime zum ersten mal voll umgesetzt werden.

Punkt acht bedingt, daß der Algorithmus modular aufgebaut sein muß, sodaß eine Implementierung validiert werden kann, indem alle Module und Schnittstellen validiert werden. Ein Algorithmus der nur über sein Ein-/Ausgabeverhalten definiert ist, kann in einer Implementierung im allgemeinen nur durch das Überprüfen aller möglichen Eingaben (Klartexte und Schlüssel) validiert werden. Sollte dies mit akzeptablen Aufwand möglich sein, so wäre der Algorithmus mit dem gleichen Aufwand zu brechen.

Die obigen Forderungen waren für die damaligen Kryptologen jedenfalls sehr hoch gesteckt, denn es dauerte mehr als drei Jahre bis der DES am 15. Januar 1977 durch die Federal Information Processing Standard Publication 46 „Data Encryption Standard“ [22, 23] als verbindlicher Standard festgeschrieben wurde.

DES ist eine Variation des von IBM entwickelten und eingereichten Verfahrens LUCIFER. Der Einfluß der NSA auf die Entwicklung des DES, sowie die Möglichkeit einer versteckten „Hintertür“, die der NSA das vergleichsweise leichte Mitlesen erlauben könnte, geben noch heute Anlaß zu Spekulationen ([16, S. 278], [18, S. 66], [20, S. 180], [21, S. 72]).

Diese Spekulationen erhielten neue Nahrung als Eli Biham und Adi Shamir 1990 ihre Methode der Differentiellen Kryptoanalyse veröffentlichten. In den folgenden Jahren wurde DES auf seine Anfälligkeit gegenüber dieser Analysemethode untersucht. Hier und gegenüber anderen Methoden erwies sich DES als überraschend stark, sodaß man annehmen muß, die NSA habe diese und andere Analysemethoden schon 15 Jahre früher gekannt als die Forscher

[6] Kerckhoffs Maxime in moderner Form.

in zivilen Organisationen ([16, S. 285 und S. 598], [18, S. 66], [17, S. 245]). Diese Vermutung gibt Anlaß zur Sorge, denn möglicherweise reicht der momentane Wissensvorsprung der NSA auch aus, um moderne Verfahren zu brechen, die heute noch als sicher gelten.

2.3.3.1 Beschreibung des DES

DES arbeitet auf Blöcken (Alphabetzeichen) der Länge 64 Bit. Die verwendeten Schlüssel haben eine Länge von 56 Bit, werden aber meist als 64 Bit lange Zahlen dargestellt. Die niederwertigsten Bits in jedem Byte sind dann bedeutungslos oder werden als Paritätsbits genutzt.

Die Permutationen IP und IP^{-1} haben keine kryptologische Bedeutung und dienen nur der effizienten Realisierung in Hardware auf 8 Bit breiten Bussen [16, S. 271]. Der Klartextblock wird zunächst in zwei Hälften geteilt. DES besteht im wesentlichen aus der sechzehnfachen Hintereinanderausführung einer relativ einfachen Operation (siehe Abbildung 2.1 auf Seite 30).

Diese sechzehn sogenannten Runden bilden ein *Feistel-Netzwerk*. Diese Anordnung bewirkt, daß die gesamte Abbildung umkehrbar wird, gleichgültig, welche Eigenschaften die Rundenfunktion f besitzt. Für die beiden Teilworte in einem Schritt gilt:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \end{aligned}$$

Bei der Entschlüsselung sind L_i, R_i und K_i bekannt während L_{i-1} und R_{i-1} berechnet werden müssen. Es gilt aber:

$$\begin{aligned} R_{i-1} &= L_i \\ L_{i-1} &= R_i \oplus f(R_{i-1}, K_i) \end{aligned}$$

Damit ist die Entschlüsselung unabhängig von Eigenschaften der Funktion f immer möglich. Feistel-Netze stellen eine mögliche Grundstruktur für symmetrische Schlüsselverfahren dar, die jedoch bei den nachfolgend vorgestellten Verfahren IDEA und RC4 nicht verwendet wird. Weitere Informationen zu Feistel-Chiffren und weitere Beispiele finden sich in [17, S. 185] und [16, S. 347].

Die Generierung der Teilschlüssel, sowie die Wahl der Rundenfunktion f sind Gegenstand zahlreicher Analysen und Spekulationen. Für genauere Darstellungen siehe [24], [16, Kapitel 12], [17, Kapitel 4], [15, S. 130], [21, Kapitel 3].

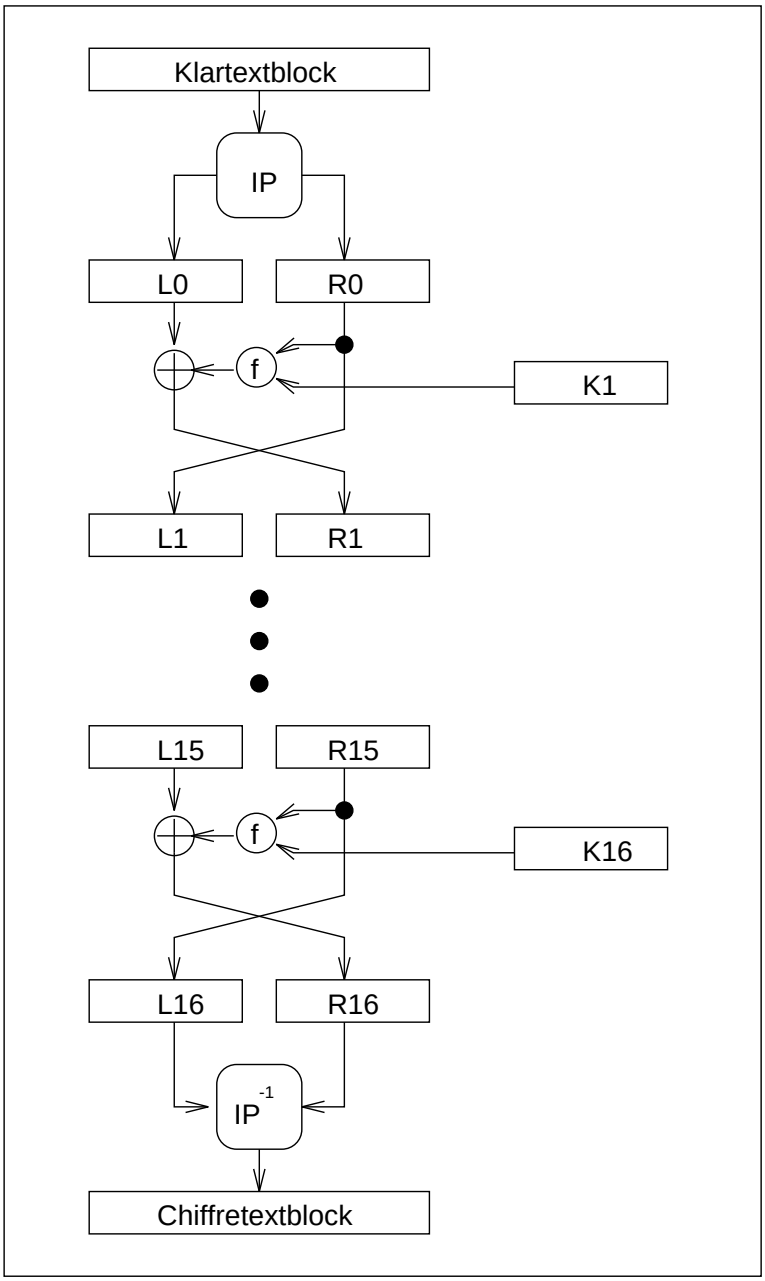

Abbildung 2.1: Grobe Struktur der DES-Verschlüsselung. $\oplus$ bedeutet XOR. f heißt Rundenfunktion. K_1 bis K_{16} sind Teilschlüssel, die aus dem (externen) DES-Schlüssel erzeugt werden.

2.3.3.2 Die Sicherheit von DES

Der DES-Algorithmus ist eine Blockchiffre, d.h. er besitzt keinen Speicher für Zustandsinformation. Gleiche 64 Bit-Blöcke des Klartextes werden auf gleiche Blöcke im Chiffretext abgebildet. Dadurch bleiben evtl. Strukturen des Klartextes erkennbar und es ist ohne die Zuhilfenahme von Prüfsummen nicht erkennbar, wenn einzelne Blöcke eingefügt, gelöscht oder geändert werden.

Die Wahrscheinlichkeit für mehrfach auftretende Blöcke im Klartext sinkt mit ansteigender Blockgröße. DES wird oft wegen der geringen Blockgröße kritisiert. Der IDEA-Algorithmus nimmt diese Kritik auf und erweitert die Blockgröße auf 128 Bit. Eine Möglichkeit, das Auftreten von gleichen Blöcken unwahrscheinlich zu machen, ist die Verwendung einer Datenkompression vor der Verschlüsselung. Eine andere Möglichkeit besteht darin, DES zu einer sogenannten Stromchiffre zu machen, die einen Zustand speichert, der sich mit jedem verschlüsselten Block ändert (siehe Abschnitt „Betriebsarten für DES“).

Ein weiterer Kritikpunkt an DES ist der kleine Schlüsselraum, der eine Erschöpfende Suche mit Spezialhardware als möglich erscheinen läßt. Siehe dazu z.B. [16, S. 300], [17, S. 243 u. S. 247]. Auch hier bietet IDEA Abhilfe, indem die Schlüssellänge auf 128 Bit vergrößert wird. Ein Versuch, diesen Mangel zu beheben ist das sogenannte Triple-DES-Verfahren (auch DES3), das aus der Hintereinanderanwendung von Ver-, Ent- und wiederum Verschlüsselung mit drei verschiedenen DES-Schlüsseln besteht ([16, S. 357], [17, S. 275], [15, Kapitel 9]).

Hier lassen sich unmöglich alle Spekulationen und Analysen andeuten, die im Zusammenhang mit DES veröffentlicht wurden. Dies liegt daran, daß – etwas überspitzt formuliert – die Kryptologie der Nachkriegszeit mit der Analyse des DES zusammenfällt. Dem interessierten Leser sei das deutschsprachige Buch von Fumy und Rieß empfohlen [17] und dem sehr interessierten Leser die etwa 1600 Einträge starke Literaturliste aus [16].

Bei aller Kritik an Blockgrößen und Schlüssellängen des DES-Algorithmus läßt sich zusammenfassend sagen, daß es auch heute, zwanzig Jahre nach Einführung des DES, kaum gelingen würde, eine Blockchiffre mit genau dieser Block- und Schlüssellänge zu entwerfen, die sicherer wäre als DES.

2.3.3.3 Betriebsarten für DES

Wichtige Entwurfskriterien für eine symmetrische Chiffren sind die *Konfusion*, die *Diffusion* und der *Avalanche-* oder *Lawinen-Effekt*.

Konfusion bedeutet das Verschleiern oder Vermeiden einer einfach formulierbaren funktionalen Abhängigkeit zwischen Chiffretext und Klartext. Eine zufällig gewählte Permutation des Alphabets ist ein Beispiel für eine schlecht funktional formulierbare Abhängigkeit.

Das Prinzip der *Diffusion* bedeutet das Verteilen der Redundanz des Klartextes über weite Teile des Chiffretextes, sodaß ein Bit des Chiffretextes von vielen, voneinander entfernten Bits des Klartextes abhängt.

Der sogenannte *Avalanche-Effekt* bedeutet, daß das i-te Bit des Chiffretextes möglichst von allen Schlüsselbits und von den Bits Nummer 0 bis Nummer i des Klartextes abhängen soll. Eine gute Einführung bieten [17, Kapitel 4] und [15, Kapitel 10].

Wie oben erwähnt bildet eine Blockchiffre grundsätzlich immer gleiche Klartextblöcke auf gleiche Chiffretextblöcke ab. Somit läßt sich ein Entwurfskriterium, der oben erwähnte Avalanche- oder Lawinen-Effekt mit einer reinen Blockchiffre nicht erreichen, da ein Bit im Chiffretext maximal von allen Bits im entsprechenden Block des Klartextes abhängen kann. Der gesamte vorangegangene Klartext hat keine Bedeutung.

In „DES Modes of Operation“ [25] wurden 1980 vier verschiedene Betriebsarten für DES standardisiert, mit deren Hilfe aus der Blockchiffre DES eine Stromchiffre konstruiert werden kann. Diese Betriebsarten sind grundsätzlich für jede Blockchiffre anwendbar [17, S. 265]. Sei P_i der i-te Block des Klartextes und sei C_i der i-te Block des Chiffretextes, $i > 0$ und F_k der Verschlüsselungsschritt der Blockchiffre. Die vier Betriebsarten arbeiten wiefolgt:

Electronic Codebook Mode (ECB) Diese Betriebsart ist nichts Neues. Die Blockchiffre wird in der üblichen Weise angewandt: $C_i = F_k(P_i)$.

Cipher Block Chaining Mode (CBC) Zur Berechnung von C_i wird auch C_{i-1} herangezogen: $C_i = F_k(P_i \oplus C_{i-1})$. Da C_0 nicht existiert, benötigt man zur Verschlüsselung des ersten Blocks einen sogenannten Initialisierungsvektor IV, der als Teil des Schlüssels betrachtet werden kann, aber nicht muß. Im Extremfall kann man IV sogar offen als ersten Block der Nachricht übertragen, was eine zufällige Wahl von häufig wechselnden Werten für IV erleichtert.

Output Feedback Mode (OFB) Die Blockchiffre wird zur Erzeugung eines Stroms von Pseudozufallszahlen R_i benutzt: $R_0 = IV$ und $R_i = F_k(R_{i-1})$. Für den Chiffretext gilt dann: $C_i = P_i \oplus R_i$. Für den Initialisierungsvektor gilt dasselbe wie oben.

Cipher Feedback Mode (CFB) Dieser Betriebsart ist dem OFB-Modus sehr ähnlich, der einzige Unterschied besteht in der Erzeugung des Stroms von Pseudozufallszahlen: $R_0 = IV$ und $R_i = F_k(C_{i-1})$.

2.3.4 IDEA (International Data Encryption Algorithm)

2.3.4.1 Beschreibung von IDEA

IDEA ist eine symmetrische Blockchiffre, die wie DES mit 64 Bit-Blöcken (Alphabetzeichen) arbeitet. Die verwendeten Schlüssel sind 128 Bit lang. Das Verfahren ist durch seine Verwendung im frei verfügbaren Programm `pgp` (Pretty Good Privacy) bekannt geworden. Auch IDEA verwendet eine relativ einfache Transformation, die mehrere Male hintereinander mit verschiedenen Teilschlüsseln ausgeführt wird.

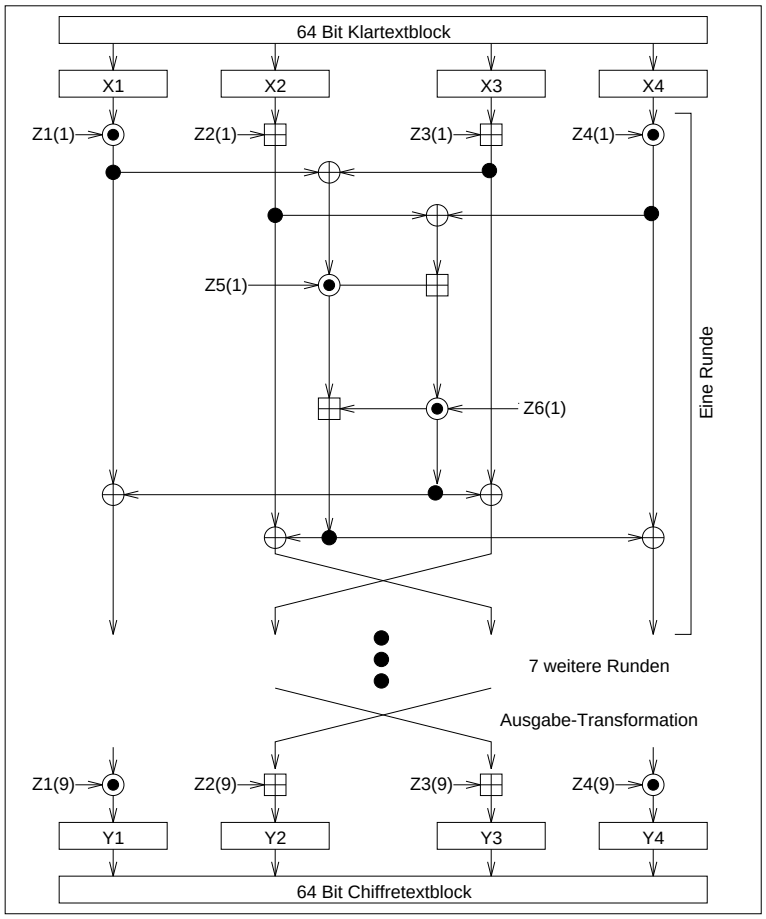

Abbildung 2.2: Struktur der IDEA-Verschlüsselung

IDEA arbeitet nicht mit Substitutionen sondern vermischt in jeder einzelnen von acht Runden verschiedene arithmetische Operationen, für die keine Distributivgesetze gelten, sodaß die Abhängigkeit des Chiffretextes vom Klartext nur als sehr komplexer algebraischer Ausdruck darstellbar ist (siehe Abbildung 2.2).

Die drei Operationen sind XOR ($\oplus$), Addition modulo 2^{16} ($\boxplus$) und Multiplikation modulo $2^{16}+1$ ($\odot$)[7]. Diese Operationen werden in jeder Runde auf 16 Bit breiten Teilblöcken ausgeführt, sodaß IDEA auch auf „kleinen“ Prozessoren effizient implementierbar ist.

Die Umkehrbarkeit der Verschlüsselung wird hier dadurch erreicht, daß alle verwendeten Operationen bei bekanntem Schlüssel umkehrbar sind[8], eine Feistel-Struktur ist also nicht notwendig.

[7] Mit der zusätzlichen Definition, daß das Nullwort als 2^{16} interpretiert wird.

[8] Die vorzeichenlosen 16Bit-Ganzzahlen bilden mit jeder der drei Operationen eine Gruppe.

Für eine genauere Darstellung vor allem zur Erzeugung der Teilschlüssel $Z_j^{(i)}$ siehe [26] und [16, S. 319]. IDEA kann wie DES in den im Abschnitt 2.3.3.3 dargestellten Betriebsarten benutzt werden.

2.3.4.2 Die Sicherheit von IDEA

Gegenüber der Erschöpfenden Suche ist IDEA natürlich 2^{128-56} mal sicherer als DES. Im Vergleich zu DES ist das IDEA-Verfahren viel zu neu, um jetzt schon vollständig analysiert worden zu sein. Beim Versuch, bekannte Analyseverfahren auf IDEA anzuwenden, erwies sich IDEA als stark.

Bruce Schneier meint dazu [16, S. 319]:

> IDEA is based on some impressive theoretical foundations and, although cryptoanalysis has made some progress against reduced-round variants, the algorithm still seems strong. In my opinion, it is the best and most secure block algorithm available to the public at this time.

Und etwas später [16, S. 323]:

> IDEA's key length ist 128 bits – over twice as long as DES. Assuming that a brute-force attack is the most efficient, it would require 2^{128} (10^{38}) encryptions to recover the key. Design a chip that can test a billion† keys per second and throw a billion of them at the problem, and it will still take 10^{13} years – that's longer than the age of the universe. An array of 10^{24} of such chips can find the key in a day, but there aren't enough silicon atoms in the universe to build such a machine. Now we're getting somewhere – although I'd keep my eye on the dark matter debate.
>
> Perhaps brute force isn't the best way to attack IDEA.

Ein anderer wichtiger Punkt im Vergleich zu DES ist die Abwesenheit von Konstanten. Die durch die NSA fest vorgegebenen Permutationen in der Rundenfunktion des DES-Algorithmus und die unbekannten Entwurfskriterien für die Auswahl der Konstanten haben immer wieder zu dem Verdacht geführt, die NSA habe eine „unsichtbare Hintertür" in den DES-Algorithmus eingebaut. Die Entwurfskriterien für IDEA sind klar und es gibt kaum ein Detail in IDEA, das nicht in [26] durch nachvollziehbare Kriterien begründet wird.

†Im amerikanischen Sprachgebrauch entspricht *a billion* unserer Milliarde, also 10^9.

2.3.5 RC4 (Rivest's Cipher oder Ron's Code)

RC4 ist eine von Ronald Rivest für die Firma *RSA Data Security, Inc.* entwickelte Stromchiffre, die auf einzelnen Bytes als Alphabetzeichen arbeitet. Eine Besonderheit von RC4 ist die variable Schlüssellänge.

RC4 ist vorallem durch seine Verwendung in den Produkten der Firma *Netscape Communications Corporation* bekannt geworden, wird aber auch in Programmen wie *Lotus Notes* und *Oracle Secure SQL* eingesetzt. Die Firma RSADSI hat versucht, die Details über die RC4-Chiffre geheimzuhalten, was offensichtlich nicht gelungen ist.

2.3.5.1 Beschreibung von RC4

RC4 arbeitet mit zwei 256 Bytes großen Feldern S und K, die zu Beginn wiefolgt initialisiert werden:

$S_0 = 0, \ldots, S_{255} = 255$ und das Feld K wird mit den Schlüsselbytes gefüllt. Ist der Schlüssel kürzer als 256 Bytes (immerhin 2048 Bits), so wird er beim Füllen einfach wiederholt. Dann wird das Feld S permutiert:

```
j=0
for i=0 to 255 do
  j=(j + S[i] + K[i]) mod 256
  swap( S[i], S[j] )
od
```

Für die eigentliche Verschlüsselung werden nun „Zufallsbytes“ generiert die mit dem Klartext- bzw. Chiffretextstrom durch XOR verknüpft werden. Dies entspricht der Betriebsart OFB, siehe Abschnitt 2.3.3.3 auf Seite 31. Ein „zufälliges“ Byte Z wird wiefolgt generiert:

```
i=(i + 1   ) mod 256 // i und j werden zu Beginn
j=(j + S[i]) mod 256 // mit 0 initialisiert
swap( S[i], S[j] )
t=( S[i] + S[j] ) mod 256
Z=S[t]
```

Und das ist alles! Implementierungen dieses Algorithmus' sind etwa 10 mal schneller als für DES.

Aufgrund der variablen Schlüssellänge kann RC4 auch mit einem 40 Bit langen Schlüssel betrieben werden (bzw. mit einem längeren Schlüssel, von dem nur 40 Bit geheimgehalten werden). In einer solch abgeschwächten Form ist die Ausfuhr von Softwareimplementierungen des RC4 aus den USA durch die NSA gestattet worden, während der Export von Software, die den DES-Algorithmus in einer ähnlich abgeschwächten Form verwendet, nicht gestattet wurde. Dieses Faktum sollte zumindest mißtrauisch machen, obwohl bisher keine grundsätzlichen Schwächen von RC4 bekanntgeworden sind.

2.3.5.2 Die Sicherheit von RC4

In [16, S. 397] wird die Frage der Sicherheit von RC4 zwar kommentiert, aber es wird keine entgültige Empfehlung oder Einschätzung gegeben.

2.3.6 Schwächen symmetrischer Verfahren

Für eine Teilnehmerzahl n müssen bei Verwendung eines symmetrischen Kryptoverfahrens $\frac{n(n-1)}{2} = O(n^2)$ Schlüssel zur Verfügung stehen.

Dies wird insbesondere zum Problem, da die Schlüssel in regelmäßigen Intervallen ausgetauscht werden sollten. Unglücklicherweise benötigt man zur Verteilung der neuen Schlüssel einen anderen Kanal als denjenigen, der mit den alten Schlüsseln abgesichert wurde. Ansonsten würde einem Angreifer das Wissen über einen alten Schlüssel genügen, um den neuen zu erfahren und der Schlüsselaustausch wäre sinnlos.

Selbst, wenn man von diesem Problem absieht, müßte man bei Aufnahme des Teilnehmers $n+1$ in das bestehende System n neue Schlüssel generieren und verteilen. Simson Garfinkel schreibt dazu in [18, 69]:

> ...the U.S. government solved this key distribution problem with a simple if crude technique: cryptographic keys were placed in locked briefcases that were handcuffed to couriers who physically transported them from Washington to embassies and consulates around the world.

Dieses Vorgehen ist für Organisationen, die kostendeckend arbeiten müssen, sicher nicht praktikabel. Eine Möglichkeit zur Lösung dieses Problems ist das Einführen einer Institution, der die beiden traditionellen Kommunikationspartner Alice und Bob trauen. Alice und Bob haben keinen gemeinsamen geheimen Schlüssel vereinbart, aber sie haben je einen geheimen Schlüssel mit einem *key distribution center* (KDC) vereinbart. Durch die Vermittlung des KDC können beide beliebig oft einen gemeinsamen Schlüssel für eine Verbindungsaufnahme vereinbaren, einen sogenannten *session key*[9]. Das KDC ist der „wunde Punkt“ in diesem System. Es ist nicht auf einfache Weise möglich, miteinander zu kommunizieren, wenn man nicht dem gleichen KDC vertraut. Zum anderen kann ein KDC jederzeit die Identität eines seiner Clienten annehmen.

Es gilt also die folgenden drei Probleme zu lösen:

Schlüsselverteilung (*key management*) Wie werden Schlüssel unter den Teilnehmern verteilt?

Spontane Kommunikation Wie können zwei Teilnehmer spontan (d.h. auch erstmalig) miteinander kommunizieren? Und weiter: wie kann man Authentisierung bei spontaner Kommunikation leisten?

Nichtabstreitbarkeit Wie kann man von einem Dokument beweisbar auf seinen Urheber schliessen? Oder pragmatischer: wie erzeugt man Digitale Unterschriften?

Diese Probleme schienen lange Zeit unlösbar, da sie nicht als Eigenschaft einer Verfahrensklasse gesehen wurden, sondern jedem kryptographischen Verfahren anzuhaften schienen. Der folgende Abschnitt beschäftigt sich mit der Widerlegung dieser Auffassung.

[9] Nach diesem Schema arbeitet das Kerberos-System [27, 28].

2.4 Asymmetrische Verfahren

Nachdem etwa viertausend Jahre lang ausschließlich symmetrische Verschlüsselungsverfahren benutzt worden waren, war der Gedanke, daß die bisher entwickelten Chiffren eine Verfahrensklasse bilden und, daß es demzufolge andere Verfahrenklassen geben könnte, revolutionär. Dieser erneuernde Gedanke wurde erstmals 1976 in dem Artikel „New Directions in Cryptography“ von Withfield Diffie und Martin Hellman [29] publiziert.

Hier wurde ein Kryptosystem gefordert, das jedem Teilnehemer einen öffentlichen und einen privaten Schlüssel zuordnet. Will Alice eine Nachricht an Bob senden, so schlägt sie Bob's öffentlichen Schlüssel in einer Art Telefonbuch nach, verschlüsselt die Nachricht und versendet sie. Nur Bob kennt seinen zugehörigen privaten Schlüssel und kann die Nachricht entschlüsseln.

Weiter sollte es das Kryptosystem im Idealfalle jedem seiner Teilnehmer ermöglichen, eine Nachricht mit Hilfe seines privaten Schlüssels zu unterschreiben. Diese Unterschrift sollte wiederum mit dem zugehörigen öffentlichen Schlüssel überprüfbar sein. in unserem Beispiel unterschreibt Alice mit ihrem privaten Schlüssel und Bob überprüft diese Unterschrift mit Alice öffentlichen Schlüssel aus dem Telefonbuch.

Ein solches Verfahren eröffnet der Anwendung von Kryptographie in kommerziellen Massenanwendungen völlig neue Möglichkeiten, schafft diese überhaupt erst. Das große Potential eines solchen Systems wurde schnell erkannt und man begann die Suche nach einem solchen System.

Zunächst wurden Systeme vorgeschlagen, die die gegen Ende des vorigen Abschnitts dargestellten Probleme nur zum Teil lösten. Dazu gehören die drei im folgenden beschriebenen Verfahren: Merkle-Puzzles, Knapsack-Verfahren und das Diffie-Hellman-Verfahren. Alle drei sind in der Praxis bedeutungslos.

Das vierte beschriebene asymmetrische Verfahren ist das RSA-Verfahren. Es ist bislang das einzige asymmetrische Verfahren mit praktischer Bedeutung und liegt allen Vorschlägen für kryptographische Protokolle zugrunde.

Nach der Beschreibung der einzelnen Verfahren wird ein Prototyp für kryptographische Protokolle entwickelt, der symmetrische und asymmetrische Verfahren kombiniert.

2.4.1 Merkle-Puzzles

Merkle-Puzzles dienen zur spontanen Vereinbarung eines Schlüssel für ein beliebiges, festes symmetrisches Verfahren. Wieder will Alice Kontakt zu Bob aufnehmen.

Sie versteckt n verschiedene Schlüssel $k_1, \ldots, k_n$ in n Puzzles $p_1, \ldots, p_n$. Diese Puzzles kann man sich zum Beispiel als Gleichungen vorstellen, deren numerische Lösung als Schlüssel interpretiert wird. Diese Puzzles werden nun an Bob geschickt.

Bob wählt ein Puzzle p_i aus, löst es und antwortet unter Verwendung des Schlüssels k_i. Alice probiert jetzt alle Schlüssel $k_1 \ldots, k_n$ aus, bis sie k_i als richtigen Schlüssel findet.

Ein Angreifer, der die Verbindung abhört, muß im Mittel $n/2$ Puzzles lösen und die enthaltenen Schlüssel ausprobieren, um die Nachrichten entschlüsseln zu können.

Das große Problem der Merkle-Puzzles ist der verhältnismäßig geringe Mehraufwand, den ein Angreifer im Vergleich mit den beiden Teilnehmern bewältigen muß. Immer wenn man Werte für n und bestimmte Realisierungen der Puzzles wählt, die einen für Alice und Bob praktikablen Aufwand verursachen, wird auch das Brechen des Verfahrens für einen motivierten Angreifer praktikabel. Dieses Problem verschärft sich insbesondere, wenn der Inhalt der Nachricht auch noch über lange Zeit geheim bleiben soll, da das Lösen der Puzzles mit immer leistungsfähigerer Hardware immer einfacher wird.

Ein anderes Problem ist die Notwendigkeit einer *on line*-Verhandlung zwischen den beiden Teilnehmern, wie sie z.B. bei einem *email*-System nicht gegeben ist.

Für weitere Informationen zu Merkle-Puzzles siehe [16, S. 34], [18, S. 69] und [30, S. 16].

2.4.2 Knapsack-Verfahren

Das in [30, S. 62] beschriebene Knapsack-Verfahren ist ein Verfahren zur Verschlüsselung, aus dem auch ein Verfahren zur Erzeugung digitaler Unterschriften konstruiert werden kann.

Bob generiert zunächst einen sogenannten *superimposing Knapsack*[10], d.h. einen Vektor $\underline{a} = (a_1, \ldots, a_n)$ mit der Eigenschaft

$$a_i > \sum_{j=1}^{i-1} a_j$$

Eine Instanz des Knapsack-Problems ist eine positive Ganzzahl s für die ein binärer Vektor $\underline{x}$ gesucht wird mit

$$s = \underline{x} \cdot \underline{a}$$

Man sucht also eine einfache Summe aus den a_i, die s ergibt. Für einen *superimposing knapsack* findet man die Lösung leicht, wie man sich schnell klarmacht.

Der Teilnehmer berechnet nun aus seinem Vektor $\underline{a}$ einen neuen $\underline{b}$ mit $b_i = a_i \cdot w \bmod m$, wobei w invertierbar sein muß, d.h. es existiert ein w^{-1} mit $1 = w \cdot w^{-1} \bmod m$. Dieser Vektor ist nicht mehr *superimposing*.

Der Vektor $\underline{b}$ wird nun als Bobs öffentlicher Schlüssel bekannt gemacht, während w und m geheim bleiben. Will Alice einen n Bit langen String $\underline{t}$ an Bob senden, so berechnet sie $T_b = \underline{t} \cdot \underline{b}$ und sendet T. Bob berechnet $T_a = T_b \cdot w^{-1} \bmod m$ mit der Eigenschaft $T_a = \underline{t} \cdot \underline{a}$ und löst das einfache Problem für einen *superimposing knapsack*.

Leider stellt sich heraus, daß dieses System und einige vorgeschlagene Varianten leicht gebrochen werden können ([16, S.462], [18, S. 79]).

2.4.3 Diffie-Hellman

Das Diffie-Hellman-Verfahren ermöglicht das spontane Verhandeln zweier asymmetrischer Schlüsselpaare auf einem unsicheren Kanal. D.h hier wird ebenso wie bei den Merkle-Puzzles eine *on line*-Verbindung vorausgesetzt.

Alice und Bob einigen sich auf zwei Zahlen g und n. n ist eine große Primzahl und g ist ein Generator für n:

$$\bigwedge_{b \in \{1,\ldots,n-1\}} \bigvee_{a} g^a = b \bmod n$$

Die Zahlen g und n können über einen unsicheren Kanal vereinbart werden, sie können sogar gemeinsam von einer größeren Gruppe von Teilnehmern benutzt werden.

Alice und Bob bereiten einen sicheren Nachrichtenaustausch nun *online* vor:

Alice würfelt eine große Zahl x und sendet X mit $X = g^x \bmod n$ an Bob.

Bob würfelt y und schickt Y mit $Y = g^y \bmod n$ zurück.

Alice berechnet $k = Y^x \bmod n$.

Bob berechnet $k' = X^y \bmod n$.

Nun gilt $k = k' = g^{x \cdot y} \bmod n$. Beim Abhören der Verbindung erhält man nur Kenntnis von g, n, X und Y. Die Sicherheit des Verfahrens beruht auf dem Problem des diskreten Logarithmus.

Bislang sind keine einfachen Methoden bekannt, Diffie-Hellman zu brechen. Trotzdem hat dieses Verfahren wegen seiner eingeschränkten Nutzbarkeit keine praktische Bedeutung[11]. Für genauere Informationen siehe [16, S. 513] und [18, S. 72].

[10] Etwa durch „überaufsteigend“ oder besser gar nicht zu übersetzen.

[11] Abgesehen von seiner Berücksichtigung im Secure Sockets Layer [31], hier macht ein Server nicht nur g und n sondern auch eine feste Wahl von X bekannt, während der dazugehörige Wert x geheimgehalten wird. Dadurch ist keine *on-line*-Verhandlung des Schlüssels nötig, aber die Berechnung von x wird besonders bei einer großen Zahl von Clients sehr lohnend.

2.4.4 RSA (Rivest, Shamir, Adleman)

Das RSA-Verfahren verdankt seine Popularität dem Einsatz im Programm `pgp` (*pretty good privacy*) und in den Produkten der Firma *Netscape Communications Corporation*. Es ermöglicht Verschlüsselung, Authentisierung und die Erzeugung digitaler Unterschriften.

Die Sicherheit des RSA-Verfahrens beruht auf der Schwierigkeit der Faktorisierung großer Zahlen. Bisher ist kein Algorithmus mit polynomieller Laufzeit in der Bitlänge der zu faktorisierenden Zahl bekannt.

Ein Beweis, daß Faktorisierung nicht in polynomieller Laufzeit möglich ist, steht noch aus, doch es ist nicht zu erwarten, daß ein polynomieller Algorithmus gefunden wird. Die Sicherheit des RSA-Verfahrens ist eher durch die Entwicklung von Heuristiken zur Faktorisierung gefährdet, die sich einer strengen Laufzeitanalyse entziehen[12]. In [18, S. 361] wird versucht, den praktischen Aufwand an Zeit und Geld abzuschätzen, der zum Brechen der RSA-Chiffre erforderlich ist.

2.4.4.1 Beschreibung des RSA-Verfahrens

Bob möchte sich ein Schlüsselpaar, bestehend aus seinem privaten Schlüssel b und seinem öffentlichen Schlüssel B generieren. Er möchte B veröffentlichen, um es Alice (und allen anderen, die den Schlüssel kennen) zu ermöglichen, ihm verschlüsselte Nachrichten zu senden.

Bob würfelt zwei große Primzahlen p und q und setzt $n = p \cdot q$.

Nun würfelt er eine Zahl e (den **e***ncryption exponent*), die teilerfremd zu $\varphi(n)$ ist.

φ ist hier die sogenannte Euler-Funktion. $\varphi(n)$ gibt an, wieviele zu n teilerfremde Zahlen es in der Menge $\{2, \ldots, n-1\}$ gibt. Im allgemeinen läßt sie sich nur sehr schwer berechnen nämlich, indem man alle Zahlen aus $\{2, \ldots, n-1\}$ aufzählt und prüft, ob sie teilerfremd zu n sind. Ist n ausreichend groß, so ist dieses Verfahren aussichtslos. In gängigen Implementierungen liegt die Bitlänge von n zwischen 512 und 2048.

Für die spezielle Wahl von n als Produkt von zwei Primzahlen gilt aber $\varphi(n) = (p-1) \cdot (q-1)$. Zur einfachen Berechnung muß also die Faktorisierung von n bekannt sein.

Passend zu n und e wird nun eine Zahl d (**d***ecryption exponent*) berechnet, mit der Eigenschaft

$$e \cdot d \equiv 1 \bmod \varphi(n)$$

Der öffentliche Schlüssel ist nun $B = (e, n)$. Der private Schlüssel[13] ist $b = (d, n)$.

Alice verschlüsselt nun einen geeigneten[14] Klartext m in den Chiffretext M durch

$$M = E(m) = m^e \bmod n$$

Bob empfängt M und berechnet

$$D(M) = M^d \bmod n = m^{e \cdot d} \bmod n = m$$

Die letzte Gleichung gilt wegen der speziellen Wahl von e und d. Um den mathematischen Hintergrund etwas zu erhellen, seien dem Leser in der Reihenfolge aufsteigender Abstraktion [33, Kapitel 33], [34] und [35] empfohlen.

[12] So sind z.B. auch fast alle Plazierungs- und Verdrahtungsprobleme sogar nachweislich NP-vollständig. Trotzdem arbeiten die bekannten Heuristiken so gut, daß ich diesen Text nicht auf einer mechanischen Schreibmaschine tippen muß. Mehr zur Lösung schwerer Probleme findet sich z.B. in [32, *Solving NP-Hard Problems*].

[13] Für eine schnelle Entschlüsselung werden die Faktoren p und q meist auch mit dem privaten Schlüssel gespeichert. Prinzipiell ist dies jedoch unnötig.

[14] Der Klartext muß durch Ganzzahlen dargestellt werden, die kleiner n sind. Interessanterweise geht niemand auf das Problem ein, daß die Zahl Null bei der Verschlüsselung auf sich selbst abgebildet wird.

2.4.4.2 Digitale Unterschriften mit dem RSA-Verfahren

Sieht man sich den Verschlüsselungsschritt E und die Entschlüsselung D an, so fällt auf, daß nicht nur – wie oben benutzt – $D \circ E = 1$ gilt, sondern auch umgekehrt $E \circ D = 1$. Man kann also auch zuerst entschlüsseln und danach verschlüsseln und erhält wieder die gleiche Nachricht.

Genau diese Eigenschaft ermöglicht es, das RSA-Verfahren zur Erzeugung digitaler Unterschriften zu benutzen. Bob schickt eine Nachricht M, die zur Vereinfachung nicht verschlüsselt werden soll, und zusätzlich seine Unterschrift s unter M, die er wiefolgt berechnet hat:

$$s = D(M) = M^d \bmod n$$

Alice überprüft nun, ob die Gleichung $M = E(s)$ erfüllt ist

$$E(s) = E(D(M)) = M \quad \checkmark$$

Nun kann Alice mit an Sicherheit grenzender Wahrscheinlichkeit davon ausgehen, daß die Unterschrift von Bob erzeugt wurde, dem einzigen, der den Exponenten d kennt. Ebenso kann Bob die Unterschrift nicht abstreiten, es sei denn, er behauptet, er habe seinen privaten Schlüssel weitergegeben[15].

Die Eigenschaft von D und E, bezüglich Hintereinanderausführung in beiden Richtungen invers zueinander zu sein, ermöglicht eine neue Angriffsmethode, wenn das RSA-Verfahren in einem unglücklich konstruierten Protokoll eingesetzt wird:

> Unterschreibt Bob eine für ihn verschlüsselte Nachricht, so entschlüsselt er sie gleichzeitig.

Wie kann man diese Tatsache ausnutzen? Wird ein Teilnehmer durch ein Protokoll gezwungen, sehr viele Daten zu unterschreiben, wird man diesen Vorgang soweit automatisieren, daß eine Kontrolle jeder unterschriebenen Nachricht[16] nicht mehr möglich ist.

Ein Angreifer könnte nun eine Nachricht an Bob abfangen, die er zunächst nicht lesen könnte. Nun würde er unter einem Vorwand Kontakt zu Bob aufnehmen, wobei er das zweifelhafte Protokoll benutzt. Er würde nun die geheime Nachricht (blockweise) von Bob unterschreiben lassen, wodurch sie entschlüsselt und für ihn lesbar wäre.

Zum anderen ist klar, daß jede Unterschrift, die Bob leistet, implizite Information über den privaten Schlüssel enthält, die evtl. für einen Angriff genutzt werden kann. Manche Protokolle schlagen deshalb vor, für jeden Teilnehmer zwei Schlüsselpaare zu generieren: eines für den Austausch verschlüsselter Nachrichten und eines für die Erzeugung von Unterschriften.

Alle Protokolle erzeugen digitale Unterschriften mit dem RSA-Verfahren nicht direkt über der Nachricht M sondern über einem Hashwert $h(M)$. Das genaue Vorgehen wird in Abschnitt 2.5.2 vorgestellt, nachdem sichere Hashfunktionen in Abschnitt 2.5.1 eingeführt worden sind.

2.4.4.3 Die Sicherheit des RSA-Verfahrens

Es wird vermutet, daß kein Verfahren zur Faktorisierung mit polynomieller Laufzeit in der Bitlänge des Produkts (Modulus n) existiert. Diese Vermutung wird dadurch erhärtet, daß die Suche nach schnellen Algorithmen für dieses Problem sehr intensiv betrieben wird. In

[15] Hier werden schon die ersten Zweifel daran deutlich, ob digitale Unterschriften als vollständiger Ersatz für handschriftliche Unterschriften dienen können. Hat Bob im oben beschriebenen Fall durch die Weitergabe des privaten Schlüssels quasi eine Vollmacht ausgestellt?

[16] Im Zusammenhang mit Protokollen wird erläutert, daß digitale Unterschriften nicht über die gesamten Nachricht erzeugt werden, sondern über Hashwerte, die aus der Nachricht berechnet werden.

[33, S. 852] wird als bisher beste erreichte Laufzeit $O\left(e^{\sqrt{\ln n \ln\ln n}^{1+o(1)}}\right)$ angegeben. Damit ist das Problem der Faktorisierung sicher nicht NP-vollständig.

Ebenso wird vermutet, daß es keine Möglichkeit gibt, das RSA-Verfahren zu brechen, die weniger komplex wäre als die Faktorisierung des Modulus n. Auch diese Vermutung ist nicht bewiesen.

Das RSA-Verfahren löst aber noch nicht alle Probleme der Schlüsselverteilung. Jemand kann ohne weiteres einen Schlüssel publizieren, von dem er behauptet, er sei von Bob erzeugt worden. Benutzt Alice gutgläubig diesen Schlüssel, so kann der Angreifer ihre Nachrichten an Bob abhören und Alice gegenüber echt wirkende Unterschriften von Bob erzeugen.

Im Extremfall überwacht der Angreifer jeglichen Nachrichtenaustausch zwischen Alice und Bob. So kann er beiden einen selbsterzeugten Schlüssel für den jeweils anderen Kommunikationspartner vorgaukeln und die Kommunikation abhören und manipulieren, inklusive der Fälschung von Unterschriften (*man in the middle attack* nicht zu verwechseln mit der *meet in the middle attack*, die eine Analysestrategie bezeichnet [16, S. 358].).

Dadurch wäre es auch nutzlos, den veröffentlichten Schlüssel zu unterschreiben, da die Unterschrift nur mit eben diesem öffentlichen Schlüssel zu verifizieren ist.

Es besteht also keine Möglichkeit für Alice und Bob, sowohl spontan Kontakt aufzunehmen, als auch, sich gegenseitig zu authentisieren. Dieses Problem kann man durch die sogenannte Zertifizierung von öffentlichen Schlüsseln lösen.

Trotz der nicht beweisbaren Sicherheit und der weiterhin aufwendige Schlüsselverteilung ist das RSA-Verfahren zur Zeit das einzige *Public Key*-Verfahren mit praktischer Bedeutung und Aussicht auf Verbreitung in Protokollen für Standardanwendungen.

2.5 Digitale Unterschriften und Zertifikate

Nun stehen leistungsfähige symmetrische und asymmetrische Verschlüsselungsverfahren zur Verfügung. Schlüsselinformationen können bei Verwendung asymmetrischer Verfahren über unsichere Kanäle übermittelt werden, und das Problem der digitalen Unterschriften ist prinzipiell gelöst. Trotzdem fehlen noch einige Bausteine, um ein leistungsfähiges kryptographisches Protokoll zu entwerfen.

Das RSA-Verfahren ist etwa um den Faktor 100 langsamer als z.B. DES in der Betriebsart CBC. Im Abschnitt 2.6 wird beschrieben, wie durch geeignete Kombination von symmetrischen und asymmetrischen Verfahren die Vorteile beider Verfahrensklassen kombiniert werden können.

Digitale Unterschriften werden nur im Prinzip so erzeugt, wie im Abschnitt 2.4.4.2 dargestellt. Würde man so vorgehen, dann hätte eine Unterschrift per se die gleiche Größe wie das unterschriebene Dokument. Der Aufwand für das Erzeugen, Verifizieren, Speichern und Übertragen von Unterschriften wäre entsprechend groß. Zudem enthielten die Unterschriften zu viel implizite Information über den privaten Schlüssel. Dieses Problem wird durch sichere Hashfunktionen gelöst.

Bei der Beschreibung des RSA-Verfahrens wurde schon die Angriffsmöglichkeit durch das Veröffentlichen „gefälschter“ öffentlicher Schlüssel angedeutet. Mit Hilfe von digitalen Unterschriften werden sogenannte Zertifikate erstellt, die die Authentizität eines Schlüssels garantieren. Von dort ist der Weg zu Zertifizierungshierarchien nicht weit. Erst im anschließenden Abschnitt 2.6 werden die Bausteine zu einem Prototyp für kryptographische Protokolle zusammengesetzt.

2.5.1 Sichere Hashfunktionen

Sichere Hashfunktionen ermöglichen die Erstellung kompakter, leicht handhabbarer Unterschriften unter beliebigen Dokumenten. Die Idee ist, ein Dokument auf einen unverwechselbaren Fingerabdruck fester Länge abzubilden, der anstelle des Dokuments unterschrieben wird. Übliche Längen für einen solchen Fingerabdruck liegen bei 128 oder 160 Bit.

Natürlich kann eine solche Funktion nicht kollisionsfrei sein, da abzählbar viele verschiedene digitale Dokumente existieren. Oder? Der Trick besteht darin, daß diese Dokumente zwar theoretisch existieren, aber in der Realität werden nur vergleichsweise wenige Dokumente formuliert und noch weniger unterschrieben werden. Damit ein Fingerabdruck eindeutig für ein Dokument ist, reicht also Kollisionsfreiheit über allen in der Praxis auftretenden Dokumenten.

Hat man diese auf den ersten Blick paradoxe Forderung akzeptiert, erwartet einen sofort die nächste Überraschung: wurde gerade gefordert, daß eine sichere Hashfunktion bezogen auf alle in der Praxis vorkommende Dokumente eineindeutig sein muß, so fordert man jetzt zusätzlich, daß sie nicht mit vertretbarem Aufwand umkehrbar sein darf.

Warum stellt man diese zweite Forderung? Weil man unter keinen Umständen ein Dokument erzeugen können darf, das zu einer vorhandenen Unterschrift paßt, denn so würde jede aus der Hand gegebene Unterschrift einem Blankoscheck entsprechen. Genauer ausgedrückt möchte man, daß eine Hashfunktion h nicht mit gängigen Operationen auf Dokumenten verträglich ist: ist T eine Operation auf Nachrichten wie etwa Einfügen, Anfügen oder Streichen von Zeichen, Worten oder Blöcken und M eine Nachricht, dann darf keine Operation t gefunden werden, sodaß gilt:

$$a = h(M) \rightarrow t(a) = h(T(M))$$

Natürlich muß eine solche Operation t prinzipiell existieren. Damit läßt sich „t darf nicht gefunden werden können“ nicht beweisen. Genausowenig kann man die Kollisionsfreiheit von h über allen in der Praxis auftretenden Nachrichten beweisen.

Diese paradox scheinenden Anforderungen an sichere Hashfunktionen sind eng mit den Anforderungen an Chiffren verwandt: besitzt man den Schlüssel nicht, so soll der Zusammenhang zwischen Klartext und Chiffretext praktisch nicht formulierbar, geschweigedenn berechenbar sein; besitzt man den Schlüssel, sollen Klartext bzw. Chiffretext leicht und schnell berechnet werden können. Die Ähnlichkeit geht so weit, daß man aus einer sicheren Blockchiffre eine Hashfunktion generieren kann [16, S. 446] und umgekehrt[17].

2.5.1.1 MD2, MD4 und MD5 (Message Digest)

Die Abkürzung MD steht für *message digest*. Diese drei Verfahren wurden von Ronald Rivest der Öffentlichkeit zur Verfügung gestellt. Sie sind in [16, Kapitel 18], [36], [37] und [38] beschrieben.

Allen drei Verfahren ist gemeinsam, daß sie eine Nachricht auf einen Hashwert der Länge 128 Bit abbilden. Ebenso iterieren alle drei Verfahren eine Rundenfunktion, ähnlich wie die Blockchiffren DES und IDEA.

Die Sicherheit von MD2 bestätigt sich gegenüber allen bekannten Analyseverfahren. MD4 wurde entwickelt, weil die Berechnung von MD2 für viele Anwendungen zu zeitaufwendig ist. Nach einigen Analyseversuchen, wurden theoretische Schwächen bei MD4 gefunden, die aller Wahrscheinlichkeit nach nicht von praktischer Bedeutung sind. Dennoch veröffentlichte Ronald Rivest die verbesserte Version MD5.

Die Hashfunktionen MD2, MD4 und MD5 werden von fast allen kryptographischen Protokollen verwendet. Oft steht kein anderes Verfahren als Alternative zur Wahl.

2.5.1.2 SHA (Secure Hash Algorithm)

Der *secure hash algorithm* [39] ist die „offizielle“ amerikanische Hashfunktion. Sie wird unter anderem zur Realisierung des *digital signature standard* [40] eingesetzt.

SHA bildet eine Nachricht auf einen 160 Bit langen Hashwert ab. Die Struktur von SHA scheint ebenso wie die von MD5 als Verbesserung des MD4 entstanden zu sein, sodaß man auch manchmal die Bezeichnung *stolen hash algorithm* findet [18, S. 139]. SHA ist in [16, S. 442] und [39] beschrieben.

2.5.2 Digitale Unterschriften

Um mit dem RSA-Verfahren Unterschriften zu erstellen, wird die in Abschnitt 2.4.4.2 dargestellte Vorgehensweise nur geringfügig geändert. Bezeichne wieder M eine Nachricht, $D()$ die Anwendung des RSA-Verfahrens mit dem privaten Schlüssel und h die sichere Hashfunktion. Dann berechnet man die digitale Unterschrift s durch

$$s = D(h(M))$$

. Der Empfänger einer Nachricht berechnet den Hashwert $h(M)$ der Nachricht. Dann wendet er das RSA-Verfahren mit dem öffentlichen Schlüssel des Unterzeichnenden $E()$ an und erhält:

$$E(s) = E(D(h(M))) = h(M) \quad \checkmark$$

Die einzige ernsthafte Konkurrenz zur Erzeugung von Unterschriften mit dem RSA-Verfahren ist der *digital signature algorithm* (DSA, [40], [16, Kapitel 20]), allerdings ist der DSA heftiger Kritik ausgesetzt [18, S. 138]. In den weiter unten behandelten Vorschlägen spielt der DSA keine Rolle.

[17] Man denke an die Betriebsart OFB aus dem Abschnitt 2.3.3.3.

2.5.3 Zertifikate

Nachdem Unterschriften schnell, kompakt und sicher realisiert werde können, verbleibt noch das Problem der „gefälschten“ öffentlichen Schlüssel. Um ein Key-Management zu ermöglichen, das die Authentizität der verteilten öffentlichen Schlüssel sicherstellt, werden sogenannte Zertifikate eingeführt. Ein Zertifikat stellt eine Verbindung her zwischen einer Identität und einem öffentlichen Schlüssel. Diese Verbindung wird durch die Unterschrift des Herausgebers (der *certificate authority*) bestätigt. Ein Zertifikat hat nach X.509 folgenden Aufbau[18]:

Version	Derzeit eine feste Null.
Seriennummer	Jeder Herausgeber numeriert seine Zertifikate beginnend mit 1.
Algorithmus-Kennung	Welchen Algorithmus benutzt der Herausgeber für seine Unterschrift.
Herausgeber	(*issuer*) Die Certificate Authority.
Gültigkeitszeitraum	Beginn und Ende der Gültigkeit.
Name	(*subject*) Für wessen Identität wurde das Zertifikat ausgestellt.
Öffentlicher Schlüssel	Algorithmus-Kennung, Parameter und der öffentliche Schlüssel.
Unterschrift	Unterschrift des Herausgebers mit Algorithmus und Parameter aus dem dritten Feld.

Empfängt man eine Nachricht, die mit einer digitalen Unterschrift versehen ist, und das zur Unterschrift passende Zertifikat, so kann man die Unterschrift verifizieren. Allerdings muß man danach die Unterschrift unter dem Zertifikat verifizieren. Dies führt direkt zu einer Zertifizierungshierarchie, wie in Abbildung 2.3 auf Seite 45 dargestellt.

2.5.3.1 Zertifizierungshierarchien

Um die Überprüfung der Unterschriften zu einem Ende kommen zu lassen, wählt jeder Teilnehmer mindestens eine *root certificate authority* oder *Root-CA* deren Unterschrift er akzeptiert. Dazu ist es notwendig, daß er den öffentlichen Schlüssel der Root-CA über einen sicheren Kanal empfangen hat, wie zum Beispiel eine Tageszeitung oder eine amtliche Veröffentlichung.

In Abbildung 2.3 sind Alice und Bob unter einer gemeinsamen *root certificate authority* oder *Root-CA* dargestellt. In diesem einfachen Fall können Alice und Bob ihre Zertifikate gegenseitig leicht prüfen, da sie beide bei der Überprüfung den Kanten in der Zertifizierungshierarchie aufwärts folgen und zu einer akzeptierten Unterschrift gelangen.

Wie sieht die Überprüfung von Unterschriften aus, falls Bob keinen Pfad von Alice' Zertifikat zu einer seiner eigenen Root-CAs findet? Die Überprüfung muß fehlschlagen. Die – im Wortsinn – radikalste Lösung wird z.B. in [60] vertreten und ist auch in [61] vorgesehen: man führt eine weltweite Zertifizierungshierarchie mit einer gemeinsamen Wurzel für alle ein, sodaß die Verifikation einer Unterschrift garantiert erfolgreich terminiert, falls die Unterschrift gültig ist.

Zur Vereinfachung werden Kreuzzertifizierungen zwischen Unterbäumen der Hierarchien vorgeschlagen, um die Zahl der bei der Verifikation verfolgten Kanten zu beschränken. Beispielsweise könnten die (deutschen) Universitäten als CA für ihre Angehörigen, Mitarbeiter und Studenten auftreten und sich untereinander kreuz-zertifizieren, um den Umweg über (inter)nationale CAs zu umgehen.

Die Art und Weise, wie auf eine Zertifizierungshierarchie zugegriffen wird, ob sie lokal bei jedem Anwender gespeichert wird oder, ob die Verwaltung und das Auffinden von Zertifikaten überhaupt Teil des Protokolls ist, unterscheidet die verschiedenen Vorschläge.

[18] [16, S. 574], [42, Kapitel 2], [51], [58] ...

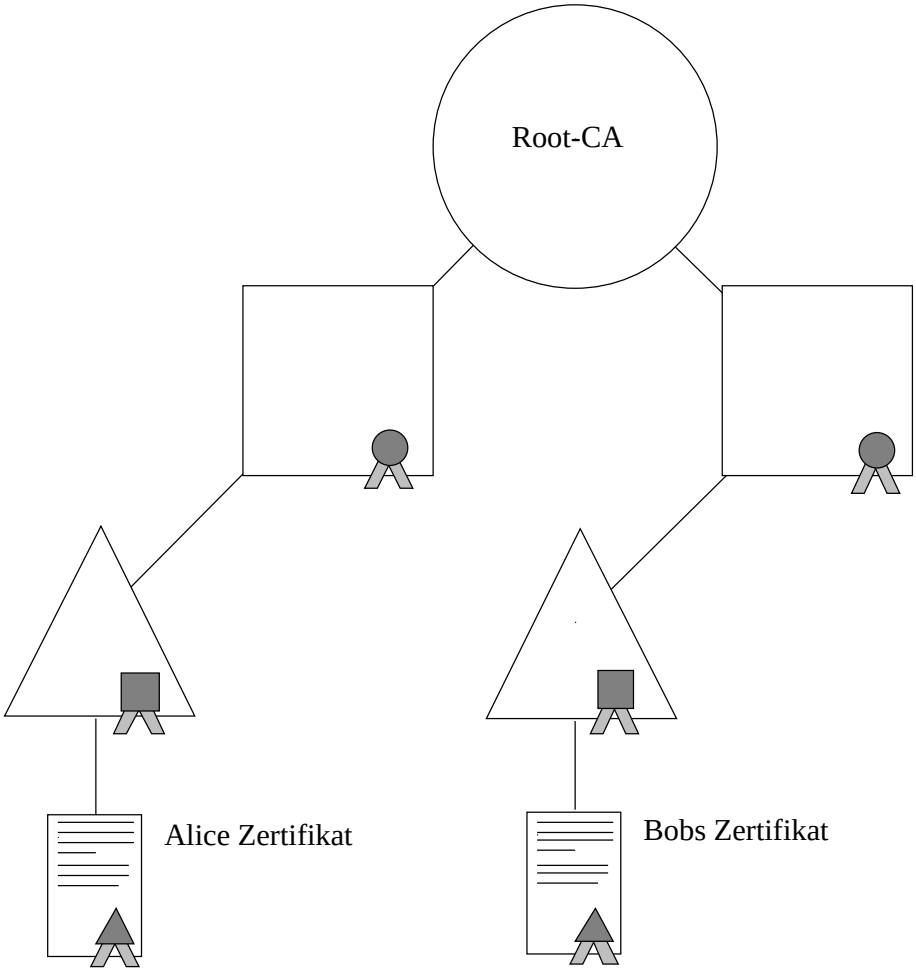

Abbildung 2.3: Alice und Bob unter einer gemeinsamen Root-CA.

Das Management der Zertifizierungshierarchie verursacht noch weitere Komplexität: Schlüssel und damit auch Zertifikate werden periodisch geändert oder werden wegen vermutlich „gestohlener“ (*comprised*[19]) privater Schlüssel ungültig. Diese Informationen müssen an die Anwender weitergegeben werden.

In diesem Zusammenhang stellt sich auch das Problem der Verifikation bzw. Gültigkeitsdauer „alter“ Unterschriften. Ist eine Unterschrift älter als eine gängige Gültigkeitsperiode von Zertifikaten, so steht bei einer fälligen Verifikation die dazu notwendige Zertifizierungshierarchie gar nicht mehr zur Verfügung. Also muß zur Verifikation alter Unterschriften die gesamte Historie der Zertifizierungshierarchie verwaltet werden. Um bei einem Wechsel des Schlüssels einer CA einen sanften Übergang zu ermöglichen, sollten die neuen Schlüssel mit den alten zertifiziert werden – falls die alten nicht kompromittiert wurden.

Wurde ein Schlüssel kompromittiert, so verfallen mit dem Rückruf des Schlüssels alle Unterschriften, die mit diesem Schlüssel generiert wurden. Dies würde es also einem Teilnehmer ermöglichen, rückwirkend von allen Vereinbarungen zurückzutreten, die er mit einem bestimmten Schlüssel unterschrieben hat, alleine durch die Behauptung, sein Schlüssel sei kompromittiert und müsse zurückgerufen werden.

Längerfristig gültige Vereinbarungen sollten also zusätzlich von einem „Notar“ unterschrieben werden, der für die Echtheit der Unterschrift und der Identität des Unterzeichnen-

[19] Ein privater Schlüssel, der vermutlich in die Hände Unbefugter gelangt ist, heiße „kompromittiert“.

den bürgt. Öffentliche Schlüssel von Notaren wären also auch mit der Zertifizierungshierarchie zu verwalten.

Damit sollte prinzipiell klar sein, welche Problem mit der Verwaltung einer Zertifizierungshierarchie einhergehen. Ebenso wurden skizziert, wie diese Probleme gelöst werden können. Es wird jedoch ersichtlich, daß bei der Implementierung ein großer Spielraum bleibt und viele Detailentscheidungen zu fällen sind. Aus diesem Grund verwundert es nicht, daß einige Protokollvorschläge die Verwaltung von Zertifikaten nicht abdecken.

2.6 Kryptographische Protokolle

In diesem Abschnitt wird zunächst ein Prototyp eines kryptographischen Protokolls vorgestellt. Dieser Prototyp ist unabhängig von der technischen Realisierung der Verschlüsselung. Bei der Auswahl eines real existierenden Protokoll spielt es jedoch eine Rolle, an welcher Position zwischen anderen Übertragungsprotokollen ein kryptographisches Protokoll positioniert wird. Man könnte sogar behaupten, daß die Positionierung im sogenannten Protokoll-Stack die vorgeschlagenen Protokolle besser charakterisiert als ihre jeweiligen Abweichungen vom Prototyp.

Im Anschluß an diese beiden Abschnitte werden – endlich – einige real existierende Protokolle bzw. Protokollvorschläge beschrieben. Aufgrund der wenigen Unterscheidungsmerkmale kann die Beschreibung nun äußerst kompakt gestaltet werden.

2.6.1 Der Prototyp

Um die Vorteile von öffentlichen Schlüsseln und die hohe Geschwindigkeit symmetrischer Verfahren nutzen zu können, kombiniert man beide Verfahrensklassen. Man wählt ein symmetrisches Verfahren und würfelt einen Schlüssel hierfür aus. Die Nachricht wird nun mit dem symmetrischen Verfahren verschlüsselt. Diesen Schlüssel chiffriert man nun mit dem RSA-Verfahren und stellt ihn der Nachricht voran.

Dadurch wird es auch möglich, eine Nachricht mit relativ geringem Aufwand an eine Gruppe von Empfängern zu richten (die aus praktischen Gründen auch den Absender enthalten soll/kann/darf): man stellt der Nachricht einfach eine Liste voran, die den Schlüssel für das symmetrische Verfahren jeweils für jeden Adressaten (mit dessen öffentlichem Schlüssel chiffriert) enthält (*digital envelope*).

Bei *on line*-Verbindungen dient das asymmetrische Verfahren nur zur Verhandlung des symmetrischen Verfahrens und des verwendeten Schlüssels (*session key*). Alle Nutzdaten werden nur noch mit Hilfe des symmetrischen Verfahrens chiffriert.

Damit sieht der Prototyp wiefolgt aus:

Initialisierung:

1. `Beschaffe mindestens ein RSA-Schlüsselpaar und ein dazugehöriges Zertifikat.`

2. `Wähle mindestens eine Certificate Authority als eigene Root-CA, üblicherweise eine, die auch obiges Zertifikat unterschrieben hat.`

3. `Lege eine Datenbank der Zertifizierungshierarchie an oder beschaffe Dir Zugang zu einer solchen.`

Für jede Kommunikation:

1. `Lasse Dir von Deinem Kommunikationspartner ein Zertifikat zeigen und verfolge die Kette der Unterschriften bis zu einer (Root-)CA Deines Vertrauens. Dabei darf der Kommunikationspartner bei der Beschaffung der benötigten Zertifikate auf dem Pfad zur Root-CA helfen.`

2. `Verfahre genauso in umgekehrter Richtung.`

3. `Verhandele nun bereits in asymmetrisch verschlüsselter Form ein symmetrisches Verfahren, einen session key und eine Hashfunktion.`

```
4. Kommuniziere von nun an unter Verwendung des symmetrischen
   Verfahrens.
```

Dieser Prototyp benötigt abgesehen von den Zertifikaten keine digitalen Unterschriften. Die Authentisierung des Kommunikationspartners geschieht implizit dadurch, daß dieser nach der Verhandlung des symmetrischen *session key* plausible Antworten generieren kann. Diese Tatsache beweist, daß er den passenden privaten Schlüssel zu seinem Zertifikat haben muß. Wie bei der Beschreibung des RSA-Verfahrens erwähnt, ist die nicht interaktive Erzeugung von digitalen Unterschriften im Protokollablauf nicht wünschenswert.

Die im Abschnitt 2.6.3 beschriebenen Protokolle unterscheiden sich hauptsächlich in der Auswahl der zur Verfügung stehenden Algorithmen für die symmetrische Verschlüsselung und die Berechnung von Hashwerten. Große Unterschiede liegen auch in der Berücksichtigung der Verwaltung der Zertifizierungshierarchie. Auf einen weiteren Unterschied, die Positionierung des Protokolls im Protokollstack, geht der nächste Abschnitt ein.

2.6.2 Ansatzmöglichkeiten für Protokolle

Die Frage, auf welcher Protokollebene man verschlüsseln sollte, wird weniger von Autoren gestellt, die Kryptographie zu ihrem Hauptgegenstand machen. Sie tritt eher dort auf, wo das Zusammenspiel verschiedener Protokolle untersucht wird, z.B. in [14, S. 370] oder [19, S. 226].

Jede Netzwerkkommunikation findet über mehrere Protokolle gleichzeitig statt, um das Anwendungsprogramm von solchen Details wie etwa den Signalspannungen auf einer seriellen Leitung fernzuhalten. Man kann sich einen sogenannten Protokoll-Stack vorstellen, der unten auf der physikalischen Ebene beginnt und bis hinauf bis zu Anwendungsprotokollen wie etwa HTTP reicht.

Eine andere Sichtweise betont, daß die Daten eines höher gelegenen Protokolls in die Strukturen eines darunterliegenden verpackt werden wie ein Brief in einen Umschlag. Genauso sinnvoll ist es, von virtuellen Verbindungen zwischen gleich hohen Protokollschichten auf zwei Rechnern zu sprechen. In der Abbildung 2.4 (angelehnt an [3, Abschnitt 1.1]) ist ein Protokoll-Stack dargestellt, wie er typischerweise bei der Benutzung des BSCW-Systems auf einem per Ethernet vernetzten Rechner aussieht.

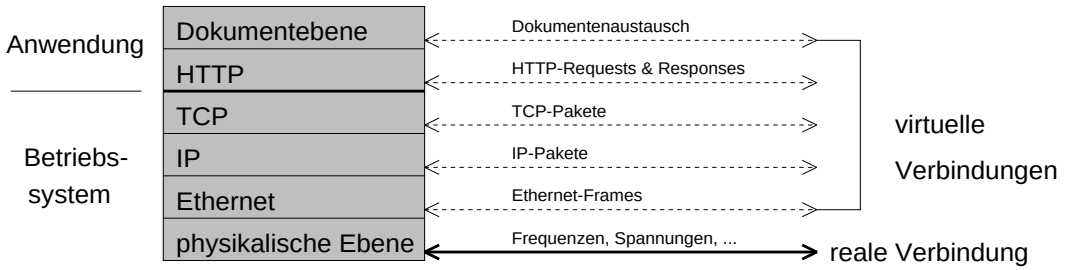

Abbildung 2.4: Ein typischer Protokollstack einer BSCW-Verbindung im Internet.

Die Leistungsfähigkeit dieser Architektur liegt in der Möglichkeit, einzelne Schichten unter Erhaltung der Schnittstellen durch andere Implementierungen zu ersetzen. Ein gängiges Beispiel wäre hier, die Ersetzung der Ethernet-Verbindung durch eine SLIP-Verbindung (Serial Line IP). Dazu sind keine Änderungen auf höher gelegenen Ebenen nötig. Das vielzitierte Referenzmodell für ein solches *layering* ist das siebenschichtige OSI-Modell (*open systems interconnection*). Es wird z.B. in [62, Abschnitt 5.6] skizziert. Die im Internet verwendeten Protokolle sind jedoch älter als das OSI-Modell.

Für die Positionierung von Kryptoprotokollen zur Absicherung des BSCW-Systems ist die nachfolgende Einteilung sinnvoll:

Ebene	Art der übertragenen Daten
Dokumentebene	Dokumente, Dateien und HTML-Seiten
Ebene des Anwendungsprotokolls	HTTP-Requests, HTTP-Responses
Ebene der Netzwerkprotokolle	TCP-Pakete, IP-Pakete
Physikalische Ebene	Ethernet-Frames, Signalpegel

Wie beeinflußt die Entscheidung für eine bestimmte Ebene der Verschlüsselung die Sicherheit des Gesamtsystems?

Auf einer bestimmten Ebene im Protokollstack stehen nur bestimmte Informationen zur Verfügung. Auf der Ebene der Netzwerkprotokolle existiert z.B. weder das Konzept des Prozesses noch das des Benutzers. Setzt man also kryptographische Verfahren auf der Ebene der Netzwerkprotokolle ein, ist eine Authentisierung von Personen unmöglich, es sei denn, man stellt auf dieser Ebene zusätzliche Informationen zur Verfügung, zerstört also das strenge Schichtenmodell.

Verschlüsselt man auf einer bestimmten Ebene, so bleiben alle zusätzlich der Nachricht beigefügten Protokollinformationen der darunterliegenden Protokolle als Verkehrsinformation sichtbar. Damit ändert sich auf jeder Ebene die Perspektive, aus der Nutzdaten und Verkehrsinformationen unterschieden werden: je tiefer im Protokoll-Stack man die Verschlüsselung ansetzen kann, desto weniger Verkehrsinformationen bleiben ungeschützt.

Bei der Absicherung des BSCW-Systems ist dem Trend, auf der tiefst möglichen Ebene zu verschlüsseln, jedoch eine Grenze gesetzt: die Protokollinformationen des IP müssen unverschlüsselt bleiben, während das Datenfeld in IP-Paketen verschlüsselt sein darf. Dies ist notwendig, da die IP-Pakete im Internet mit Hilfe der IP-Header geroutet werden. Fehlt die Header-Information, so kann der adressierte Rechner nicht erreicht werden.

Verschlüsselung der IP-Pakete ist also im Internet prinzipiell nur dann möglich, wenn diese Pakete wieder über ein höherliegedes Protokoll versandt werden oder, wenn nur eine direkte Verbindung zwischen zwei Teilnetzen abgesichert werden soll (*link level encryption*, [14, S. 370], [19, S. 226]).

Das entgegengesetzte Vorgehen wäre die Verschlüsselung auf Dokumentebene. Im BSCW-Kontext wäre dies sehr einfach dadurch zu realisieren, daß sich z.B. alle Benutzer (eines Arbeitsbereiches) auf ein Verfahren einigen und nur noch verschlüsselte Dokumente mit dem BSCW-System austauschen.

Durch Veschlüsselung auf der Dokumentebene hat man ohne zusätzlichen Aufwand die Möglichkeit, digitale Unterschriften an den Dokumenten anzubringen, die nicht auf irgendeiner Protokollebene abgestreift werden, sondern dem endgültigen Empfänger zur Verfügung stehen. Für das Erreichen der „Nichtabstreitbarkeit“ von Nachrichten ist das Verschlüsseln auf Dokumentebene also ideal. Allerdings bleibt zur Authentisierung auch keine andere Möglichkeit als die explizite Erzeugung einer digitalen Unterschrift.

Diese Lösung ist zwar alles andere als benutzerfreundlich, aber äußerst flexibel, da man so verschlüsselte Dokumente auch über elektronische Post oder *ftp*-Transfers versenden könnte.

Zwischen diesen beiden Extremen liegt die Verschlüsselung auf der Ebene des HTTP oder TCP. So könnte man den *body* jeder HTTP-Nachricht verschlüsseln, bevor die Nachrichten an das TCP weitergereicht werden. Genauso kann man die gesamte HTTP-Nachricht inklusive HTTP-Header verschlüsseln, bevor sie in TCP-Pakete verpackt wird.

Die Realisierung einer solchen Lösung macht scheinbar einen Eingriff in die Anwendungsprogramme nötig, der zumindest dann problematisch wird, wenn eine Vielzahl von Plattformen auf der Client-Seite unterstützt werden soll.

2.6.3 Reale Protokolle

Die Entwicklung von kryptographischen Protokollen und Anwendungen erfordert einen enormen Aufwand. Darüber täuschen oft die äußerst knappen Dokumente hinweg, die den jeweiligen Vorschlag beschreiben. Der Umfang der Dokumentation hängt meist davon ab, ob dem Vorschlag eine allgemeine Einführung in die Kryptographie vorangestellt wird und, ob neben einem abstrakten Vorschlag auch noch eine Referenzimplementierung beschrieben wird.

Die hier gegebenen Kurzbeschreibungen können dem Entwicklungsaufwand der einzelnen Vorschläge natürlich kaum gerecht werden. Hier wird nur die Zielsetzung eines Vorschlages skizziert und seine grundsätzlichen Merkmale werden beschrieben.

Diese Kurzbeschreibungen können und wollen auf keinen Fall die intensive Beschäftigung mit den einzelnen Vorschlägen ersetzen, wenn ein Sicherheitsmechanismus für ein anderes System als BSCW entwickelt werden soll. Ebenso kann hier unmöglich ein Überblick über alle Vorschläge gegeben werden: wahrscheinlich übertreibt man nur wenig, wenn man behauptet, daß für jedes existierende Übertragungsprotokoll eine kryptographische Erweiterung erwogen wird.

2.6.3.1 *Pgp* (Pretty Good Privacy)

Pgp ist ein Programm zum Verschlüsseln und Signieren von Dokumenten, arbeitet also auf der Dokumentebene.
Pgp wurde mit dem Ziel entwickelt, jeder Privatperson die Möglichkeit zur sicheren Verschlüsselung zu geben, ohne dabei durch staatliche Eingriffe oder Normierungsversuche eingeschränkt zu werden.
Die verwendeten Verfahren sind MD5, IDEA und RSA.
Pgp ist als Protokoll zu betrachten, da abgesehen von den verwendeten Dateiformaten für verschlüsselte Dokumente auch noch Mechanismen zum Austausch von gültigen Schlüsseln, kompromittierten Schlüsseln und Zertifikaten implementiert sind.
Pgp sieht im Gegensatz zu den meisten anderen Vorschlägen keine Zertifizierungshierarchie vor, sondern ein sogenanntes *web of trust* [18, Kapitel 12]: hier wird ein öffentlicher Schlüssel anerkannt, wenn *genügend* viele vertrauenswürdige Personen seine Authentizität bezeugen. Die Entscheidung, was *genügend* bedeutet und wer vertrauenswürdig ist, obliegt jedem Einzelnen.

2.6.3.2 SecuDE (Security Development Environment)

SecuDE ist eine Sammlung von Programmen, Routinen und Bibliotheken zur Verschlüsselung auf Dokumentebene.
Der Schwerpunkt liegt auf dem *key management*: auf Basis des X.500-Standards [63] werden Verzeichnisse für Namen, Adressen, Schlüssel, Zertifikate und andere Daten realisiert.
Der Zugriff auf Verzeichnisse selbst kann wieder durch kryptographische Methoden abgesichert werden.
Die verwendeten Verfahren sind MD5, DES und RSA.
Die Leistungen von SecuDE liegen weniger im Bereich der Kryptographie als in der Integration von Kryptographie und den Verzeichnisdiensten nach X.500. Eine Besonderheit von SecuDE liegt in der Realisierung von digitalen Unterschriften: sie enthalten einen kompletten Pfad vom Zertifikat des Unterzeichnenden bis zu einer Root-CA. Dies beschleunigt zwar die Verifikation von Unterschriften, setzt aber implizit voraus, daß der „Adressat der Unterschrift“ die gleiche Root-CA anerkennt wie der Unterzeichnende.
Eine weitere Besonderheit von SecuDE ist die Möglichkeit, Chipkarten zur Verwahrung des privaten Schlüssels zu verwenden.
Leider steht SecuDE in der aktuellen Version 4.4 nicht mehr frei zur Verfügung, da kommerziell vermarktete Produkte Teil des Systems sind.

2.6.3.3 PEM (Privacy Enhancement for Internet Electronic Mail)

PEM [57, 58, 59, 60] ist ein Standardisierungsvorschlag für die Verschlüsselung von Nachrichten, die per elektronischer Post versandt werden. PEM arbeitet entweder eingebettet in das *simple mail transfer protocol* (SMTP, [64]) oder auf Dokumentebene.
Die verwendeten Verfahren sind MD2, MD5, DES und RSA.
Das Sicherheitsmodell von PEM entspricht dem oben skizzierten Prototyp, wobei noch Vorschläge zur Gliederung der Zertifizierungshierarchie gemacht werden. Der Schwerpunkt bei PEM liegt in der Einbettung verschlüsselter Nachrichten in das SMTP.
Der Austausch von Zertifikaten und Listen kompromittierter Schlüssel ist Teil der PEM, die Verwaltung von Zertifizierungshierarchien jedoch nicht.

2.6.3.4 PKCS (Public Key Cryptography Standards)

Die PKCS ([45] bis [56]) beschreiben kein bestimmtes Protokoll, sondern geben eine allgemeine Einführung in die Kryptographie. Es werden symmetrische und asymmetrische Verfahren vorgestellt sowie Zertifizierungshierarchien und Probleme des *key management*.
Die in den PKCS vorgeschlagenen Formate für verschlüsselte Nachrichten, Schlüssel und Zertifikate sind weitgehend kompatibel zu PEM. Über die in PEM verwendeten Verfahren hinaus werden noch MD4 und Diffie-Hellman genannt.
Die PKCS wurden von der Firma *RSA Data Security, Inc.* veröffentlicht. Sie enthalten zwar Hinweise auf die symmetrische Stromchiffre RC4 und die symmetrische Blockchiffre RC2, aber die genaue Funktionsweise versuchte RSADSI geheimzuhalten.

2.6.3.5 S-HTTP (Secure Hypertext Transfer Protocol)

S-HTTP [65] ist eine Erweiterung des HTTP. Ziel ist die Integration der Techniken aus PEM bzw. PKCS in das HTTP. Authentisierung ist sowohl für den Client als auch für den Server möglich, kann aber entfallen.
Da das HTTP transaktionsorientiert ist und eine Verbindung immer nur aus dem Austausch von Request und Response besteht, liegt hier fast die gleiche Situation vor, wie bei einem *off line*-Protokoll wie PEM. Das Zertifikat bzw. der öffentliche Schlüssel des Servers müssen schon vor dem Senden des ersten Requests bekannt und akzeptiert sein und der Request muß alle Parameter für die Response festlegen. Von einer Verhandlung der Verbindungsparameter kann also keine Rede sein.
Das ohnehin schon wildwuchernde HTTP um weitere möglichst orthogonale Optionen zu erweitern, scheint fragwürdig.

2.6.3.6 SSL (Secure Sockets Layer)

SSL [31, 66] wurde mit dem Ziel entwickelt, spontane kommerzielle Transaktionen zwischen einem Kunden und einem Anbieter zu ermöglichen. Aus diesem Grund ist die Authentisierung des Clients optional, während die Authentisierung des Server verpflichtend ist. Das bedeutet ein Server kann von seinen Clients Authentisierung fordern.
Die verwendeten Verfahren sind MD5, DES, IDEA, RC2, RC4, Diffie-Hellman, Fortezza[20] und RSA.
In [66] wird besonders die Unabhängigkeit höherer Anwendungsprotokolle von SSL betont, da SSL fast die gleiche Schnittstelle bietet wie die Berkeley-Sockets [62, Kapitel 6].
SSL verschlüsselt nicht auf einer vorhandenen Ebene, sondern schafft eine neue Schicht, die zwischen Anwendungsprotokollen und den Berkeley-Sockets liegt.

[20] „Fortezza“ ist eine Reinkarnation des sogenannten Clipper- oder Capstone-Chips als PCMCIA-Karte. Hierzu finden sich wiederum Hinweise z.B. in [18, S. 127] und [16, Abschnitte 24.16 und 24.17]. Die genaue Funktionsweise von „Fortezza“ wird geheimgehalten.

Es ist prinzipiell denkbar, SSL in den Betriebssystemkern zu integrieren und den Anwendungen weiter die gleichen Betriebssystemaufrufe zur Verfügung zu stellen. Die Konfiguration der SSL-Verbindungen muß dann außerhalb der Anwendungen vorgenommen werden.

2.6.3.7 SET (Secure Electronic Transactions)

SET [67, 68] verschlüsselt nicht auf einer vorhandenen Ebene im Protokoll-Stack, sondern umfasst ein eigenes Kommunikations- bzw. Transaktionsmodell. Hier geht es um das Schließen von Verträgen über elektronische Medien sowie um das Transferieren von Zahlungsanweisungen. Dies entspricht am ehesten noch der Verschlüsselung auf Dokumentebene.
Das Transaktionsmodell sieht neben Kunden, Anbietern und CAs noch sogenannte *payment gateways* vor, die die Verbindung zu Banken und Kreditkartengesellschaften herstellen. Da es hier um kommerzielle Transaktionen geht, ist klar, daß sich jede Partei den anderen gegenüber authentisieren muß.
Die verwendeten Verfahren sind SHA, DES und RSA. SET enthält keinen Vorschlag zur Absicherung beliebiger Netzwerkverbindungen.

2.6.3.8 Zusammenfassung

Entfernt man die kommerziell zu lizensierende Software zur Verwaltung von Verzeichnissen nach X.500, bleibt nur noch ein einfaches System zur Verschlüsselung auf Dokumentebene zurück, das auf dem schon betagten DES beruht. SecuDE ist in der aktuellen Version 4.4 noch nicht für MS-DOS bzw. Windows verfügbar.

Die PKCS-Papiere der Firma RSADSI beschreiben im wesentlichen nicht eine bestimmte Implementierung, sondern sie beschreiben ganz allgemein Konzepte, die bei der Verwendung von Kryptographie zu verfolgen sind. Ist der Kunde durch die Papiere von der Kompetenz der Firma überzeugt, soll er nicht eigene Software erstellen, sondern Produkte von RSADSI inklusive Maulkorbvertrag (*non disclosure agreement*) erwerben. Ein Vorgehen, das bei der Geheimhaltung der Verfahren RC2 und RC4 offenbar nicht von Erfolg gekrönt war.

Der Vorschlag SET ist noch weit von der Realisierung entfernt. Die Spezifikationen sind auch noch nicht so weit ausgereift, daß auf ihrer Grundlage eine Implementierung möglich wäre. Es ist zu erwarten, daß der Vorschlag nach seiner vorläufigen Validierung hinter verschlossenen Türen der Kreditkartengesellschaften weiterentwickelt wird. In seiner derzeitigen Form sieht SET nur bestimmte (kommerzielle) Transaktionen vor, die auf „Dokumentebene“ verschlüsselt und unterzeichnet werden. Verschlüsselung auf der Ebene von Netzwerkverbindungen ist nicht vorgesehen.

Damit bleiben nach Durchsicht der verschiedenen Vorschläge vier Kandidaten in der engeren Auswahl: *pgp*, PEM, S-HTTP und SSL. Die vier akzeptablen Vorschläge liegen entweder in ausreichend präziser Form vor, um eine Implementierung zu ermöglichen, oder sind schon als Referenzimplementierung erhältlich. Unter diesen Vorschlägen ist S-HTTP von der Realisierung noch am weitesten entfernt.

Diese Vierergruppe repräsentiert sehr gut die Ansatzmöglichkeiten im Protokoll-Stack von *pgp* und PEM auf der Dokumentebene über S-HTTP auf der Ebene des Anwendungsprotokolls bishin zu SSL direkt zwischen Anwendungsprotokoll und Netzwerkprotokoll.

In der folgenden Tabelle sind die wesentlichen Merkmale der Protokollvorschläge tabellarisch zusammengefasst:

	PGP	SecuDE	PEM	PKCS	SHTTP	SSL	SET
Hash-Vefahren							
MD2	–	–	✓	✓	✓	✓	–
MD4	–	–	–	✓	✓	–	–
MD5	✓	✓	✓	✓	✓	✓	–
SHA	–	–	–	✓	✓	✓	✓
Symmetrisch							
DES	–	✓	✓	✓	✓	✓	✓
IDEA	✓	–	–	–	✓	✓	–
RC2	–	–	–	✓	✓	(✓)	–
RC4	–	–	–	✓	✓	(✓)	–
Asymmetrisch							
RSA	✓	✓	✓	✓	✓	✓	✓
Fortezza	–	–	–	–	–	✓	–
Diffie-Hellman	–	–	–	✓	✓	✓	–
Sicherheitsmodell							
X509-Zertifikate	(✓)	✓	✓	✓++	✓	✓	✓++
Zertifikatverwaltung	✓	✓	(✓)	(✓)	(✓)	(✓)	(✓)

Kapitel 3

Auswahl, Entwurf und Implementierung

Zu Beginn der Arbeit stand fest, daß die exemplarische Implementierung des noch zu entwerfenden Sicherheitsmechanismus auf der Clientseite sowohl Unix[1] als auch Windows unterstützen sollte. Auf der Serverseite sollte Unix verwendet werden.

Weniger eindeutig war die Frage zu beantworten, ob das bereits existierende BSCW-System Version 1.0 abgesichert werden sollte oder, ob im Rahmen der Diplomarbeit ein eigener, „kleiner“ Dokumentenserver programmiert und abgesichert werden sollte.

Der im folgenden Abschnitt beschriebene, erste Entwurf ging davon aus, daß die Integration eines Sicherheitsmechanismus' in eigenen und deshalb gut bekannten Quellcode einfacher ist als in den umfangreichen Quellcode des BSCW-Systems, das zudem noch in der objektorientierten Interpretersprache Python [69] geschrieben ist.

Der erste Entwurf wurde letztendlich zugunsten eines anderen verworfen. Deswegen werden im ersten Abschnitt einige Themen nur kurz gestreift, die wesentlich genauer zu beschreiben wären, wenn sie zu einer funktionierenden Lösung beigetragen hätten.

Der zweite Entwurf kommt völlig ohne einen Eingriff in den abzusichernden Dokumentenserver aus, sodaß das BSCW-System nicht angetastet werden muß. Er wird im zweiten Abschnitt dieses Kapitels beschrieben.

Im dritten Abschnitt des Kapitels werden die Performanzprobleme des zweiten Entwurfs dargestellt, die hauptsächlich aus der grundsätzlichen Unverträglichkeit zwischen dem transaktionsorientierten HTTP und dem zeichenstromorientierten TCP liegen. Eine mögliche Lösung wird hier vorgestellt.

3.1 Der erste Entwurf: Verschlüsselung auf Dokumentebene

Der erste Entwurf sieht einen eigenständigen Dokumentenserver vor, der über das *common gateway interface* (CGI) von einem beliebigen HTTP-Server angesprochen wird. Im Gegensatz dazu benötigt das BSCW-System in der Version 1.0 einen speziellen, modifizierten HTTP-Server.

Diese Abhängigkeit ist nicht der einzige Grund für die Überlegung, einen eigenen Dokumentenserver zu bauen. Grundsätzlich scheint es vernünftig, im Rahmen der Diplomarbeit ein kleines System mit vollständiger Kontrolle über den Quellcode zu benutzen, als sich in ein großes, fremdkontrolliertes (Versionssprünge) System einzuarbeiten.

[1] Die exemplarische Implementierung verwendet Linux, aber keine speziellen Eigenschaften dieses Betriebssystems, sodaß im Folgenden nur noch von Unix gesprochen wird.

Insbesondere sollte aus Effizienzgründen möglichst das gesamte System in einer kompilierten Sprache realisiert werden, im Gegensatz zu BSCW, das in der Interpretersprache Python realisiert ist.

3.1.1 Integration der Ver- und Entschlüsselung

Die Einbindung von Ver- und Entschlüsselung auf der Serverseite erscheint als unproblematisch, da hier zunächst ausschließlich Unix zum Einsatz kommen soll. Die Einbindung auf der Clientseite soll durch Standardmechanismen geleistet werden, die von allen Web-Browser unabhängig vom jeweiligen Betriebssystem und der jeweiligen Fensteroberfläche zur Verfügung gestellt werden.

Die Abbildung 3.1 zeigt den Datenfluß für die Übertragung von Dokumenten. Beim Transfer eines Dokuments vom Dokumentenserver hin zum Benutzer (*download*) soll das Dokument für den Benutzer (mit seinem öffentlichen Schlüssel) verschlüsselt werden und mit einem speziellen Dokumenttyp (*MIME type*) an den Browser gesandt werden. Dieser startet eine für diesen Dokumenttyp konfigurierte Hilfsanwendung (*helper*, *helper application* oder *external viewer*). Diese Hilfsanwendung entschlüsselt das Dokument und speichert es im lokalen Dateisystem.

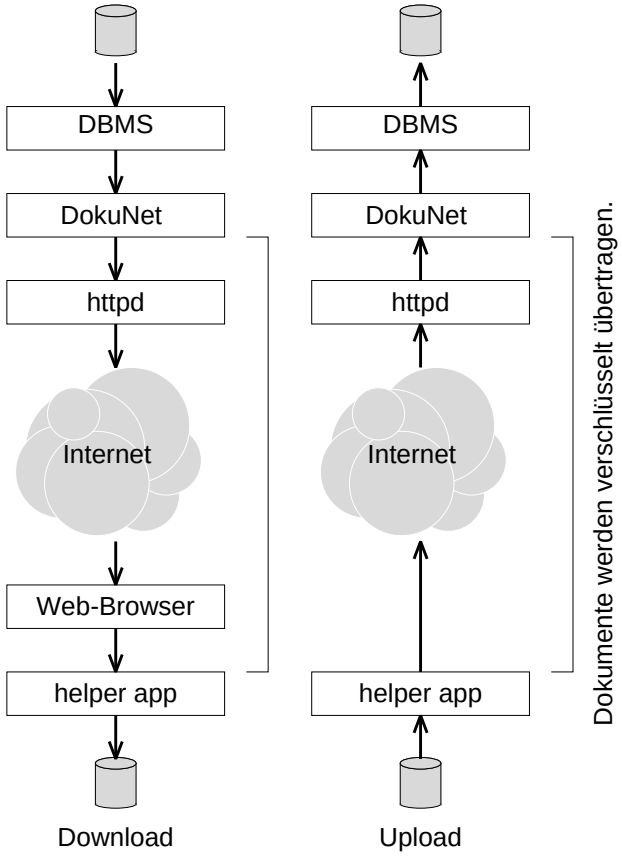

Abbildung 3.1: *Upload* und *Download*. Beim *Upload* spielt der Web-Browser keine Rolle. Ver- und Entschlüsselung findet in der Hilfsanwendung und im Dokumentenserver *DokuNet* statt.

Für den Transfer eines Dokuments in umgekehrter Richtung zum Server (*upload*) wird dieses durch eine Hilfsanwendung mit dem öffentlichen Schlüssel des Servers verschlüsselt und mit der HTTP-Methode POST oder PUT direkt ohne Umweg über den Web-Browser an den HTTP-Server geschickt.

3.1.2 Hilfsanwendungen, *plug in's* und Java

Die Lösung, Ver- und Entschlüsselung durch Hilfsanwendungen zu realisieren, ist nicht sehr gut in den Web-Browser integriert, sodaß hier nach Verbesserungsmöglichkeiten gesucht wurde. Zunächst scheint es, als Java eine Möglichkeit für die Einbettung der Hilfsanwendungen direkt in einen javafähigen Web-Browser.

Zu Beginn der Arbeit war der Browser *HotJava* der Firma Sun Microsystems der einzige stabil laufende Web-Browser, der Java untertützte und dieser lief nur unter Solaris auf Sparc-Rechnern. Das einzige *java development kit* (JDK) für Intel-Rechner lief nur unter Windows 95 und nicht unter Linux. Nachdem Windows 95 und das JDK auf einem Rechner installiert waren, stellte sich heraus, daß schon die mitgelieferten Demoprogramme nicht stabil liefen, ganz zu schweigen von ihrer Integration in einen Web-Browser.

Dem an Java Interessierten Leser sei der in der Zeitschrift *iX*, Ausgabe Mai 1996 beginnende Einführung in Java und empfohlen [70]. Hier finden sich auch kritische Literaturhinweise zum Thema.

Der eigentliche Grund dafür, daß Java als Implementierungssprache auf der Clientseite verworfen wurde, ist aber die prinzipielle Entwurfsentscheidung der Java-Entwickler, den Javaprogrammen, die in Web-Browser ausgeführt werden, Zugriffe auf das lokale Dateisystem zu verbieten. Diese Entscheidung ist aus sicherheitstechnischer Sicht plausibel, doch würde man sich Konfigurationsmöglichkeiten wünschen, die es erlauben, bestimmten Javaprogrammen Schreib- und Leserechte für Teile des lokalen Dateisystems zuzuweisen.

Sogenannte *plug in's* (Module, die als dynamische Bibliothek in den Web-Browser integriert werden.) wurden ebenso in Erwägung gezogen. *Plug in's* sind leider auf jeweils genau einen Browser für genau eine Plattform zugeschnitten. Selbst, wenn man sich aus leicht nachvollziehbaren Gründen auf den *Netscape Navigator* für Windows beschränkt, machen einem die häufigen Versionswechsel[2] hier zu schaffen.

Im Vergleich zu eigenständigen Hilfsanwendungen, die ebenso wie *plug in's* für jede Plattform angepaßt werden müssen, bringen *plug in's* wenige Vorteile bei höherem Arbeitsaufwand und wurden deshalb nicht weiter untersucht. Damit blieb es auf der Clientseite bei externen Hilfsanwendungen.

3.1.3 Verschlüsselungsverfahren

pgp war zu Beginn der Diplomarbeit die einzige[3] bekannte Implementierung des RSA-Verfahrens. In *pgp* wird IDEA als symmetrisches Verschlüsselungsverfahren und MD5 als Hashfunktion eingesetzt. Die verwendeten Verfahren geben also keinerlei Anlaß zu Kritik an der Sicherheit, höchstens die Tatsache, daß man nicht mehrere symmetrische Verfahren zur Auswahl hat (siehe Kapitel 2).

Der Quellcode von *pgp* ist nicht zuletzt durch den Verzicht auf eine graphische Benutzeroberfläche sehr portabel gehalten, sodaß *pgp* auf allen in Betracht gezogenen Plattformen verfügbar ist. Die Wahl fiel also mangels anderer bekannter Alternativen auf *pgp*.

Damit stand auch fest, daß auf Dokumentebene verschlüsselt wird, genauso wie die verwendeten Verfahren. *pgp* sollte zunächst aus den eigenen Programmen aufgerufen werden. Später sollte die Funktionalität von *pgp* in eine Bibliothek verpackt und direkt zu den eigenen Programmen gebunden werden. Es war auch klar, daß ein Benutzer ein Dokument, das er an den Server senden möchte, mit einer digitalen Unterschrift versehen muß. In umgekehrter Richtung muß der Server sich ebenfalls durch eine Unterschrift authentisieren. Wie diese Interaktion stattfinden soll und, ob sie bei jedem Transfer stattfinden soll, war allerdings unklar. Wie in Abschnitt 2.4.4.2 beschrieben, birgt die Automatisierung der Erstellung von digitalen Unterschriften unter Umständen ein Risiko.

[2] Während der Beschäftigung mit der Frage, ob *plug in's* sinnvoll genutzt werden können, kamen die *Netscape 2.0*-Betaversionen in zweiwöchigen Abständen heraus.

[3] Abgesehen von der Bibliothek RSAREF der Firma RSADSI, die in der amerikanischen Version von *pgp* verwendet wird.

3.1.4 Beginn der Implementierung

Bis hierher schien der Entwurf völlig klar zu sein und nachdem mit einem kleinen, shellbasierten Demonstrator unter Unix die prinzipielle Tauglichkeit des Entwurfs bewiesen war, wurde mit der Implementierung eines eigenen Dokumentenservers *DokuNet* begonnen.

DokuNet speichert sämtliche Dokumente und Daten über Benutzer und Arbeitsbereiche in einer relationalen Datenbank mit SQL-Schnittstelle[4]. Das ist ein wesentlicher Unterschied zum BSCW-System, das Daten teilweise im lokalen Dateisystem des Servers und teilweise mit Hilfe des Programms *gdbm* (*GNU database manager*) verwaltet.

Ein weiterer Unterschied zu BSCW ist, daß *DokuNet* aus einer einzigen Binärdatei besteht, während das BSCW-System aus einer großen Zahl von Python-Scripts besteht. Beide Merkmale steigern die Übertragungsleistung von *DokuNet* im Vergleich zum BSCW-System.

Die Oberfläche von *DokuNet* besteht nur aus in HTML-Seiten eingebetteten Formularen. Der Datenaustausch mit dem HTTP-Server findet über das *common gateway interface* (CGI) statt. In Abbildung 3.2 ist die Login-Seite von *DokuNet* zu sehen. Es wird keine *base authentication* verwendet, der Benutzername wird genauso wie andere Statusinformationen als Variable verwaltet.

Die Entscheidung, *base authentication* nicht (zusätzlich) zu benutzen, ist einfach durch das allgemeine Wissen um die Unsicherheit dieses Mechanismus' begründet: wenn man *base authentication* verwendet, ist man gezwungen, überzeugend darzulegen, daß die Sicherheit des Systems nicht von der Sicherheit der *base authentication* abhängt. Verwendet man diesen Mechanismus nicht, kommt man nicht in Beweisnot.

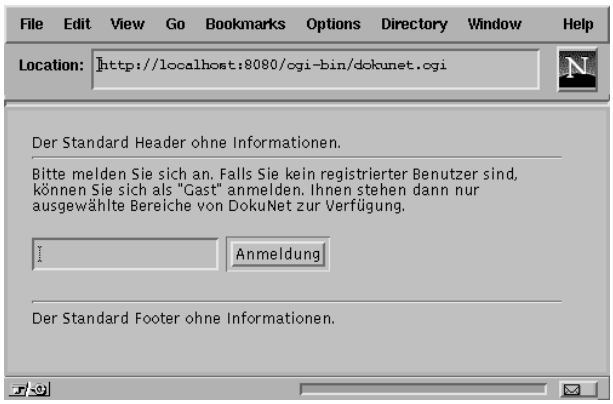

Abbildung 3.2: Das spartanische Login von *DokuNet*.

Die Abbildung 3.3 zeigt ein Verzeichnis der Dokumente im Arbeitsbereich „Geschuetzt“: einmal als HTML-Seite und einmal im HTML-Quelltext. Man erkennt, daß das gesamte Verzeichnis als Formular in HTML realisiert wurde. Die Statusinformationen Benutzername und Arbeitsbereich werden in verdeckten Variablen abgespeichert, die von Seite zu Seite weitergereicht werden.

Natürlich sind solche verdeckten Variablen nicht besser gegen unbefugte Einsichtnahme geschützt als die bei *base authentication* verwendeten Paßworte, aber sie gaukeln auch niemandem eine trügerische Sicherheit vor.

Noch bevor der Dokumentenserver *DokuNet* um Sicherheitsmechanismen erweitert wurde, stellte sich die Frage, wie man auch die sogenannten Verkehrsinformationen gegen unbefugte Einsichtnahme schützen kann. Da für diese Frage keine überzeugende Antwort gefunden werden konnte, wurde *DokuNet* nie fertiggestellt.

[4]Für die exemplarische Implementierung wurde Postgres 95 benutzt. An dieser Stelle sei meinem Kommilitonen und Kollegen Tobias Kunze für die Implementierung der gemeinsam entwickelten Datenbankschnittstelle gedankt.


```
<html><title>Der Titel aus dem Standard Header</title>
<body>Der Standard Header ohne Informationen.<hr>
<H3>Arbeitsbereich Geschützt</H3><br>
<form method="POST" action="http://localhost:8080/cgi-bin/dokunet.cgi" >
<input type=hidden name="dn_ABereich" value="Geschützt" >
<input type=hidden name="dn_LoginName" value="Georg" >
<input type=hidden name="dn_Seite" value="VList" >
<input type=submit name=dn_sendfile value="Datei speichern"><br><hr><table>
<tr><th>Dateiname<th>Größe   <th>Dateityp   <th>Zeit der  <th>Besitzer
<tr><th>(Laden)  <th>in Bytes<th>(mime type)<th>Erstellung<th>LoginName
<tr><td><input type=submit name="dn_getfile" value="bild.gif"> <td> 42098 <td>
<tr><td><input type=submit name="dn_getfile" value="bild.pcx"> <td> 66947 <td>
<tr><td><input type=submit name="dn_getfile" value="dos.txt"> <td> 226 <td> text
<tr><td><input type=submit name="dn_getfile" value="ole.doc"> <td> 25600 <td> ap
<tr><td><input type=submit name="dn_getfile" value="unix.txt"> <td> 19657 <td>
</table></form>
<hr>Der Standard Footer ohne Informationen.
</body></html>
```

Abbildung 3.3: Der Arbeitsbereich „Geschuetzt“ als HTML-Seite und im Quelltext.

3.1.5 Absicherung der Verkehrsinformation

Der bisher dargestellte Entwurf verzichtete völlig auf die Absicherung der Verkehrsinformationen. Ein passiver Angreifer, der den Datenverkehr zwischen den Clients und dem Server abhört, kann bei Realisierung dieses Entwurfs feststellen, welche Benutzer es gibt, wo sie arbeiten, welche Arbeitsbereiche (Projekte, Abteilungen) existieren, wer welche Dokumente schreibt, bearbeitet, liest, beurteilt.

Diese nicht unwesentlichen Informationen einem Angreifer zur Verfügung zu stellen, der sich nur die geringe Mühe macht, eine Netzwerkverbindung abzuhören, während man den Inhalt der Dokumente gegen die stärksten bekannten Analyseverfahren absichert, erscheint – vorsichtig formuliert – unverhältnismäßig.

Es wurde nun untersucht, wie unter weitestgehender Beibehaltung des bisherigen Entwurfs die Verkehrsinformationen ebenso stark abgesichert werden können wie die Dokumente selbst.

Dazu könnte man auch die HTML-Seiten wie die Dokumente verschlüsselt an den Web-Browser senden. Hier entschlüsselt man sie mit einer Hilfsanwendung. Problematisch ist jetzt nur noch, wie die entschlüsselten HTML-Seiten „in“ den Web-Browser gelangen sollen.

Zu diesem Zweck existiert für viele Kombinationen aus Web-Browser und Fensteroberfläche eine mehr oder weniger elegante Möglichkeit zur Fernsteuerung. Insgesamt ist aber nicht zu erwarten, daß man für jede Plattform eine akzeptable Lösung findet. Eine solche Vorgehensweise entspricht auch nicht der Anforderung, Standardkomponenten und Standardschnittstellen zu benutzen.

Auch muß man sich darüber klar sein, daß bei dieser Lösung die Benutzeraktionen auf Formularen dennoch unverschlüsselt über das Netz gesandt werden.

An diesem Punkt wird klar, daß es einen wesentlich einfacheren Weg zur Absicherung eines web-basierten Dokumentenservers geben **muß** als den bisher eingeschlagenen. Es wurden weitere Vorschläge für kryptographische Protokolle auf ihre Eignung hin untersucht und zu guter Letzt doch noch eine tragfähige Lösung gefunden, die den gesamten Datenverkehr zwischen Web-Browser und HTTP-Server absichert.

3.2 Der zweite Entwurf: Verschlüsselung auf TCP/IP-Ebene

Weil die Benutzeraktionen im Browser immer noch nicht abgesichert werden konnten, wurde nach einem Weg gesucht, die gesamte TCP/IP-Kommunikation des Servers abzufangen und zu ver- bzw. entschlüsseln.

Dabei wurde zunächst an Manipulationen am Netzwerkcode des jeweiligen Betriebssystems gedacht, die unmöglich für jedes auf der Clientseite unterstützte Betriebssystem durchführbar sind, weil für die meisten Implementierungen kein Quellcode zur Verfügung steht. Wenig später kam die rettende Idee, jedem Client so etwas ähnliches wie einen Proxy-Server auf dem eigenen Rechner zur Verfügung zu stellen.

Normalerweise dient ein sogenannte HTTP-Proxy den Rechnern eines lokalen Netzwerkes als *cache* für HTML-Dokumente. Die meisten Web-Browser erlauben es, einen HTTP-Proxy zu konfigurieren, an den dann alle HTTP-Requests des Web-Browsers weitergeleitet werden. Zu diesem Zweck wird das HTTP leicht modifiziert: lautete die erste Zeile eines Requests in HTTP „GET dateiname HTTP/1.0“ so wird sie für den HTTP-Proxy durch die Serveradresse und die Portnummer beispielsweise zu „GET http://server.name:8080/dateiname HTTP/1.0“ ergänzt. Diese Erweiterung ist notwendig, damit der HTTP-Proxy Verbindung zum „richtigen“ HTTP-Server aufnehmen kann.

Genauere Informationen zum Proxy-Protokoll und zu Erweiterungen der HTTP-Headerinformationen findet sich in [71]. Die im Verlauf der Arbeit entwickelten Programme sind jedoch im Unterschied zu sogenannten *proxy caches* fest auf einen Server konfigurierbar und lassen sich nicht mit dem Proxy-Protokoll ansprechen. Trotzdem wird das auf der Benutzerseite eingesetzte Programm Client-Proxy genannt, da es einen Dienst „nahe“ am Benutzer anbietet. Auf der Serverseite ein *daemon* nötig, der den Datenstrom ebenfalls ver- und entschlüsselt und ihn an den eigentlichen HTTP-Server weitergibt. Dieser trägt aus Symmetriegründen den Namen Server-Proxy.

Grundsätzlich besteht auch die Möglichkeit, den Quellcode eines HTTP-Servers um kryptographische Routinen zu erweitern, aber man verstellt sich dadurch einige Entwicklungsmöglichkeiten: immer mehr Hersteller gehen dazu über, HTTP-Server und z.B. Datenbanken zu integrieren. Möchte man die Verbindung zu einem solchen Server absichern, so ist es sinnvoller einen separat laufenden Server-Proxy zu schreiben, der die Daten zwischen Krypto-Proxy und HTTP-Server weiterreicht, da man meist nicht den Quellcode eines solchen integrierten Programms zur Verfügung hat.

Im linken Teil der Abbildung 3.4 sieht man das Szenario einer gewöhnlichen Verbindung zwischen Web-Browser und HTTP-Server. Im rechten Teil der Abbildung 3.4 sieht man das geplante, abgesicherte Szenario. Der HTTP-Server darf nur noch Anfragen bearbeiten, die vom lokalen Rechner stammen. Um dies zu anzudeuten, ist das Symbol für den Port 80 in den Server gerückt. Um diese Bedingung zuverlässig prüfen zu können, ist ein Paketfilter nötig, der von außen kommende Pakete mit gefälschter lokaler Absendeadresse verwirft. Weitere Informationen zur Funktionsweise von Paketfiltern finden sich in [13, S. 134], [19, S. 54] und [14, S. 131].

Der in der Abbildung 3.4 dargestellte Aufbau setzt implizit voraus, daß die Verbindung zwischen Web-Browser und Client-Proxy auf dem Client-Rechner sicher ist. Ist diese Annahme vertretbar? Die Antwort auf diese Frage kann nur unter Beachtung jeder einzelnen Plattform gegeben werden, die auf der Client-Seite genutzt werden kann.

Bei der Verwendung von Windows ohne spezielle Erweiterungen auf der Clientseite kann die Sicherheit der Verbindung zwischen dem Web-Browser und dem Client-Proxy garantiert werden, da Windows zwar ein Multitasking, aber ein Single-User-Betriebssystem ist.

Bei der Verwendung von Unix auf der Clientseite muß vorausgesetzt werden, daß der Rechner gegen Einbrüche Unbefugter abgesichert wird. Ebenso muß man fordern, daß zugelassene Benutzer und insbesondere der Superuser vertrauenswürdig sind. Wie im Abschnitt 2.2 beschrieben, können kryptographische Verfahren eine Organisation nicht vor Angriffen

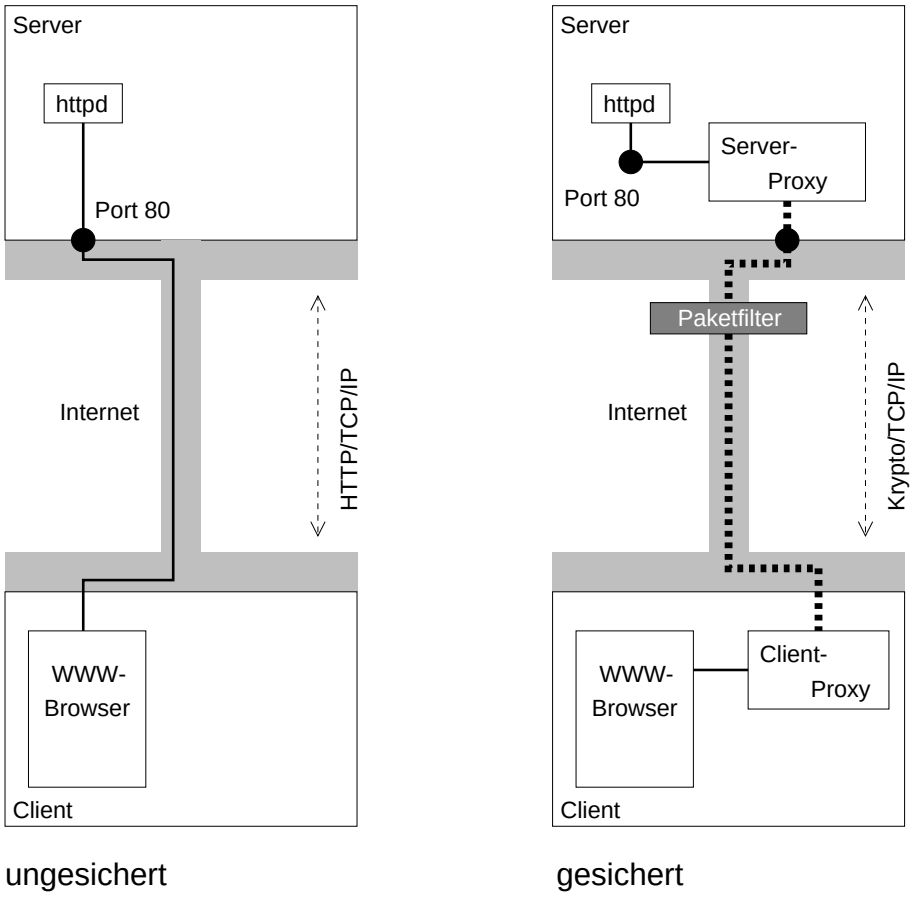

Abbildung 3.4: **Links:** der normale Kommunikationsweg zwischen Web-Browser und HTTP-Server. Der Port 80 ist auf dem Rand des Servers symbolisiert, um anzudeuten, daß Zugriffe von außen zugelassen sind. **Rechts:** die abgesicherte Kommunikation. Port 80 ist nicht mehr von außen zugänglich. Ein Paketfilter verhindert das Eindringen von Paketen mit gefälschter lokaler Absendeadresse.

„von Innen“ absichern, sodaß diese Voraussetzungen nicht als große Einschränkung zu bewerten sind.

3.2.1 Die Auswahl des Protokolls

Nun war die Frage zu klären, welches Protokoll zwischen dem Client-Proxy und dem Server-Proxy verwendet werden sollte.

Die Realisierungen von kryptographischen Protokollen, die auf Dokumentebene arbeiten, verarbeiten das Dokument in einem Schritt als Block. So muß ein HTTP-Request bzw. eine HTTP-Response erst komplett gepuffert werden, bevor er verschlüsselt bzw. entschlüsselt werden kann.

Dies erzeugt insbesondere auf der Serverseite eine unnötige Belastung, die nur bis zu einer begrenzten Benutzerzahl durch Investitionen in leistungsfähigere Hardware bewältigt werden kann.

Das SSL-Protokoll ist die einzige Lösung, die eine paketweise Verschlüsselung gestattet und so einen hohen Durchsatz bei geringer Rechnerbelastung verspricht. Das gesamte Datensegment der TCP-Pakete wird hierbei verschlüsselt. Die Wahl fiel auf SSL, da SSL auf der tiefst möglichen Ebene verschlüsselt, die noch direkt vom Anwendungsprogramm kontrolliert wird, dadurch bleibt ein Minimum an Verkehrsinformationen sichtbar.

Die von der Firma Netscape Communications, Corp. vertriebene Bibliothek SSLREF darf nicht aus den USA exportiert werden, aber mit der von Eric A. Young veröffentlichten Bibliothek SSLeay steht eine brauchbare Implementierung des SSL 2.0 zur Verfügung[5]. Der Autor gestattet die frei Verwendung dieser Bibliothek sogar in kommerziellen Produkten.

Bei der Entwicklung von SSLeay wurde versucht, die Umstellung von Protokollen, die auf den Systemaufrufen der Berkeley-Sockets aufsetzen, möglichst einfach zu halten. Sobald man auf hohem Niveau geklärt hat, welche Zertifikate und Schlüssel genutzt werden, kann man in den *low level*-Routinen die Systemaufrufe *read* und *write* gegen ihre SSL-Pendants austauschen. Man kann also im Extremfall sogar byteweise lesen und schreiben, genauso als würde man eine gewöhnliche TCP/IP-Verbindung benutzen.

SSLeay läßt sich unter praktisch jeder Unix-Variante mit einem einfachen C-Compiler kompilieren. Erfreulicherweise sind im Quellcode der Bibliothek SSLeay Vorkehrungen getroffen, um eine Portierung nach Windows zu erleichtern. Die mitgelieferten Konfigurationsdateien, Scripte und Makefiles sind allerdings nur eingeschränkt unter DOS verwendbar, wenn man eine breite Palette an frei verfügbarer Software unter MS-DOS installiert. Um den daraus resultierenden Aufwand einzusparen, wurden die notwendigen Compileraufrufe „von Hand“ durchgeführt.

SSL erfüllt weitgehend die Sicherheitsanforderungen des BSCW-Systems aus Abschnitt 1.3. Die Punkte **Integrität**, **Authentizität**, **Abhörsicherheit** und **Offenheit** werden von von SSL erfüllt. Durch die paketweise Verschlüsselung durch SSL soll die Implementierung leistungsfähig genug sein, um auch die Anforderung **Komfort** zu erfüllen.

3.2.2 Implementierung unter Unix

Die Implementierung unter Unix umfasste sowohl die Server- als auch die Clientseite. Leider ist die für SSLeay zu Verfügung stehende Dokumentation nicht sehr ausführlich und letztlich bleibt das intensive Studium des Quellcodes der mitgelieferten Beispielprogramme nicht erspart. Zum Einstieg sei hier der Artikel [72] empfohlen, der SSLeay Version 0.5.1b beschreibt.

Der Client-Proxy unter Unix arbeitet nach folgendem Schema:

```
Eingabe: clientport     // Port, der dem lokalen Web-Browser angeboten
                        // wird.
         severadresse   // Internetadresse des Servers.
         serverport     // Port des Server-Proxy.
         cacert         // Das Eigenzertifikat der Root-CA.
         owncert        // Das von der Root-CA ausgestellte Zertifikat
```

[5] Ftp-Server in Deutschland: ftp.uni-mainz.de/pub/internet/security/ssl/SSL .

```
                        // für diesen Benutzer.
         ownkey         // Der private Schlüssel dieses Benutzers.

Programm:

Initialisierungen.
Prüfung von Schlüsseln und Zertifikaten auf Gültigkeit.

while TRUE do
  Erwarte Request des Web-Browsers auf Port clientport.
  Erzeuge Kindprozeß.
  if Kindprozeß then
    Öffne SSL-Verbindung zu serveradresse:serverport.
    Lies Request von clientport und schreibe ihn auf SSL-Verbindung.
    Lies Antwort von SSL-Verbindung und schreibe nach clientport.
    Schließe SSL-Verbindung zum Server-Proxy.
    Schließe TCP-Verbindung zum Web-Browser.
    Terminiere Kindprozeß.
  fi
od
```

Der Server-Proxy unter Unix arbeitet nach folgendem Schema:

```
Eingabe: httpdport      // Port, der vom HTTP-Server angeboten wird.
         serverport     // Port des Server-Proxy.
         cacert         // Das Eigenzertifikat der Root-CA.
         owncert        // Das von der Root-CA ausgestellte Zertifikat
                        // für diesen Server.
         ownkey         // Der private Schlüssel dieses Servers.

Programm:

Initialisierungen.
Prüfung von Schlüsseln und Zertifikaten auf Gültigkeit.

while TRUE do
  Erwarte Request des Client-Proxy auf Port serverport.
  Erzeuge Kindprozeß.
  if Kindprozeß then
    Öffne SSL-Verbindung auf serverport.
    Lies Request von SSL-Verbindung und leite weiter an httpdport.
    Lies Antwort von httpdport und schreibe auf die SSL-Verbindung.
    Schließe TCP-Verbindung zum HTTP-Sever.
    Schließe SSL-Verbindung zum Client-Proxy.
    Terminiere Kindprozeß.
  fi
od
```

Beide Programme arbeiten nach dem üblichen Schema für TCP-Server unter Unix, das z.B. in [62, S. 284] beschrieben ist.

Zu diesem Schema gehört das Erzeugen eines Kindprozesses für jeden Request, um die Gesamtleistung des Systems durch Parallelisierung zu steigern. Diese Vorgehen ist zumindest bei Servern üblich, so auch bei gängigen HTTP-Servern. Meist wird die Zahl der Kindprozesse beschränkt, um die Servermaschine bei vielen, gehäuft eintreffenden Requests nicht zu überlasten.

Ungewöhnlich ist die Vorgehensweise, auch den Client-Proxy Kindprozesse erzeugen zu lassen. Einige Web-Browser bieten die Möglichkeit, mehrere Netzwerkverbindungen gleichzeitig zu öffnen, um z.B. mehrere Icons einer HTML-Seite „gleichzeitig“ zu laden. Zu Beginn bestand die Vermutung, ein parallelisierender Client-Proxy würde den Gesamtdurchsatz steigern können.

Im Zusammenhang mit der SSLeay-Bibliothek wird noch ein weiterer Vorteil von Kindprozessen sichtbar: SSLeay enthält (mit jedem Versionssprung weniger zahlreiche) Speicherlecks (*memory leaks*), die dadurch entstehen, daß mit `malloc()` belegter Speicher beim Schließen einer SSL-Verbindung nicht wieder freigegeben wird. Durch den sofortigen „Tod“ der Kindprozesse nach getaner Arbeit wird der gesamte belegte Speicherplatz zuverlässig wieder an das Betriebssystem zurückgegeben.

3.2.3 Implementierung unter Windows

Die Realisierung des Client-Proxy unter Windows verursachte erwartungsgemäß Probleme. Zunächst konnte SSLeay unter Windows 3.11 nicht mit Erfolg kompiliert und getestet werden.

Erst unter Benutzung der von Microsoft frei verbreiteten Betriebssystemerweiterung Win32s und des dadurch zur Verfügung stehenden flachen Speichermodells gelang es, SSLeay erfolgreich zu kompilieren.

Damit ist klar, daß der Client-Proxy unter Windows als sogenannte 32Bit-Anwendung kompiliert werden muß, was andererseits die Portierung auf Windows95 (und evtl. Windows NT) erleichtern sollte.

Bei der Portierung des Client-Proxy von Windows 3.11 nach Windows 95 trat nur ein einziges Problem auf. Herauszufinden, welches genau, war allerdings nicht ganz einfach.

Was W. Richard Stevens für den Programmierer von Unix-Anwendungen ist, ist Charles Petzold für den Windows-Programmierer, gleichgültig um welche Version dieses Systems es sich handelt. Charles Petzold erwähnt in der 95er-Version seiner Windows-Programmierbibel [73] ein einziges Mal auf der Seite 46 den Begriff der Reentranz, um dann 1050 Seiten lang kein Wort mehr darüber zu verlieren:

> But notice that the window procedure must be reentrant. That is, Windows often calls *WndProc* with a new message as a result of *WndProc* calling *DefWindowProc* with a previous message. in most cases the reentrancy of the window procedure presents no problem, but you should be aware of it.

Wie wahr. Die erste Version des Client-Proxy, die stabil unter Windows 3.11 mit Win32s lief, brach unter Windows 95 völlig zusammen.

Wie sich herausstellt, muß das Problem der Reentranz unter Windows 95 ernster genommen werden als bisher. Dies ist wahrscheinlich der Preis für die Möglichkeit echter Nebenläufigkeit innerhalb eines Prozesses (*threads*).

Das Win32-API stellt sogenannte Kritische Abschnitte (*critical sections*) zur Verfügung, die es ermöglichen, bestimmte Blöcke von Instruktionen zusammenzufassen, um sie quasi als atomare Aktionen auszuführen. Die Arbeitsweise solcher Mechanismen ist z.B. in [74, S. 44] und in [75, S. 17] beschrieben. Nachdem praktisch alle Zugriffe auf Variablen, die den Status einer Netzwerkverbindung darstellen, durch Kritische Bereiche geschützt waren, lief der Client-Proxy auch unter Windows 95.

Andere wertvolle Hinweise zur Portierung von 16Bit-Windowsprogrammen nach Win32s, Windows 95 und Windows NT finden sich in [76] und [77].

Leider kann unter Windows das übliche Unix-Server-Skelett nicht übernommen werden. Ein Programm darf nicht völlig zum Stillstand kommen, während es auf Requests auf einem Port wartet, sondern es muß weiter Nachrichten empfangen und bearbeiten. Das WinSock-API [78] erlaubt es vielmehr, sich über Zustandsänderungen an einem Port durch den gleichen Nachrichtenmechanismus informieren zu lassen, der auch die Änderung z.B.

der Fenstergröße an eine *window procedure* übermittelt. D.h., man bindet die Nachrichten eines Sockets an die Event-Queue eines Fensters.

Das Erzeugen eines Kindprozesses, der sämtliche Variablen und offene Dateien und Netzwerkverbindungen des Elternprozesses erbt, ist unter Windows leider nicht möglich.
Der Client-Proxy unter Windows arbeitet nach folgendem Schema:

```
Eingabe: clientport     // Port, der dem lokalen Web-Browser angeboten
                        // wird.
         severadresse   // Internetadresse des Servers.
         serverport     // Port des Server-Proxy.
         cacert         // Das Eigenzertifikat der Root-CA.
         owncert        // Das von der Root-CA ausgestellte Zertifikat
                        // für diesen Benutzer.
         ownkey         // Der private Schlüssel dieses Benutzers.

Programm:

Initialisierungen.
Prüfung von Schlüsseln und Zertifikaten auf Gültigkeit.
WinSock: Request auf clientport löst ACCEPT-Event aus.

while TRUE do
  switch EVENT do
    case WindowEvent: Tue, was Windowsprogramme tun müssen.
    case ACCEPT     : Sperre weitere ACCEPTS.
                      Öffne SSL-Verbindung zu serveradresse:serverport.
                      Konfiguration der WinSocks:
                        1. Write auf serverport bzw. clientport löst
                           WRITE-Event aus.
                        2. Abbrechen einer Verbindung löst CLOSE-Event aus.
    case WRITE      : Lies von der entsprechenden Verbindung und
                      schreibe auf die andere.
    case CLOSE      : Schliesse beide Verbindungen und gestatte weitere
                      ACCEPTS auf clientport.
  od
od
```

Schon bei den ersten Experimenten dem Client-Proxy unter Unix wurde klar, daß die Fähigkeit des Client-Proxy, mehrere Verbindungen gleichzeitig offenzuhalten, keine Geschwindigkeitsvorteile brachte, eher im Gegenteil. Aus diesem Grund wurde bei der Windows-Version der Einfachheit halber auf diese Fähigkeit verzichtet.

3.2.4 Probleme

Diese Lösung arbeitet stabil und zuverlässig. Leider kann man sie nicht als praxistauglich bezeichnen. Das Laden der in Abbildung 3.5 auf Seite 67 dargestellten HTML-Seite des BSCW-Systems dauert weit über 2 Minuten, unabhängig davon, ob man Windows oder Unix auf der Client-Seite benutzt.

Die HTTP-Requests, die der Web-Browser zum Aufbau dieser Seite an den Client-proxy stellte, wurden mitprotokolliert und nach der ersten Zeile des HTTP-Headers durchsucht, mit folgendem Ergebnis:

```
GET /workspaces/1_GeorgsPlace/ HTTP/1.0
GET /icons/WScompass.gif HTTP/1.0
GET /icons/BSCW_AddDocument.gif HTTP/1.0
```

```
GET /icons/BSCW_AddLink.gif HTTP/1.0
GET /icons/BSCW_AddFolder.gif HTTP/1.0
GET /icons/BSCW_EditBanner.gif HTTP/1.0
GET /icons/BSCW_Catchup.gif HTTP/1.0
GET /icons/WSeventToggleOn.gif HTTP/1.0
GET /icons/WSdescToggleOff.gif HTTP/1.0
GET /icons/WSactionToggleOff.gif HTTP/1.0
GET /icons/WSlinkEnd.gif HTTP/1.0
GET /icons/WSworkspace.gif HTTP/1.0
GET /icons/WSinfo.gif HTTP/1.0
GET /icons/WStext.gif HTTP/1.0
GET /icons/WSnewEvent.gif HTTP/1.0
GET /icons/WSwriteEvent.gif HTTP/1.0
GET /icons/WSversionEvent.gif HTTP/1.0
GET /icons/WSunknown.gif HTTP/1.0
GET /icons/WSpostscript.gif HTTP/1.0
GET /icons/WSfolder.gif HTTP/1.0
GET /icons/WSsubEvent.gif HTTP/1.0
GET /icons/WSgroup.gif HTTP/1.0
GET /icons/WSemptyTrash.gif HTTP/1.0
```

Das sind insgesamt 23 Zugriffe zum Aufbau einer einzigen Seite. Für jeden Zugriff wird eine SSL-Verbindung aufgebaut, d.h. es werden Zertifikate ausgetauscht und überprüft, Verschlüsselungsverfahren verhandelt usw.

Der Aufbau einer SSL-Verbindung benötigt etwa vier bis fünf Sekunden. Dabei spielt die Größe der übertragenen Grafiken praktisch keine Rolle für die benötigte Gesamtzeit des Seitenaufbaus.

Experimente unter Unix, in denen mehrere Netzwerkverbindungen gleichzeitig geöffnet wurden, erbrachten keinen Geschwindigkeitsvorteil durch Parallelisierung, sondern die Leistung brach weiter ein.

Dieser Effekt macht sich nach einer Weile nicht mehr ganz so katastrophal bemerkbar, da die meisten Web-Browser Bilddateien auf der lokalen Platte abspeichern und nur erneut laden, wenn sie dazu explizit aufgefordert werden. Das Nachladen aller Bilder wird beispielsweise durch ein *reload* der Seite ausgelöst und ist dann meist unbeabsichtigt.

Das Laden einer neuen Seite, die keine neuen eingebetteten Grafiken enthält, benötigt aber weiterhin trotz *caching* vier bis fünf Sekunden, selbst über eine schnelle Netzwerkverbindung oder wenn sämtliche beteiligten Progamme auf ein und demselben Rechner laufen.

Leider muß man zusammenfassend sagen, daß die erreichte Leistung für ein kontinuierliches Arbeiten mit dem BSCW-System inakzeptablel ist. Der nächste Abschnitt beschäftigt sich mit dem Versuch, die Übertragungsleistung zu steigern.

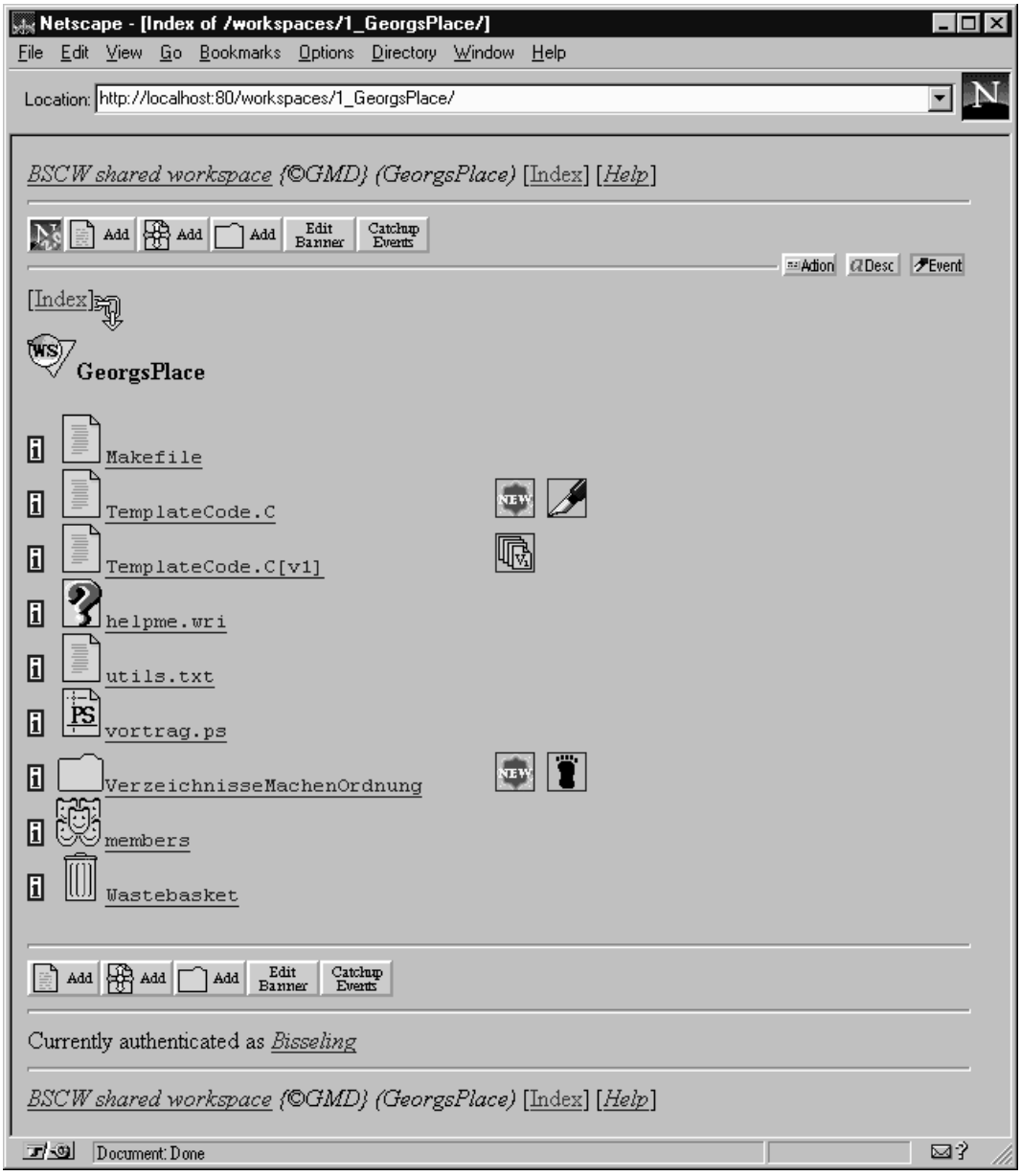

Abbildung 3.5: Eine ganz normale BSCW-Seite mit 22 verschiedenen eingebetteten Grafiken.

3.3 Leistungssteigerung der Implementierung

Die enttäuschende Leistung der oben dargestellten Lösung führte zu weiteren Experimenten. Die in der folgenden Tabelle dargestellten Meßwerte wurden auf einem Linux-Rechner durchgeführt, auf dem alle beteiligten Programme lokal liefen. Als Verschlüsselungsverfahren wurde DES im CBC-Modus verwendet. Die Meßwerte sind handgestoppt bzw. vom *Netscape Navigator* abgelesen. Die absoluten Werte sind nur auf genau diesem Rechner so reproduzierbar, die Verhältnisse zwischen den Übertragungsleistungen sollten aber typisch sein:

Datenart	Durchsatz unverschlüsselt kByte/s	Durchsatz verschlüsselt kByte/s
Leere HTML-Seite mit zehn Icons zu 1 kByte	10,0	0,2
860 kByte große Textdatei	130,0	60,0

Man sieht, daß der Datendurchsatz bei der Übertragung vieler kleiner einzelner Dateien wesentlich stärker gegenüber dem unverschlüsselten Durchsatz einbricht als bei der Übertragung einer großen Datei.

Bei der Betrachtung dieser Meßwerte darf nicht vergessen werden, daß die Daten bei Verschlüsselung durch zwei zusätzliche Prozesse geleitet werden, während die Daten ohne Verschlüsselung direkt vom HTTP-Server zum Web-Browser gelangen.

3.3.1 Die Ursache der mangelnden Leistung

Das HTTP-Protokoll löst sehr viele kleine Transfers aus. Die Anfragen der Clients liegen häufig bei etwa 300 Bytes, typische Antworten der HTTP-Server liegen um 3000 Bytes ([5, S. 172]). Diese Zahlen dürften ebenso für ein BSCW-System gelten, wenn man vom eigentlichen Dokumententransfer absieht. Stevens schlägt aus diesem Grunde in [5, Teil 1] auf etwa 150 Seiten ein neues Protokoll *TCP for Transactions* (T/TCP), eine Modifikation des TCP (*transmission control protocol*) vor, um den Overhead des TCP zu vermeiden.

Stevens stellt sehr überzeugend dar, warum das HTTP zu *performance problems* führt. TCP ist an Byte-Strömen orientiert, während HTTP aus einzelnen Transfers besteht. TCP-Verbindungen sind so realisiert, daß sie sicher über Tage und Wochen bestehen bleiben können, sie sollen nicht häufig abgebrochen und wiederaufgebaut werden.

Das hauptsächliche Problem dabei, das HTTP-Protokoll auf einem *stream*-Protokoll zu realisieren, liegt darin, daß eine Antwort auf einen Request im HTTP keine Längenangabe besitzen muß. Überträgt ein HTTP-Server als Antwort auf einen Request eine Datei, die in seinem lokalen Dateisystem liegt, so kann er im HTTP-Header leicht eine Längenangabe setzen, da ihm die Größe der Datei vor Beginn der Übertragung bekannt ist.

Die HTML-Seiten des BSCW-Systems werden jedoch von Programmen erzeugt, die über das *common gateway interface* (CGI) aufgerufen werden. Im wesentlichen wird der Zeichenstrom, den die Programme auf ihrer Standardausgabe produzieren, in die Netzwerkverbindung umgeleitet.

Terminiert das Programm, so schließt der HTTP-Server die Verbindung zum Web-Browser und zeigt so das Ende der Übertragung an. Dieses eher schlichte Vorgehen erlaubt es, auf der Serverseite *quick & dirty* sogenannte CGI-Skripte zu programmieren, deren Ausgabe nicht im lokalen Dateisystems des Servers gepuffert werden muß, nur um die Größe der Ausgabe festzustellen.

3.3.2 Lösungsentwurf

Stevens verweist in [5, S. 175] auf einen HTTP-NG genannten Vorschlag, der vorsieht, HTTP auf einem paketorientierten Zwischenprotokoll aufzusetzen. Dieses Zwischenprotokoll setzt seinerseits direkt auf TCP/IP auf. Es überträgt die Daten paketweise und erlaubt es, zusätzliche Informationen wie „Ende dieses Transfers“ zu übertragen, ohne die TCP/IP-Verbindung zu unterbrechen.

Die inakzeptable Übertragungsleistung der vorliegenden Implementierung hat die gleichen Ursachen wie die von Stevens beklagten *performance problems*, nur in größerem Ausmaß, da der Overhead beim Aufbau einer SSL-Verbindung um ein Vielfaches größer ist als der beim Aufbau einer reinen TCP/IP-Verbindung[6].

Aus diesem Grund kann man die Leistung der oben beschriebenen Implementierung auch mit ähnlichen Mitteln steigern. Der verbesserte Entwurf sieht einen Krypto-Proxy auf dem Clientrechner vor, der bei seinem Start einmalig eine SSL-Verbindung zu einem Krypto-Daemon auf dem Server aufbaut, d.h. auf dem Server wird für jeden aktiven Client ein Kindprozeß des Krypto-Daemons erzeugt. Diese SSL-Verbindung bleibt bestehen, bis entweder der Benutzer den Krypto-Proxy auf der Clientseite beendet oder der Server wegen Ressourcenmangels[7] die Verbindung kappt.

Um Beginn und Ende einzelner HTTP-Requests und -Responses zu erkennen, benutzen die beiden Kryptoprogramme ein Zwischenprotokoll, das auf dem SSL-Protokoll aufsetzt. Das Zwischenprotokoll ist paketorientiert. Pakete bestehen aus einem Header, der wiederum nur ein 16Bit *signed integer* in *network byteorder* ist, gefolgt von einem optionalen Datenpaket.

Die verschiedenen Pakettypen sind wiefolgt aufgebaut:

Header	Bedeutung
$h \geq 0$	Es folgen h Bytes Daten.
$h = -1$	Ein Request/eine Response ist beendet.
$h = -2$	Die Gegenstelle verabschiedet sich endgültig.
$h = -3$	Ping Request
$h = -4$	Ping Response

Der Client-Proxy arbeitet nach folgendem Schema:

```
Eingabe: clientport     // Port, der dem lokalen Web-Browser angeboten
                        // wird.
         severadresse   // Internetadresse des Servers.
         serverport     // Port des Server-Proxy.
         cacert         // Das Eigenzertifikat der Root-CA.
         owncert        // Das von der Root-CA ausgestellte Zertifikat
                        // für diesen Benutzer.
         ownkey         // Der private Schlüssel dieses Benutzers.

Programm:

Initialisierungen.
Prüfung von Schlüsseln und Zertifikaten auf Gültigkeit.
Öffne SSL-Verbindung zum Server-Proxy.
Setze Zwischenprotokoll auf SSL-Verbindung

while TRUE do
  Erwarte Request des Web-Browsers auf Port clientport.
```

[6] Eine SSL-Verbindung beinhaltet immer eine TCP/IP-Verbindung.

[7] Ein denkbares Schema wäre es, bei zu vielen Kindprozessen diejenigen Prozesse zu terminieren, die relativ lange inaktiv waren.

```
  Erzeuge Kindprozeß.
  if Kindprozeß then
    Lies Request und schreibe ihn im Zwischenprotokoll zum Server-Proxy.
    Schreibe Request-Ende auf Zwischenprotokoll.
    Lies Response paketweise vom Server-Proxy bis das Zwischenprotokoll
         das Ende des Requests meldet und schreibe Pakete zum Web-Browser.
    Schließe Verbindung zum Web-Browser.
  fi
od
```

Dieser Pseudocode läßt nur die Frage offen, wie der Client-Proxy das Ende des HTTP-Requests des Web-Browsers erkennt. HTTP-Requests bestehen entweder nur aus einem HTTP-Header, der mit einem doppelten Zeilenende terminiert wird, oder enthalten im Header eine Längenangabe über die Daten, die dem Header folgen. Der Client-Proxy „versteht“ also das HTTP wenigstens soweit, um das Request-Ende erkennen zu können.

Der wesentliche Unterschied der Implementierung dieses Client-Proxy gegenüber der des alten ist, daß die SSL-Verbindung vor dem ersten Request aufgebaut und nicht mehr geschlossen wird[8].

Der Pseudocode des zugehörigen Server-Proxy sollte klar sein: für jeden Request eines Client-Proxy wird ein Kindprozess gestartet und eine SSL-Verbindung mit Zwischenprotokoll aufgebaut. In der momentanen Version existiert dieser Kindprozeß bis der Client-Proxy die Verbindung aktiv über das Zwischenprotokoll schließt.

Stirbt der Client-Proxy aus irgendwelchen Gründen, ohne sich „abzumelden“, so bemerkt der zugehörige Kindprozess auf der Serverseite dies bei dem Versuch, weitere Daten über das Zwischenprotokoll zu lesen. Da die TCP-Verbindung, die der SSL-Verbindung zugrundeliegt, schon geschlossen wurde, stirbt er dann ebenfalls.

3.3.3 Erfolgsnachweis

Eine Implementierung, die das oben beschriebene Zwischenprotokoll benutzt, wurde unter Unix realisiert. Unter den gleichen Randbedingungen wie oben wurden bei der HTML-Seite mit eingebetteten Grafiken über 5 kByte/s erreicht, bei der etwa ein MByte großen Datei wurden wie bei der ersten Lösung über 60 kByte/s erreicht. Die Leistungsbilanz sieht nun so aus:

Datenart	Durchsatz unverschlüsselt kByte/s	Durchsatz verschlüsselt (alt) kByte/s	Durchsatz verschlüsselt (neu) kByte/s
Leere HTML-Seite mit zehn Icons zu 1 kByte	10,0	0,2	5,0
860 kByte große Textdatei	130,0	60,0	60,0

Damit ist das wesentliche Ziel erreicht: die Verschlüsselung ist nicht mehr das *bottle neck* verglichen mit gängigen Durchsatzraten auf dem World-Wide-Web.

Um noch einmal genau herauszufinden, wie der Zeitaufwand zwischen Verschlüsselung und dem Transport der Daten durch zwei zusätzliche Prozesse verteilt ist, wurde die große Textdatei auch einmal mit der Verschlüsselungsmethode „identische Abbildung“ über die „neue“ Lösung transportiert: Netscape mißt in diesem Fall über 90 kByte/s. Daraus ergibt sich, daß die Verschlüsselung mit DES im CBC-Modus genauso wenig Leistung kostet wie der reine Transport-Overhead durch die beiden zusätzlichen Prozesse.

Es muß eigentlich überraschen, daß zwischen zwei Prozessen auf dem gleichen Rechner nur ein Datendurchsatz von 120 kByte/s zu erreichen ist, obwohl der Netzwerk-Code z.B. die

[8] Dadurch machen sich die Speicherlecks in SSLeay erfreulicherweise auch nicht mehr bemerkbar

Bandbreite von Ethernet-Hardware von theoretisch einem Kilobyte pro Sekunde ausnutzen soll. Die Verzögerungen werden entweder in den beiden Programmen Web-Browser und HTTP-Server verursacht oder im Kernel.

Eine weitere Beschleunigung ist wohl nicht mehr auf Seiten der Ver- und Entschlüsselung möglich, sondern durch Veränderungen des HTTP. So sollten Längenangaben in den Responses der HTTP-Server Pflicht werden. Auf diese Weise könnte man sich den Overhead eines Zwischenprotokolls ersparen. Eine andere Möglichkeit, die ebenfalls in [5, S. 174] erwähnt wird, wäre es, HTML-Seiten inklusive der benötigten Icons in einem Request-Response-Paar zu transferieren – natürlich nur die Icons, die der Web-Browser noch nicht lokal gepuffert hat.

3.4 Resümee

Die Entwicklung und das Scheitern des ersten Entwurfs zeigen deutlich, daß das Gute nicht immer so nahe liegt, wie angenommen. In die Implementierung des ersten Entwurfs und in die Einarbeitung in *plug in's*, Java und diverse „Fernsteuerungsmechanismen“ für Web-Browser wurde sehr viel Arbeitszeit investiert.

Es dauerte relativ lange, bis die Entscheidung fiel, nach anderen Möglichkeiten zu suchen. Als die Idee zum zweiten Entwurf gefunden war, eröffnete sich zum ersten Mal die Möglichkeit, wirklich frei zu wählen, auf welcher Ebene die Verschlüsselung stattfindet. Im ersten Entwurf wäre z.B. die Verwendung von SSL gar nicht möglich gewesen. SSL ist sicherlich eine gute Wahl, da man die hier vorgestellte Lösung prinzipiell auch für völlig andere Anwendungsprotokolle als HTTP (wieder)verwenden kann.

Die Realisierung des zweiten Entwurfs unter Unix gelang relativ schnell, da für dieses Betriebssystem hochwertiger Quellcode für die Lösung fast aller Aufgaben der Netzwerkkommunikation vorliegt, der auf etwa 25 Jahren Programmiererfahrungen beruht.

Abgesehen von rein technischen Problemen, die gewählte SSL-Bibliothek unter Windows zu kompilieren, lag das Hauptproblem der Realisierung des Client-Proxy unter Windows darin, kurz vorher für die Implementierung unter Unix mühsam erlernte Programm-Schemata nun wieder aufgeben zu müssen.

Das Ergebnis der Bemühungen war zwar eine „funktionsfähige Machbarkeitsstudie“, aber die Übertragungsleistung war inakzeptabel bis frustrierend.

Die Suche nach dem eigentlichen Grund für die miserable Leistung führte zu dem überraschenden Ergebnis, daß das HTTP-Protokoll in seiner jetzigen Form sehr verschwenderisch mit der Übertragungskapazität des Netzes umgeht und eigentlich auf dem falschen Internet-Protokoll aufsetzt: zumindest die Requests wären besser über das *user datagram protocol* (UDP) realisiert, um den aufwendigen Handshake beim Aufbau von TCP-Verbindungen zu vermeiden. Ersetzt man die TCP-Verbindung durch eine SSL-Verbindung und behält ansonsten das übliche Vorgehen beim Bearbeiten von HTTP-Nachrichten bei, wird das zu „schlichte“ Design des HTTP offensichtlich.

Angesichts ungeschickt entworfener Protokolle dürfte auch handoptimierter Assemblercode zur Realisierung der kryptographischen Verfahren kaum zur Steigerung des Durchsatzes beitragen können. Offensichtlich wird hier mit zweierlei Maß gemessen, da die meisten Menschen den Rechenaufwand für die Verschlüsselung intuitiv überschätzen.

Die verbesserte Implementierung unter Unix umgeht die wesentlichen Schwächen von HTTP-Verbindungen, ohne in die Arbeitsweise von Web-Browser und HTTP-Servern einzugreifen und kann so den Maximalforderungen aus Abschnitt 1.3 voll gerecht werden.

Kapitel 4

Bewertung und Ausblick

Ziel der Arbeit waren der Entwurf und die exemplarische Implementierung eines Sicherheitsmechanismus' für das BSCW-System. Zu Beginn der Arbeit wurden nach einer Analyse des Einsatzzwecks und der verwendeten Techniken des BSCW-Systems die Maximalanforderungen an einen Sicherheitsmechanismus formuliert. Zu diesem Zeitpunkt war noch nicht klar, inwieweit diese Anforderungen erfüllt werden können.

Im zweiten Kapitel wird neben der kurzen Beschreibung der prominentesten, kryptographischen Verfahren ein Schema zur Bewertung kryptographischer Protokolle entwickelt. Als wesentliches Kriterium zur Bewertung eines Protokollvorschlags wird die Verschlüsselungsebene eingeführt. Verschlüsselung in der Netzwerkkommunikation findet grundsätzlich auf vier verschiedenen Ebenen statt: auf der Dokumentebene, auf der Ebene der Anwendungsprotokolle, auf Ebene der Netzwerkprotokolle und auf der Verbindungsebene.

Anhand der entwickelten Kriterien werden einige Protokollvorschläge bewertet. Die Auswahl ist weder erschöpfend noch repräsentativ, denn es finden sich praktisch für jedes Übertragungsprotokoll, gleich welcher Ebene, Vorschläge für kryptographische Erweiterungen. Diese scheinbare Vielfalt läßt sich aber weitgehend auf einen gemeinsamen Prototyp eines kryptographischen Protokolls zurückführen, der ebenfalls in diesem Kapitel skizziert wird.

Bei der Beschreibung der Implementierung wird im dritten Kapitel zunächst ein Entwurf vorgestellt, der entstand, lange bevor die Einarbeitung in kryptographische Verfahren und Protokolle abgeschlossen war. Der erste Entwurf war demnach sehr stark durch das augenscheinlich technisch einfach Machbare geprägt und weniger durch wohlüberlegte Entwurfsentscheidungen. Er ist im nachhinein als einer jener sprichwörtlichen Fehler zu sehen, aus denen man lernt. Schließlich wurde kategorisch gefordert, daß eine bessere Lösung existieren muß, ohne zunächst eine genaue Vorstellung von ihr zu haben.

Am Anfang der Entwicklung des zweiten Entwurfs stand die Idee, daß es möglich sein sollte, die gesamte Netzwerkkommunikation des Web-Browsers abzufangen und zu ver- bzw. entschlüsseln. Dabei wurde zunächst an völlig falscher Stelle, nämlich im Netzwerkcode der Client-Betriebssysteme, die einzige Möglichkeit vermutet, dieses Ziel zu erreichen. Natürlich ist eine Modifizierung dieses Codes nicht nur wegen des hohen Arbeitsaufwandes bei der Unterstützung verschiedener Client-Betriebssysteme praktisch undurchführbar: für die meisten Betriebssysteme steht der entsprechende Quellcode gar nicht zur Verfügung. Erst die eher beiläufige Beschäftigung mit Firewalls und Proxy-Servern führte zu der sauberen Lösung aus dem zweiten Entwurf.

Dieser Entwurf ermöglicht durch seine Struktur die freie Entscheidung, ob auf Dokumentebene, auf Ebene des HTTP oder auf Ebene der Netzwerkprotokolle verschlüsselt werden soll. Die Wahl, auf Ebene der Netzwerkprotokolle zu verschlüsseln, stellt hier also eine echte Entwurfsentscheidung dar, während die Wahl der Dokumentebene im ersten Entwurf eher durch Sachzwänge entschieden wurde.

Schon die erste Implementierung des zweiten Entwurfs erfüllte die Maximalanforderungen an einen Sicherheitsnmechanismus für das BSCW-System nahezu vollständig. Lediglich die Forderung nach größtmöglichem Komfort wurde nicht erfüllt, da die Arbeitsgeschwindigkeit der Implementierung inakzeptabel niedrig war. Die niedrige Leistung war praktisch unabhängig von der verwendeten Hardware und ebenso annähernd unabhängig vom konkret eingesetzten Verschlüsselungsverfahren. Auch das Volumen der transferierten Daten spielte eine untergeordnete Rolle. Als Ursache wurde eine Schwäche des HTTP gefunden, die auch schon ohne die Verwendung von Verschlüsselung sehr viel Netzwerkbandbreite durch häufigen Verbindungsauf- und -abbau verschwendet. Der zusätzliche Overhead beim Aufbau einer verschlüsselnden Verbindung läßt diese ärgerliche Schwäche als einen inakzeptablen Designfehler erscheinen.

Durch die Einführung eines weiteren paketorientierten Protokolls zwischen dem HTTP und dem Verschlüsselungsprotokoll konnte eine permanente Verbindung zwischen Client und Server sichergestellt werden, deren Übertragungsleistung auch bei der Verwendung von Verschlüsselungsverfahren so hoch ist, daß bei der Benutzung kaum ein merklicher Leistungsunterschied zur unverschlüsselten Kommunikation entsteht. Unter sehr günstigen Bedingungen mag diese Implementierung durch die permanente Verbindung sogar eine höhere Leistung erbringen als die Verwendung einer gewöhnlichen HTTP-Verbindung ohne Verschlüsselung.

Mit der verbesserten Implementierung des zweiten Entwurfs ist das Ziel der Arbeit vollständig erreicht. Obwohl die entwickelte Lösung durchweg positiv zu bewerten ist, bestehen noch Möglichkeiten zur Erweiterung und Verbesserung. Im folgenden werden Möglichkeiten zur Weiterentwicklung der konkreten Lösung beschrieben. Im Anschluß daran wird kurz auf die Möglichkeiten der Weiterentwicklung der Kryptographie in der elektronischen Kommunikation eingegangen, wobei hier nur eine kurze, subjektive Einschätzung gegeben werden kann.

4.1 Weiterentwicklung der Implementierung

4.1.1 Verwendung von Chipkarten

Fast alle praktischen Lösungsvorschlägen haben eines gemeinsam: die privaten Schlüssel für das RSA-Verfahren, also die Identität des Benutzers, werden in Dateien des lokalen Dateisystems abgelegt. Diese Dateien sind meist durch ein symmetrisches Verfahren geschützt, dessen Schlüssel aus einer sogenannten *pass phrase* abgeleitet wird. Damit kann diese Datei zumindest gegen Zugriffe aus dem lokalen Netzwerk kaum abgesichert werden. Zugriffe von außen müssen durch geeignete Schutzmechanismen vereitelt werden.

Problematisch an dieser Lösung ist, daß eine Kompromittierung, d.h. ein ungewolltes Bekanntwerden, von privaten Schlüsseln nicht bemerkt wird. Ein Angreifer, dem es gelingt die Datei mit dem privaten Schlüssel zu kopieren, kann beliebig großen Aufwand treiben, um die Verschlüsselung der Datei zu brechen. Wurde hier z.B. DES verwendet, so ist der Rechenaufwand bei geeigneter Hardware handhabbar.

Das *Security Development Environment* (SecuDE) ist die einzige Ausnahme unter den im Abschnitt 2.6.3 beschriebenen Protokollen. SecuDE bietet die Möglichkeit, den privaten Schlüssel des RSA-Verfahrens auf einer Chipkarte zu verwahren. Der Schlüssel verläßt die Karte auch während der RSA-Berechnungen nicht. Die Karte enthält einen Microcontroller, der den RSA-Algorithmus mit dem auf der Karte gespeicherten RSA-Schlüssel berechnen kann. Da nur relative wenige Daten direkt mit dem RSA-Verfahren verschlüsselt werden müssen, ist diese Lösung sogar mit einem seriell angeschlossenen Chipkarten-Terminal praktikabel.

Auf diese Weise kann ein Angreifer, selbst wenn er den Rechner des Benutzers kontrolliert, den privaten Schlüssel prinzipiell nicht in Erfahrung bringen. Unter hohem Aufwand und mit hohem Risiko, entdeckt zu werden, kann die Software auf dem lokalen Rechner theoretisch so manipuliert werden, daß ein Mißbrauch des privaten Schlüssels möglich wird.

Dieser Aufwand ist für den Angreifer um einiges höher als der, sich Root-Rechte auf dem lokalen Rechner zu beschaffen. Prinzipiell ist die Integration von Chipkarten auch bei der Verwendung des SSL-Protokolls realisierbar und erstrebenswert.

4.1.2 Integration mit der *base authentication*

Sichert man das BSCW-System mit dem hier vorgeschlagenen Proxy-Mechanismus ab, so existieren zwei orthogonale Authentisierungsmethoden nebeneinander: die *base authentication* des BSCW-Systems und die Authentisierung durch Zertifikate.

Diese Struktur verhindert einerseits, daß sich die beiden Mechanismen gegenseitig stören, andererseits ist dadurch die Zuordnung zwischen dem Zertifikat eines Anwenders und seinem Benutzernamen im BSCW-System nur sehr locker.

Nehmen wir an, Bob besucht Alice in ihrem Büro. Alice und Bob benutzen nun gemeinsam auf Alice' Rechner das BSCW-System. Sie arbeiten also auf Alice' Unix-Account mit Alice' Zertifikat und mit Alice' *base authentication*. Nun möchte Bob ein Dokument aus einem Arbeitsbereich holen, zu dem Alice keinen Zugang hat. Dies ist möglich, wenn er seine *base authentication* zum Zugriff auf diesen Arbeitsbereich benutzt. Nun erscheint er dem BSCW-System als Bob, obwohl er mit Alice' Zertifikat arbeitet. In dieser Situation erscheint dies als vorteilhaft, aber unter Umständen ergeben sich daraus Angriffsmöglichkeiten. Eine mögliche Lösung ist es, den Namen aus dem verwendeten Zertifikat auf den Namen in der *base authentication* abzubilden.

Eine saubere Lösung ergibt sich erst dann, wenn man auf jeden anderen Mechanismus als den der Zertifikate verzichtet. Erst dann können Zertifikate ihren Vorteil voll ausspielen: ein neuer Benutzer bekommt ein neues Zertifikat und kann das System sofort benutzen, da das System ihn selbstständig als neuen und zugelassenen Benutzer erkennt und ihn in seine Datenbanken einträgt.

4.2 Weiterentwicklung der Verwendung von Kryptographie

4.2.1 Der Markt

Kreditkartengesellschaften und Banken haben großes Interesse an der Standardisierung kryptographischer Protokolle, um sich den Markt des sogenannten *electronic shopping* zu eröffnen. Hier liegen große Chancen für die Vereinheitlichung von Lösungen, die auf Chipkarten beruhen, weil die Kosten für Entwicklung und Hardware durch die hohen Stückzahlen vernachlässigbar werden. Es darf vermutet werden, daß die Kosten für die Herstellung von Chipkarten auch heute schon relativ gering sind, angesichts von Telefongebührenkarten im Nennwert von zwölf DM und weniger.

Genauso gut läßt sich argumentieren, daß Banken dazu tendieren, proprietäre Verfahren zu verwenden und sich dabei nicht unbedingt geschickt verhalten. Hier wird nach dem Prinzip *security through obscurity* gearbeitet, das der Kerkhoff'schen Maxime widerspricht.

Es bleibt abzuwarten, ob der SET-Standard in seiner endgültigen Form der Allgemeinheit zugänglich gemacht wird oder, ob Implementierungsdetails hinter verschlossenen Türen festgelegt werden, nachdem die grundsätzliche Tauglichkeit des Standards von niemandem angezweifelt wird.

Wie das Beispiel der Algorithmen RC2 und RC4 zeigt, ist das Interesse an kryptographischen Verfahren und Protokollen einfach zu groß, um sie auf lange Sicht geheimhalten zu können.

4.2.2 Die Politik

Der Möglichkeit der Entwicklung hin zu einer breiten Akzeptanz kryptographischer Verfahren widersprechen einige Signale. Möglicherweise wird die Verwendung von Kryptographie im zivilen Bereich soweit eingeschränkt, daß sie nicht mehr direkt durch den Einzelnen kontrolliert werden kann.

Amerika spielt in dieser Hinsicht eine Vorreiterrolle, aber auch das französische Verbot der Nutzung von Kryptographie durch den Normalbürger ist alarmierend. Bruce Schneier widmet der Politik ein eigenes Kapitel [16, Kapitel 25]. Die amerikanischen Ausfuhrbestimmungen stehen der Entwicklung internationaler Standards auch in Europa entgegen, weil der Markt für Standardanwendungen von amerikanischen Herstellern dominiert wird.

Für national gültige Einschränkungen der Verwendung von Kryptographie wird meist die Sicherung der Verbrechensbekämpfung zur Argumentation herangezogen. Diese Argumentation ist aus zwei Gründen fragwürdig.

Erstens existieren Präzedenzfälle für die Einführung von Technologien, die auch für kriminelle Aktionen verwendet werden können. Meines Wissens nach ist noch niemand auf den Gedanken gekommen, Telefone zu verbieten, weil die Polizei seither nicht mehr erkennen kann, welche Personen einen Verdächtigen zu Hause aufsuchen.

Zweitens läßt sich die Verwendung kryptographischer Methoden nicht unbedingt zuverlässig nachweisen. Die Ausgabe guter kryptographischer Verfahren läßt sich nicht zuverlässig von zufälligem Rauschen unterscheiden. Es existiert Software, die eine solche Ausgabe mit in das Hintergrundrauschen eines digitalisierten Bildes mischt. Hat man nicht das unmanipulierte Originalbild, kann man die bloße Anwesenheit von verschlüsselten Informationen nicht nachweisen, geschweigedenn die enthaltenen Daten zurückgewinnen.

Die Diskussion um die Verwendung kryptographischer Verfahren kreist im wesentlichen um die Frage, welches Rechtsgut höher zu bewerten ist: das Recht des Einzelnen auf Privatsphäre oder die staatliche Kontrolle. Dieser Graben ist wahrscheinlich nicht durch Argumente zu überbrücken, da man seinen Standpunkt hier wahrscheinlich weniger aufgrund von rationalen Überlegungen wählt, als intuitiv und im Einklang mit den eigenen, verinnerlichten Werten.

4.2.3 Die Patente

Der Frage, wie lange in welchen Staaten für welche Algorithmen Patente bestehen, wird meines Erachtens zu viel Raum gegeben. Patente auf Algorithmen sind, zumindest in den Vereinigten Staaten, nichts Besonderes, sie betreffen nicht speziell die Kryptographie. Bei der Vermarktung technologisch hochwertiger Produkte fallen meist Lizenzgebühren an, die sich an Angebot und Nachfrage orientieren. Aus diesem Grund stellen Patente auf kryptographische Algorithmen kein dramatisches Problem bei ihrer Verwendung dar ([16, Abschnitt 25.5], [18, Kapitel 6]).

Ob Patente an sich eine gute Idee sind, ob sie den technischen Fortschritt eher beflügeln oder hemmen, soll hier nicht entschieden werden. Für die Befürwortung beider Standpunkte existieren sicherlich überzeugende Beispiele. Zum Trost sei bemerkt, daß im Jahre 2008 das bislang letzte wichtige, in den USA gültige Patent im Zusammenhang mit *Public Key*-Kryptographie auslaufen wird.

4.3 Schlußwort

Das Schlußwort soll Philip Zimmermann überlassen bleiben, der durch die Veröffentlichung seines Programms *pgp* viel dazu beigetragen hat, die Diskussion um die frei Verfügbarkeit der Kryptographie einer breiten Öffentlichkeit zu Bewußtsein zu bringen. Es bleibt zu hoffen, daß er recht behalten wird mit folgender Einschätzung, die er am zwölften Oktober 1993 vor einem Ausschuß des amerikanischen Repräsentantenhauses gab:

> This convergence of technology – cheap ubiquitous PCs, modems, FAX, digital phones, information superhighways, et cetera – is all part of the information revolution. Encryption is just simple arithmetic to all this digital hardware. All these devices will be using encryption. The rest of the world uses it, and they laugh at the US because we are railing against nature, trying to stop it. Trying to stop this is like trying to legislate the tides and the weather.

Literaturverzeichnis

[1] *SunOS Reference Manual – Part 1*, volume 5. SUN Microsystems.

[2] Tim Berners-Lee & R. T. Fielding & H. Nielsen. *Hypertext Transfer Protocol (HTTP 1.0)*. 1994. Internet Draft (expired, update: [79]).

[3] W. Richard Stevens. *TCP/IP illustrated*, volume 1. Addison–Wesley, Bonn u.a., 1994.

[4] Gary R. Wright & W. Richard Stevens. *TCP/IP illustrated*, volume 2. Addison–Wesley, Bonn u.a., 1994.

[5] W. Richard Stevens. *TCP/IP illustrated*, volume 3. Addison–Wesley, Bonn u.a., 1996.

[6] Craig Hunt. *TCP/IP Network Administration*. O'Reilly, Sebastopol, USA, 1994.

[7] Stephanie Teufel & Christian Sauter & Thomas Mühlherr & Kurt Bauknecht. *Computerunterstützung für die Gruppenarbeit*. Addison–Wesley, Bonn u.a., 1995.

[8] Tim Berners-Lee & L. Masinter & M. McCahill. *Uniform Resource Locators (URL)*. 1994. RFC 1738.

[9] N. Borenstein & N. Freed. *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*. 1993. RFC 1521.

[10] J. Postel. *Media Type Registration Procedure*. 1994. RFC 1590.

[11] Walter F. Tichy. RCS – A System for Version Control. *Software–Practice & Experience*, (7):637–654, July 1985.

[12] Stefan Jablonski. *Workflow-Management-Systeme – Modellierung und Architektur*. Thomson Publishing, Bonn, 1995.

[13] Othmar Kyas. *Sicherheit im Internet*. DATACOM, 50105 Bergheim, 1996.

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

[15] Friedrich L. Bauer. *Entzifferte Geheimnisse*. Springer, Berlin, Heidelberg, New York u.a., 1995.

[16] Bruce Schneier. *Applied Cryptography*. Wiley, Chichester, New York u.a., 1996.

[17] Walter Fumy & Hans Peter Rieß. *Kryptographie - Entwurf, Einsatz und Analyse symmetrischer Kryptoverfahren*. R. Oldenbourg, München, Wien, 1994.

[18] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly, Sebastopol, USA, 1995.

[19] William R. Cheswick & Steven M. Bellovin. *Firewalls and Internet Security: repelling the wily hacker*. Addison-Wesley, Bonn u.a., 1994.

[20] Deborah Russel & G. T. Gangemi Sr. *Computer Security Basics.* O'Reilly, Sebastopol, USA, 1991.

[21] D. W. Davies & W. L. Price. *Security for Computer Networks.* Wiley, Chichester, New York u.a., 2. edition, 1989.

[22] Data Encryption Standard (DES). FIPS PUB 46–1, NBS, 1994. Ersetzt Nummer 46 von 1977, veröffentlicht z.B. in [23, S. 59].

[23] D. W. Davies. *The Security of Data in Networks.* IEEE Computer Society, Los Angeles, 1981.

[24] Harry Katzan. *The Data Encryption Standard (DES).* Petrocelli Books, New York, Princeton, 1977.

[25] DES Modes of Operation. FIPS PUB 81, NBS, 1980.

[26] Xuejia Lai. *On the Design and Security of Block Ciphers*, volume 1 of *ETH Series on Information Processing.* Hartung-Gorre Verlag, Konstanz, Schweiz, 1992. Beschreibung des IDEA Algorithmus'.

[27] J. Kohl & C. Neuman. *The Kerberos Authentication Service (V5).* 1993. RFC 1510.

[28] Don Davis & Ralph Swick. Workstation services and the kerberos authentication at project athena. Technical Memorandum TM0424, Laboratory for Computer Science, MIT, Cambridge, Massachusetts, 1990.

[29] Whitfield Diffie & Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6), November 1976. Abgedruckt auch in [23].

[30] Ralph Charles Merkle. *Secrecy, Authentication and Public Key Systems.* PhD thesis, Stanford University, 1979.

[31] Kipp E. B. Hickman. *The Secure Sockets Layer (SSL) Protocol.* 1994. Internet Draft Dec. 94 (update: [66]).

[32] Thomas Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley, Chichester, New York u.a., 1990.

[33] T. H. Cormen & Ch. E. Leiserson & R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[34] Evangelos Kranakis. *Primality and Cryptography.* Teubner, Stuttgart, 1986.

[35] Ekkehard Krätzel. *Zahlentheorie.* VEB Deutscher Verlag der Wissenschaften, Berlin, 1981.

[36] B. Kaliski. *The MD2 Message Digest Algorithm.* 1992. RFC 1319 (Ersetzt RFC 1115).

[37] R. Rivest. *The MD4 Message Digest Algorithm.* 1992. RFC 1320 (Ersetzt RFC 1186).

[38] R. Rivest. *The MD5 Message Digest Algorithm.* 1992. RFC 1321.

[39] Secure Hash Standard. FIPS PUB 180, NBS, 1993.

[40] Digital Signature Standard (DSS). FIPS PUB 186, NBS, 1994.

[41] Wolfgang Schneider, editor. *SecuDE (Security Development Environment)*, volume Overview. Darmstadt, Mai 1994. Version 4.3.

[42] Wolfgang Schneider, editor. *SecuDE (Security Development Environment)*, volume 1. Darmstadt, Mai 1994. Version 4.3.

[43] Wolfgang Schneider, editor. *SecuDE (Security Development Environment)*, volume 2. Darmstadt, Mai 1994. Version 4.3.

[44] Wolfgang Schneider, editor. *SecuDE (Security Development Environment)*, volume 3. Darmstadt, Mai 1994. Version 4.3.

[45] Burton S. Kaliski Jr., editor. *Public Key Cryptography Standards, Overview.* RSA Data Security, Inc., 1993.

[46] Burton S. Kaliski Jr., editor. *Public Key Cryptography Standards, PKCS #1: RSA Encryption Standard.* RSA Data Security, Inc., 1993.

[47] Burton S. Kaliski Jr., editor. *PKCS #2.* RSA Data Security, Inc., 1993. Wurde in [46] integriert.

[48] Burton S. Kaliski Jr., editor. *Public Key Cryptography Standards, PKCS #3: Diffie-Hellman Key-Agreement Standard.* RSA Data Security, Inc., 1993.

[49] Burton S. Kaliski Jr., editor. *PKCS #4.* RSA Data Security, Inc., 1993. Wurde in [46] integriert.

[50] Burton S. Kaliski Jr., editor. *Public Key Cryptography Standards, PKCS #5: Password-Based Encryption Standard.* RSA Data Security, Inc., 1993.

[51] Burton S. Kaliski Jr., editor. *Public Key Cryptography Standards, PKCS #6: Certificate Syntax Standard.* RSA Data Security, Inc., 1993.

[52] Burton S. Kaliski Jr., editor. *Public Key Cryptography Standards, PKCS #7: Cryptographic Message Syntax Standard.* RSA Data Security, Inc., 1993.

[53] Burton S. Kaliski Jr., editor. *Public Key Cryptography Standards, PKCS #8: Private-Key Information Syntax Standard.* RSA Data Security, Inc., 1993.

[54] Burton S. Kaliski Jr., editor. *Public Key Cryptography Standards, PKCS #9: Selected Attribute Types.* RSA Data Security, Inc., 1993.

[55] Burton S. Kaliski Jr., editor. *Public Key Cryptography Standards, PKCS #10: Certification Request Syntax Standard.* RSA Data Security, Inc., 1993.

[56] Burton S. Kaliski Jr., editor. *Public Key Cryptography Standards, PKCS #11: Cryptographic Token Interface Standard.* RSA Data Security, Inc., 1993.

[57] J. Linn. *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures.* 1993. RFC 1421 (Ersetzt RFC 1113).

[58] J. Linn. *Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management.* 1993. RFC 1422 (Ersetzt RFC 1114).

[59] J. Linn. *Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes and Identifiers.* 1993. RFC 1423 (Ersetzt RFC 1115).

[60] J. Linn. *Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services.* 1993. RFC 1424 (Ersetzt RFC 1116 **nicht**).

[61] The directory–authentication framework. Recommendation X.509, CCITT, 1988.

[62] W. Richard Stevens. *Unix Network Programming.* Prentince-Hall, Englewood Cliffs, 1990.

[63] The directory–overview of concepts, models and services. Recommendation X.500, CCITT, 1988.

[64] Jonathan B. Postel. *Simple Mail Transfer Protocol (SMTP).* 1982. RFC 821.

[65] A. Schiffman E. Rescorla. *The Secure HyperText Transfer Protocol (S-HTTP).* 1996. Internet Draft (expires August 96).

[66] Kipp E. B. Hickman & Taher Elgamal. *The Secure Sockets Layer (SSL) Protocol.* 1995. Internet Draft Jun. 95 (expires Dec. 95).

[67] *Secure Electronic Transaction (SET) Specification – Business Description.* Mastercard & Visa, 1996. http://www.mastercard.com/.

[68] *Secure Electronic Transaction (SET) Specification – Technical Specifications.* Mastercard & Visa, 1996. http://www.mastercard.com/.

[69] Mark Lutz. *Programming Python.* O'Reilly, Sebastopol, USA, September 1996.

[70] Ute Schneider. Applets, schöne Applets – Java-Anwendungen selbst entwickeln. *iX Multiuser Multitasking Magazin*, (5):62–68, Mai 1996.

[71] Ari Luotonen & Kevin Altis. World-Wide Web Proxies, 1994. http://www.w3.org-/pub/WWW/Proxies/Proxies.ps.

[72] Holger Reif. Schlüsselfertig – Secure Socket Layer: Chiffrieren und Zertifizieren mit SSLeay. *iX Multiuser Multitasking Magazin*, (6):128–137, Juni 1996.

[73] Charles Petzold & Paul Yao. *Programming Windows 95.* Microsoft Press, Redmont, Washington, 1996.

[74] Prof. Dr. Andrew S. Tanenbaum. *Moderne Betriebssysteme.* Carl Hanser Verlag, München, Wien, zweite edition, 1992.

[75] M. Ben-Ari. *Grundlagen der Parallel-Programmierung.* Carl Hanser Verlag, München, Wien, 1984.

[76] Rainer Wallwitz. *Systemprogrammierung unter Windows für Win16 & Win32/NT.* Addison–Wesley, Bonn u.a., 1993.

[77] Matt Pietrek. *Windows 95 System Programming Secrets.* IDG Books Worldwide, Foster City, CA 94404, 1996.

[78] Martin Hall & Mark Towfiq & Geoff Arnold & David Treadwell & Henry Sanders. *Windows Sockets.* Jannuar 1993. Version 1.1, verfügbar z.B. auf http://www.microsoft.com.

[79] Tim Berners-Lee & R. T. Fielding & H. Nielsen. *Hypertext Transfer Protocol (HTTP 1.0).* 1995. Internet Draft (expires Feb. 1996).

Hiermit erkläre ich, Georg Bißeling, die vorliegende Arbeit selbstständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.