Sicherheitsverwaltung in einer Umgebung heterogener Datenbanken im Intranet

Diplomarbeit an der Universität Ulm
Fakultät für Informatik

durchgeführt bei der DaimlerChrysler AG
Forschung und Technologie
Abteilung FT3/EK

vorgelegt von:

Jochen Rütschlin

1. Gutachter: *PD Dr. Johannes Köbler*
2. Gutachter: *Dr. Günter Sauter*

1999

Diese Arbeit wurde nach den Regeln der neuen Rechtschreibung verfasst.

Satz: LaTeX 2_ε

Inhaltsverzeichnis

Darstellungsverzeichnis

Abkürzungsverzeichnis

AES	advanced encryption standard
AP	Anwendungsprogramm
ARPA	Advanced Research Projects Agency
ASCII	American standard code for information interchange
BSI	Bundesamt für Sicherheit in der Informationstechnologie
CA	certification authority
CAD	computer aided design
CBC	cipher block chaining
CCC	Chaos Computer Club
CERN	Centre Européen de Recherches Nucléaire[1]
CERT	computer emergency response teams
CFB	cipher feedback
CGI	common gateway interface
CPU	central processing unit
CRL	certificate revocation list
DAC	discretionary access control
DARPA	Defense Advanced Research Projects Agency
DB	Datenbank
DB2	Database 2
DB2RA	Database 2 remote access
DBMS	Datenbankmanagementsystem
DBS	Datenbanksystem
DEA	data encryption algorithm
DES	data encryption standard
ECB	electronic codebook
ER	entity relationship
EDV	elektronische Datenverarbeitung
FAQ	frequently asked questions
FIPS-PUB	Federal Information Processing Standards Publication
FK	foreign key
FTP	file transfer protocol
ggT	größter gemeinsamer Teiler
GUI	graphical user interface
HP	Hewlett Packard
HTML	hypertext markup language

[1] Wegen der unerwünschten Assoziation von „nucléaire“ mit Reaktor und militärischen Anwendungen wurde die Institution in „European Laboratory for Particle Physics“ umbenannt, das Akronym CERN blieb erhalten.

IBM	International Business Machines
IDEA	international data encryption algorithm
IETF	Internet Engineering Task Force
IP	internet protocol
IPnG	internet protocol next generation
IPv4	internet protocol version 4
IPv6	internet protocol version 6
ISOC	internet society
IT	Informationstechnologie (informationstechnisch)
JAR	Java archive
JCA	Java cryptography architecture
JDBC	Java database connectivity
JDK	Java development kit
JVM	Java virtual machine
MAC	message authentication code (*oder* mandatory access control)
MAD	militärischer Abschirmdienst
MDC	manipulation detection code *oder* modification detection code
MEntAs	Motorentwicklungsassistent
MIC	message integrity code
MLS	multilevel security
NBS	National Bureau of Standards[2]
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OFB	output feedback
OSI	open systems interconnection
OSI-RM	open systems interconnection reference model
OTP	one-time pad
PA	process accounting
PCA	policy certification authority
PGP	pretty good privacy
PIN	persönliche Identifikationsnummer
PK	primary key
PKCS	public-key cryptography standards
RFC	request for comments
RMI	remote method invocation
RSA	Rivest, Shamir, Adleman
S-HTTP	secure hypertext transfer protocol
SQL	structured query language
SSL	secure socket layer
TAN	Transaktionsnummer
TAR	tape archive
TC	trust center
TCP/IP	transmission control protocol / internet protocol
URL	universal resource locator

[2] Wurde später in NIST umbeannt.

WWW ..... World Wide Web

Kurzfassung

Seit seiner Entstehung 1969 erlebte das Internet ein beispielloses Wachstum, welches unter anderem auf den populären Internet-Dienst World Wide Web (WWW) zurückzuführen ist. Es gibt kaum eine Branche, die nicht im Internet Präsenz zeigt oder ihr lokales Netzwerk – das Intranet – ausbaut. Die Plattformunabhängigkeit, erreicht durch die nahezu auf allen Rechnersystemen verfügbaren Web-Browser, sowie der einheitliche HTML-Standard bilden die Basis für einen weit verzweigten Informationsaustausch. Selbst die Ausführung von Programmen über das Web oder die Anbindung von Datenbanken stellen schon längst kein Problem mehr dar. Neben all diesen Vorteilen ergeben sich allerdings auch gewisse Gefahren durch die Vernetzung. Unverschlüsselt übertragene Daten und die Ausführung von unbekanntem Programmcode seien dabei nur exemplarisch herausgegriffen. Einen großen Schritt in Richtung von sichereren Anwendungen wurde mit der Programmiersprache Java gemacht, bei der schon während der Konzeption einige wichtige Sicherheitsaspekte berücksichtigt wurden. Leider gibt es aber noch eine Vielzahl von Anwendungsumgebungen und Übertragungsprotokollen, die keine oder nur sehr eingeschränkte Möglichkeiten der Sicherung gegenüber unberechtigter Einsichtnahme anbieten. Für sie sind zusätzliche Vorkehrungen notwendig. Ziel dieser Arbeit ist es, eine derzeit in der Entwicklung befindliche Intranet-Anwendung mit einer Datenbankanbindung über heterogenen Datenquellen um ein Sicherheitskonzept zu erweitern, das die Vertraulichkeit, Integrität und Authentizität der verarbeiteten Daten gewährleisten soll. Zu diesem Zweck wurden neben einigen generellen Betrachtungen zu den Anforderungen an ein sicheres Computersystem und der Erläuterung kryptographischer Grundfunktionen, die Komponenten bzw. die Umgebung der vorliegenden Beispielanwendung bezüglich ihrer Sicherheit genauer beleuchtet. Dies waren im Einzelnen die verwendete Programmiersprache Java, die eingesetzten Datenbanksysteme, das Web-Umfeld sowie diverse im Internetbereich anzutreffende Übertragungsprotokolle. Desweiteren wurden Konzepte für die Beispielanwendung zur Lösung der Zugangskontrolle, einer gesicherten Übertragung und der Zugriffsregelung auf die in der Datenbank gespeicherte Informationen erörtert.

Kapitel 1

Einleitung

Seit seiner „Geburtsstunde“ im Jahre 1969 wurde das Internet zu einem Medium der Massen. Was damals unter dem Namen ARPAnet[1] mit einem Netzwerk von vier Hostrechnern begann, entwickelte sich innerhalb der letzten 30 Jahren zu einem Informationssystem ohne gleichen – im Juli 1998 waren es 36 739 000 Hosts (Hobbes, 1998).
Mitverantwortlich dafür war sicherlich die Entwicklung des World Wide Web Internet-Services, dessen Grundstein 1991 am European Laboratory for Particle Physics (CERN) gelegt wurde. Die Öffentlichkeit und die Medien registrierten (und nutzten) das Potenzial dieses Dienstes allerdings erst, als die Firma Mosaic zwei Jahre später einen grafikfähigen Browser auf den Markt brachte und damit einen regelrechten Internet-Boom auslöste.
Mit der Zeit nahm die Funktionalität der Web-Browser zu. Über sie war es fortan möglich, auf andere Internet-Dienste wie FTP oder beispielsweise News zuzugreifen. Sogar der statische Charakter von HTML-Seiten wurde durch die Einführung von JavaScript und server-seitig eingesetzten Programmen (*cgi-binaries*) aufgelockert. Mit der Bekanntmachung der Programmiersprache Java von Sun im Jahre 1995 konnten schließlich Anwendungen, sogenannte Applets, direkt aus dem Web gestartet und auf dem lokalen System ausgeführt werden.

Heute gibt es mittlerweile kaum mehr eine Branche, die nicht im Internet präsent ist und die weit verzweigten Verkehrswege für ihre Zwecke zu nutzen versucht – Electronic Commerce, Tele-Shopping, oder Online-Banking sind nur einige Schlagworte.
Aber auch immer mehr Firmen bauen ihr Intranet (ein „kleines“ und nach aussen abgeschottetes Internet innerhalb des Unternehmens) weiter aus. Die Plattformunabhängigkeit durch eine Browser-Verfügbarkeit für nahezu jedes Rechnersystem und die einheitliche Schnittstelle machen den Einsatz für das Unternehmen interessant. Hinzu kommt, dass die meisten Daten ohnehin in elektronischer Form vorliegen und durch deren Bereitstellung im Intranet ein hoher Aktualitätsgrad erreicht werden kann. Selbst die Anbindung von Datenbanken stellt mittlerweile kein Problem mehr dar.

So viele Vorteile das Internet mit sich bringt, so viele Gefahren birgt es auch in sich. Die Erstellung von Benutzerprofilen über die Auswertung von zugegriffenen Web-Seiten

[1] Das Akronym steht für Advanced Research Projects Agency. Später wurde es in DARPA (Defense ARPA) umbenannt, und bezeichnet eine US-Behörde, die 1957 gegründet wurde, um nach dem Start des Sputniks den Vorsprung der Sowjetunion auf dem Gebiet der Raumfahrttechnik aufzuholen.

ist dabei eines der kleineren Probleme. Weitaus problematischer ist die Übermittlung von vertraulichen Daten. Seien dies Kreditkartennummern für die Bestellung bei einer Online-Buchhandlung, oder die Erteilung eines Dauerauftrages bei der Hausbank, oder einer Datenbankanfrage über die neuesten Forschungsergebnisse am Arbeitsplatz – alle Daten können (in der Regel) mitgelesen werden. Das Bundesamt für Sicherheit in der Informationstechnik (BSI, 1997) vergleicht den Informationsaustausch über das Internet sogar mit dem Versand von Postkarten: jeder kann die darauf enthaltene Nachricht lesen, nur mit dem Unterschied, dass bei der Nutzung des Internets die „Karte“ durch mehr Hände geht, als bei dem gewöhnlichen Postweg.
Langsam wurde man sich der Problematik bewußt und erarbeitete Ergänzungen zu den bestehenden Protokollen bzw. entwickelte komplett neue. Eine nicht unwesentliche Rolle dabei spielten sicherlich auch die Meldungen in den Medien über Hacker und deren Einbrüche in diverse Computersysteme. 1997 wurden den Computer Emergency Response Teams (CERT) bereits 2 134 sicherheitsrelevante Vorfälle gemeldet; 1988 waren es gerade einmal 8 (Hobbes, 1998).

Ziel dieser Diplomarbeit soll es sein, eine durchgängige Sicherheitsverwaltung für das DaimlerChrysler Forschungsprojekt MEntAs zu entwickeln. Die einzelnen Komponenten dieser Intranet-Anwendung werden zunächst vorgestellt, um anschließend die Frage zu klären: Was kann an dieser Stelle für die Sicherheit der Anwendung getan werden? Dabei erfolgt eine Orientierung an der für eine sichere Kommunikation notwendigen Basissicherungsmechanismen. Für deren Realisierung in Frage kommende Verfahren werden einer Untersuchung bezüglich gebotener Sicherheit und ihrer Einsetzbarkeit in MEntAs unterzogen. Wichtig dabei ist auch die Benutzerakzeptanz, ohne die sich noch so sichere Konzepte nicht durchsetzen können.

In *Kapitel 2* werden die notwendigen Grundlagen für das Verständnis der Arbeit beschrieben. Dabei wird zuerst eine begriffliche Bestimmung und Einordnung in den Bereich der Datensicherheit und deren Bedrohungen vorgenommen. Danach wird auf die Grundlagen der Kryptologie eingegangen, was sich weitestgehend darauf beschränkt, die Terminologie zu erläutern. Eine kurze Einführung in die Konzepte von Java und ein Abschnitt über Datenbank-Middleware schließen das Kapitel.

Kapitel 3 stellt das Projekt MEntAs vor. Nach einer Beschreibung der Probleme, zu deren Lösung MEntAs eingesetzt werden soll, wird dessen Architektur erläutert, welche maßgeblich für die Inhalte der folgenden Kapitel ist.

In *Kapitel* 4 wird die Systemumgebung, in der MEntAs zum Einsatz kommt, behandelt. Nach einem Überblick über die bereits bestehende Sicherungsinfrastruktur in Abschnitt 4.1 wird in Abschnitt 4.2 auf die Sicherheitsaspekte der Programmiersprache Java eingegangen, das dann mit einem kurzen Ausblick auf die Neuerungen des JDK 1.2 schließt. Nach einer Erläuterung der Sicherheitsmechanismen in den verwendeten Datenbanken (Abschnitt 4.3) wird ein Vorschlag für die Gestaltung eines Benutzerkonzepts innerhalb der MEntAs-Datenbank gemacht, welches eine Zugriffsregelung auf Tupelebene zulässt.

Kapitel 5 widmet sich der Zugangskontrolle. Es werden die Vor- und Nachteile der drei in der Einleitung skizzierten Verfahren zur Authentifizierung diskutiert; der wissensbasierte Ansatz wird eingehender betrachtet und beschäftigt sich mit der Wahl,

Überprüfung und Speicherung von Passwörtern. In diesem Zusammenhang werden auch die sogenannten Einmal-Passwörter erläutert, welche bei dem das Kapitel abschließenden Konzeptvorschlag für die Realisierung der Zugangskontrolle in MEntAs zum Tragen kommen.

Mit Schutzmechanismen für die Übertragung beschäftigt sich *Kapitel 6*. Die in der MEntAs-Architektur verwendeten Kommunikationsprotokolle unterstützen von Haus aus keinen sicheren Datenaustausch, so dass nach Ergänzungen gesucht werden muss. Neben einigen (die Grundlagen erweiternden) Betrachtungen zu der Thematik der Verschlüsselungsalgorithmen und Nachrichtenauthentifizierung werden die wohl bekanntesten (modernen) Chiffren sowie einige im Web-Umfeld anzutreffende Sicherheitsprotokolle vorgestellt. Auch in diesem Kapitel bildet ein Vorschlag zur Umsetzung des Besprochenen in MEntAs den letzten Abschnitt.

Nach der Identifizierung und Authentifizierung in Kapitel 5, einigen Anmerkungen zur Autorisierung in Kapitel 4 und den Betrachtungen zur Geheimhaltung in Kapitel 6 wird in *Kapitel 7* schließlich noch auf den fünften der in Kapitel 2 vorgestellten Basismechanismen einer sicheren Kommunikation eingegangen – der Überwachung.

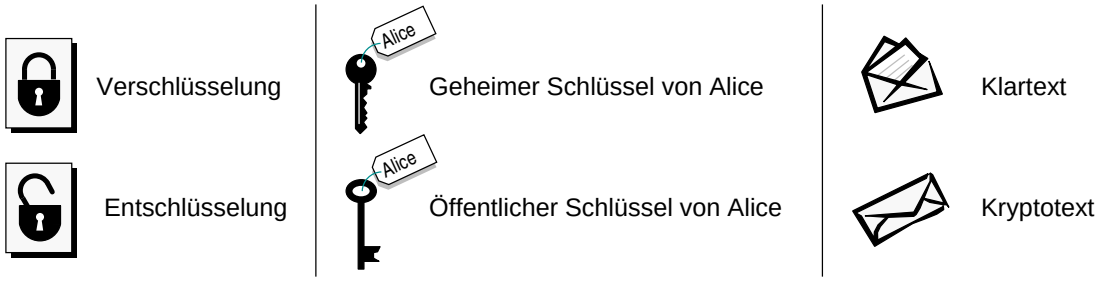
Abschließend werden in *Kapitel 8* die Ergebnisse und Erkenntnisse aus den Untersuchungen zusammenfassend dargestellt sowie ein Ausblick auf zukünftige Entwicklungen gegeben.

Anhang A enthält die Java-Klassen zu einer Beispielimplementierung des International Data Encryption Standard (IDEA).

Hinweise

Für die vereinfachende Beschreibung von Sicherheitsprotokollen haben sich in der Literatur bestimmte Personennamen eingebürgert, auf die auch in dieser Arbeit zurückgegriffen werden soll. Dabei werden den jeweiligen Namen bestimmte Funktionen zugeordnet. So sind Alice, Bob und Carol normale Beteiligte an Protokollen; Eve ist eine Lauscherin (*eavesdropper*), die durch Zuhören versucht, unrechtmäßig in den Besitz von Informationen zu gelangen (passive Angreiferin); Mallory führt bösartige (*malicious*), aktive Angriffe auf Informationen und Ressourcen durch.

Desweiteren werden folgende Symbole mit angegebenen Bedeutungen in einigen Abbildungen der Arbeit verwendet.



Kapitel 2

Grundlagen

Dieses Kapitel beschreibt grundlegende Betrachtungen zu einer sicheren Kommunikation und deren Bedrohungen. Weiterhin wird eine kurze Einführung in die Terminologie der Kryptographie und zu den Besonderheiten der Programmiersprache Java gegeben. Der letzte Abschnitt widmet sich der Datenbank-Middleware.

2.1 Sicherheit in der Informationsverarbeitung

Bevor man sich mit der Sicherheit von Informationssystemen befassen kann, muss zunächst geklärt werden, was unter einer sicheren Kommunikation zu verstehen ist, oder anders formuliert: Was bedeutet Sicherheit in der Datenverarbeitung?

Wenn man generell von *Sicherheit* spricht, schließt man das Eintreten von unerwünschten Vorfällen aus – das System im weitestgehenden Sinne funktioniert und reagiert in der erwarteten Art und Weise. Im Gegensatz dazu geht man bei einer *Absicherung* davon aus, dass bestimmte Vorfälle durchaus eintreten können, man versucht sich allerdings vor deren Folgen zu schützen; Beispiele dafür sind Stromausfälle, jegliche Art von Naturkatastrophen oder Fehler auf Speichermedien.
Eine Systemreaktion auf erwartete Art und Weise impliziert, dass zuerst das gewünschte Verhalten festgelegt werden muss. Schutz kann nicht pauschal erfolgen. Es muss geklärt werden, was für eine *Sicherheitsstrategie* verfolgt wird, d. h. gegen welche Bedrohungen man sich schützen möchte (siehe Kapitel 2.2). Erst dann kann man diesen mittels geeigneten *Sicherheitsmechanismen* begegnen.

Zwei im Zusammenhang mit Sicherheit häufig auftretende Begriffe sind *Datensicherheit* (*security*) und *Datenschutz* (*privacy*). Sie werden oft äquivalent gebraucht, ihre Bedeutung lässt sich dennoch unterscheiden. Während Datensicherheit im Allgemeinen für alle Aspekte der Sicherheit steht, wird mit ihr im Speziellen oft der Schutz gegen den Verlust von Informationen bezeichnet (dafür wäre der Begriff Datensicherung allerdings geeigneter). Allgemein steht Datenschutz für die Garantie der Vertraulichkeit; im Speziellen wird damit aber häufig der Schutz des Einzelnen gegen missbräuchliche Verarbeitung seiner personenbezogenen Daten verstanden. (Biller, 1995)

Die Daten selbst stellen noch kein zu schützendes Gut dar. Vielmehr soll eine Sicherung vor unerwünschtem Zugriff auf die in ihnen enthaltenen Informationen erfolgen.

Problematisch gestaltet sich dabei nicht der Schutz an sich, sondern vielmehr der differenzierte Zugriff auf die Daten. Hätte man nicht die Bedürfnisse, dass der Zugriff auf manche Ressourcen allen Benutzern gestattet sein soll, manche Ressourcen hingegen nur einer bestimmten Gruppe zugänglich sein sollen und wieder andere nur einem einzigen Benutzer, wäre vieles einfacher. Allerdings wäre in einem solchen System entweder allen alles erlaubt und zugänglich, was unweigerlich in Anarchie enden würde, oder aber die Ressourcen stünden nur ihrem jeweiligen Besitzer zur Verfügung, was einer nicht vorhandenen Informationsverarbeitung gleichkommen würde, da auch keinerlei Kommunikation stattfinden könnte. Erschwerend kommt hinzu, dass auf Ressourcen mit unterschiedlichen Befugnissen zugegriffen werden soll. Damit ein System diese Anforderungen erfüllen kann, bedarf es der fünf in Darstellung 2.1 gezeigten Basissicherungsmechanismen.

Darstellung 2.1: **Grundvoraussetzungen** für eine sichere Kommunikation.

2.1.1 Identifizierung

Überall wo Sicherheit zum Tragen kommt, darf es keine unbekannten Benutzer geben, mal davon abgesehen, dass es für manche Anwendungen durchaus sinnvoll sein kann, Anonymität zuzulassen, worauf aber im Folgenden nicht weiter eingegangen werden soll. Da Sicherheit maßgeblich mit individuellen Rechten auf Daten und Systemressourcen zusammenhängt (Abschnitt 2.1.3), muss zu deren Einhaltung jeder, der sich in dem Sicherheitsbereich aufhält, dem System bekannt sein. Dies bedeutet, dass vor einer Nutzung des Systems eine Anmeldungsphase erforderlich ist, in derem Verlauf der Benutzer auf eine wie auch immer geartete Form seine Identität angeben muss.

2.1.2 Authentifizierung

Laut Fremdwörterbuch bedeutet „authentifizieren“ die Echtheit bezeugen und „identifizieren“ die Echtheit einer Sache oder einer Person feststellen – im Grunde ein und dieselbe Sache, so dass auch im Allgemeinen diese beiden Begriffe äquivalent verwendet werden. Um allerdings die im EDV-Bereich übliche Vorgehensweise der Echtheitsüberprüfung hervorzuheben, nämlich Angabe einer Kennung (Identität) und anschließender Beweis der Echtheit dieser Angabe, soll hier dennoch eine Unterscheidung stattfinden.

Benutzer-Authentifizierung

Da nicht unbedingt gewährleistet ist, dass die Person auch wirklich die ist, die sie vorgibt zu sein, muss sie sich gegenüber dem System authentifizieren[1], d. h. die Richtigkeit

[1] Für das englische Wort *authentication* (= Beglaubigung, Bestätigung) werden in der deutschsprachigen Literatur die Begriffe Authentikation, Authentisierung, Authentifikation oder Authentifizierung

ihrer Angabe glaubhaft nachweisen. Grundlegend kann man drei Varianten bei der Überprüfung der Identität einer Person unterscheiden, die sich jeweils auf das Vorhandensein bestimmter Aspekte des Menschen konzentrieren: seinen Körpermerkmalen, seinem Wissen und seinem Besitz (Gegenstände).

Körpermerkmale

Zur Identifikation werden bestimmte körperspezifische Eigenschaften herangezogen, von denen man ausgeht, dass sie bei jedem Menschen verschieden sind, beispielsweise der Fingerabdruck, die Stimme oder seine Gene, um nur einige zu nennen. Systeme, die auf dieser Art der Authentifizierung basieren, werden auch als biometrische Systeme bezeichnet, da mit ihnen Eigenschaften des Körpers gemessen/erfasst werden.

Wissen

Überprüft wird ein bestimmtes Wissen, von dem man der Meinung ist, dass es nur die rechtmäßige Person erlangen kann. Aus einem Katalog werden zufällig einige Fragen ausgewählt, deren richtige Beantwortung – nach Art eines Frage-Antwort-Spiels (*challenge-response* Verfahren) – die Person als „echt“ authentifiziert. Um dies zu garantieren, muss es sich offensichtlich um die Person betreffende Fragen handeln, deren Antworten nicht aus allgemein bzw. einfach zugänglichen Quellen besorgt werden können.

Gegenstände

Eine Person weist sich durch den Besitz eines oder mehrerer Gegenstände aus. Solche Systeme sollen im Folgenden als *token-basierte* Authentifizierungssysteme bezeichnet werden (abgeleitet vom englischen Begriff *token* = Zeichen, Merkmal; Marke).

Als ein triviales Beispiel kann sicherlich der Besitz von Krone und Zepter (Reichsapfel) im Mittelalter angesehen werden, mit denen sich zum Zeichen ihres Standes Könige oder Kaiser ausgewiesen haben.

Um die Sicherheit zu erhöhen, werden diese Verfahren oft kombiniert eingesetzt. Beispielsweise bei Bankkarten, bei denen neben der Karte selbst (Gegenstand) auch das Wissen über die vierstellige Geheimzahl für eine erfolgreiche Abhebung erforderlich ist. Die Unterschrift auf der Rückseite der Karte deckt den dritten Bereich (biometrisches Merkmal) ab. Manche Karten enthalten zusätzlich ein Foto des Eigentümers. Für das Abheben von Geld werden somit immer zwei Authentizitätsmerkmale geprüft: am Automaten der Besitz der Karte und das Wissen über die geheime PIN, am Schalter der Besitz der Karte und die Übereinstimmung der Unterschrift/des Bildes.

Alle drei Verfahren haben die Gemeinsamkeit, dass das Computersystem gewisse Informationen über eine Person in einer Datenbank – dies kann im einfachsten Fall auch eine Datei sein – speichern muss, um sie bei dem Authentifizierungsvorgang mit den durch die Anmeldungsschnittstelle gelieferten Daten zu vergleichen. Diese Datenbank unterliegt natürlich besonderem Schutz, da durch Manipulationen an ihr unberechtigten Nutzern Zugang zu dem System verschafft werden kann. Dass dieser Schutz einige Probleme mit sich bringen kann, zeigt sich in Kaptiel 5.

gleichbedeutend verwendet. In dieser Arbeit soll zur Vermeidung von Verwirrungen entsprechend der im Duden verwendeten Schreibweise nur letztgenannter Begriff verwendet werden.

Nachrichten-Authentifizierung

Neben der Garantie, dass ein übertragenes Datenpaket wirklich von dem in ihm angegebenen Absender stammt (*Authentizität*), muss im Gegensatz zur Benutzer-Authentifizierung darüber hinaus sichergestellt werden, dass das Paket während seines Transportes nicht verändert wurde (*Integrität*). Realisiert werden diese Vorderungen durch kryptographische Hashverfahren (2.3.3) und meist asymmetrischen Kryptosystemen (2.3.2). Der konkrete Einsatz dieser Mechanismen wird in Abschnitt 6.4 genauer behandelt.

2.1.3 Autorisierung

Mittels Identifizierung und Authentifizierung hat sich ein rechtmäßiger Benutzer bei dem System angemeldet. In den seltensten Fällen ist es aber erwünscht, dass von diesem Zeitpunkt an jede authentifizierte Person das System uneingeschränkt nutzen darf. Es bedarf eines Autorisierungsvorganges, bei dem die individuellen Befugnisse jedes Benutzers auf Systemressourcen festgelegt werden. Dies kann auf zwei Arten erfolgen. Die eine besteht darin, die Vergabe der Rechte dem Benutzer zu überlassen; der Eigentümer entscheidet, wem er welche Zugeständnisse an seinen Daten macht. Dabei kann es sich auch um Rechte handeln, die die Weitergabe von Rechten an Dritte betreffen, sogenannte *Meta-Rechte*. In der englischsprachigen Literatur wird diese Art der Rechtevergabe als *discretionary access control* (DAC) bezeichnet (vgl. auch Amoroso, 1994, S. 254). Die eigentliche Vergabe erfolgt häufig bei der Erstellung einer Benutzerkennung, wo durch Zuteilung einer bestimmten Rolle oder Zuordnung zu einer Benutzergruppe implizit gewisse Basisbefugnisse vergeben werden. Die explizite Vergabe erfolgt im Weiteren über den Eigentümer einer Ressource (oder einer dazu berechtigten Instanz) und den entsprechenden Systembefehlen. Mit der großen Felxibilität, die dieser Ansatz dem Benutzer bietet, ist er aber auch nicht ganz unproblematisch. Da der Schutz der Daten bzw. Ressourcen in erster Linie der Willkür des Einzelnen unterliegt, kann es leicht passieren – sei es aus Vergesslichkeit oder aus mangelndem Sicherheitsbewusstsein –, dass wichtige Ressourcen nur unzureichend geschützt sind oder Rechte umgangen werden, wie das folgende Beispiel[2] zeigt: Alice vergibt Leserechte auf bestimmte Daten D an Mallory, aber nicht an Eve. Damit Mallory diese Rechte nicht in Eigenregie weitergibt, erhält sie auch keine Meta-Rechte für D. Trotz dieser Vorkehrungen kann sie das Schutzsystem umgehen. Dazu legt sie eine Kopie D' von D an (schließlich hat sie die Leserechte), auf die sie dann ihrerseits als Eigentümerin Leserechte an Eve vergibt.

Diese mehr oder weniger willkürliche Rechtevergabe soll in dem zweiten Ansatz vermieden werden, indem Befugnisse an bestimmten Daten gewissermaßen zwingend vorgeschrieben werden (*mandatory access control*, MAC[3]; im Deutschen auch oft als *regelbasierte* Modelle bezeichnet). Da nun die Schutzparameter von dem einzelnen Benutzer nicht mehr so einfach geändert werden können, ist der MAC Mechanismus weniger anfällig für Angriffe durch „Trojanische Pferde“.

[2] Das Beispiel ist dem relationalen Datenbankenbereich entlehnt, wo unter ORACLE7 eine Vergabe von Meta-Rechten möglich ist (`GRANT ... WITH GRANT OPTION`).

[3] Dieses Akronym soll in den folgenden Kapiteln nicht weiter verwendet werden, da es zu Verwechslungen mit der in der Sicherheitsliteratur häufiger auftretenden Bedeutung von *message authentication code* kommen könnte.

Lampsons Zugriffsmatrix

Der „klassische“ Vertreter, der häufig die Grundlage für DAC bildet, ist das konzeptionelle Modell von Lampson (1971). Es basiert auf einer Menge von Subjekten (*domains*) und Objekten, die in einer Tabelle (Matrix) gegenübergestellt werden (siehe Darstellung 2.2). Die Tabelleneinträge bestehen aus Rechten (beispielsweise Lesen, Schreiben oder Ausführen), die es dem Subjekt ermöglichen, eine bestimmte Operation auf das Objekt anzuwenden. Dabei müssen die beiden Mengen nicht notwendigerweise disjunkt sein: man denke an ein Programm (Objekt), das von einem Benutzer (Subjekt) ausgeführt werden darf, das aber selbst wieder – sozusagen als Subjekt – gewisse Rechte an anderen Objekten, z. B. einer Datei, hat. Wichtig ist nur, dass die Subjekte und Objekte systemweit eindeutige Namen haben.

Objekte / Subjekte	Objekt 1	Objekt 2	Objekt 3	. . .
Subjekt 1	lesen	ausführen		
Subjekt 2			lesen schreiben	
Subjekt 3		ausführen		
. . .				

Darstellung 2.2: **Lampsons Zugriffsmatrix**.

Eine naheliegende Umsetzung des Modells in einem zweidimensionalen Feld ist aus folgenden Gründen nicht empfehlenswert. Zum Einen ist die Zugriffsmatrix in der Regel nur sehr dünn besetzt, weshalb sich eine vollständige Speicherung nicht lohnt. Zum Anderen ist die Matrix wegen der großen Häufigkeit, mit der sich die Subjekte und vor allem die Objekte ändern (neues Anlegen und Löschen), einer großen Dynamik unterworfen. Daher sind üblicherweise die beiden folgenden Alternativen (auch in Mischform) anzutreffen:

Access control lists **(Zugriffskontrolllisten)**
: Bei jedem Objekt wird eine Liste von Subjekten – zusammen mit den entsprechenden Zugriffsrechten – gespeichert, die Zugriff auf das Objekt haben; man speichert also nur die (nicht-leeren) Spalten der Matrix.

Capability lists **(Befugnislisten)**
: Zu jedem Subjekt wird eine Liste von Objekten – zusammen mit den entsprechenden Zugriffsrechten – gespeichert, auf die das Subjekt Zugriff hat (Speicherung der nicht-leeren Matrixzeilen).

Die Äquivalenz der beiden Umsetzungen ist offenkundig, jedoch haben sie in der Praxis unterschiedliche Auswirkungen (Keedy, 1994). Während bei Zugriffskontrolllisten die Rechtevergabe einfacher zu handhaben ist, haben Capability-Listen den Vorteil, dass ein direkter Zugriff auf das Objekt erfolgen kann. Dies rührt hauptsächlich daher, dass bei Zugriffskontrolllisten in den meisten Fällen die Rechte bei dem Objekt gespeichert

werden; eine Änderung der Rechte ist für den Besitzer in diesem Fall leichter zu bewerkstelligen, wohingegen er für einen Zugriff auf das Objekt, dieses erst einmal finden muss (bei fremden Objekten meist in der Umgebung deren Besitzer).
Bei Capablity-Listen hingegen werden die Rechte bei den Subjekten gespeichert, wodurch sich die Rechtevergabe nicht ganz einfach gestaltet, da festgehalten werden muss, an welche Benutzer welche Befugnisse vergeben wurden. Unproblematisch ist dagegen der Zugriff auf die Objekte, der direkt über die Liste erfolgen kann.

Schutzklassensysteme

Ein häufig angeführter Vertreter von MAC sind die Schutzklassensysteme (*multilevel security*, MLS).[4] Bei ihnen wird eine total geordnete Menge von Klassen definiert, beispielsweise *allgemein zugänglich* < *zur freien Verfügung im Konzern* < *vertraulich* < *streng vertraulich*.[5] Jeder Person wird eine solche Klasse als Berechtigung (*clearance*) zugewiesen. Außerdem erhält jedes Objekt eine Einstufung (*classification*) in eine der Klassen. Der Zugriff regelt sich wie folgt: eine Person darf ein Objekt lesen, falls ihre Berechtigung größer oder gleich der Einstufung des Objektes ist; schreiben hingegen darf sie nur Objekte, deren Klassifikation größer oder gleich der eigenen Berechtigung ist. Dadurch wird der Datenfluss von einer hohen zu einer niedrigen Einstufung verhindert. Ausserdem wird nicht mehr so sehr das Datenobjekt, sondern vielmehr die Information selbst geschützt.
Neben eben genannten Vorteilen, wirft dieses einfache Modell auch neue Probleme auf. Einstufungen erfolgen oft höher als unbedingt notwendig, beispielsweise bei einem neuen Produkt, das bis zur Messepräsentation als vertraulich eingestuft wird, danach aber allgemein zugänglich ist. Auch die Integrität kann gefährdet werden, da Benutzer mit niedriger Berechtigung beliebige (unsinnige oder bewusst falsche) Daten in höher klassifizierte Objekte schreiben können.

2.1.4 Isolierung

Die alleinige Festlegung von Zugriffsrechten ist wirkungslos – ihre Einhaltung muss auch überprüft werden. Im Falle einer Verletzung von Rechten darf die diese hervorrufende Aktion nicht ausgeführt werden, die Ressource muss vor ihrem „Angreifer“ isoliert werden.
Während die Daten bei einer Verarbeitung in dem Computer selbst meist einem gewissen Schutz durch das Betriebssystem oder der Hardware unterliegen, kann die Wahrung der Rechte bei einem Transport über Netzwerke oder durch physische Weitergabe von Speichermedien, wenn überhaupt, nur sehr unzureichend gewährleistet werden. Hier kann zumindest eine Isolierung der Informationen – im Sinne von Geheimhaltung – und die Integrität der Daten durch Methoden der Kryptographie (siehe Abschnitt 2.3) erreicht werden.

[4] Eine konkrete Implementierung von MLS wurde mit Trusted Oracle7 im Datenbankbereich realisiert; nähere Informationen darüber gibt es unter `http://www.oracle.com/database/trusted/html/chapter1.html`.
[5] Diese Klassen spielen auch in den Informationsflussmodellen eine wichtige Rolle; siehe Gerhardt (1993, S. 100 f) oder auch Denning (1976).

2.1.5 Überwachung

Da es keine absolute Sicherheit gibt, ist es erforderlich, die Wirksamkeit der verwendeten Maßnahmen zu überwachen (*auditing*). Zu diesem Zweck setzt man Protokollierungsmechanismen (*logging*) ein, mit Hilfe derer Vorgänge im System rückverfolgt werden können, um somit eventuelle Sicherheitslücken zu stopfen oder die Angreifer zumindest im Nachhinein zu fassen.

2.2 Bedrohungen

Nachdem der Sicherheitsbegriff im vorhergehenden Abschnitt geklärt wurde, muss man sich, bevor man an die Ergreifung geeigneter Maßnahmen geht, Gedanken über mögliche Bedrohungen machen. Dabei erweist es sich als recht hilfreich, wenn man sich die Frage stellt

Für wen sind welche Daten aus welchem Grund interessant,
und wie kann er an sie gelangen?

Darin stecken bereits alle Merkmale einer Bedrohung: der Angreifer (das Wer), die informationstragenden Datenobjekte (das Was), die Motive für den Angriff (das Warum) und die Vorgehensweise bei der Attacke (das Wie). Nur unter Berücksichtigung aller Aspekte lässt sich wirkungsvoller Schutz praktizieren.

2.2.1 Angreifer

Weck (1984) stellt fest, dass sich prinzipiell hinter jeder Person, die im EDV-Bereich arbeitet, ein potenzieller Angreifer verbergen kann. Diese Annahme geht von einer Schätzung (Carter, Karger und Lipner, 1981)[6] aus, dass nur auf ungefähr 10% der Bevölkerung die Extreme „ehrlich“ oder „unehrlich“ zutreffen, die restlichen 80% bewegen sich dazwischen. Das bedeutet, dass dieser Personenkreis prinzipiell ehrlich ist, aber unter gewissen Umständen, kann dies auch leicht vergessen werden. Daher macht es vielmehr Sinn, sich diese Umstände genauer zu betrachten, als nach konkreten Täterprofilen zu suchen, wie zum Beispiel nach der von Carter, Karger und Lipner herausgefundenen Charakteristik: „Der „typische“ Delinquent ist männlich, 35 Jahre alt, verheiratet, hat zwei Kinder und wohnt in einer respektablen Gegend.“[6].
Generell müssen für das unehrliche Verhalten drei Faktoren zusammenkommen: Die Person benötigt die Daten bzw. Ressourcen aus irgendeinem Grund, meist um daraus persönliche, wirtschaftliche Vorteile zu ziehen (*Bedarf*). Der zweite Punkt betrifft die *Gelegenheit*, treffend formuliert durch das Sprichwort „Gelegenheit macht Diebe“. Und zuletzt setzt die Durchführung der Tat eine bestimmte moralische *Haltung* voraus.
Daher ist es zweckmäßig bei der Gestaltung eines sicheren Systems diese drei Faktoren bei einer Person nicht zusammenkommen zu lassen. Während auf den Bedarf und die Moral nur sehr begrenzt Einfluss genommen werden kann, bzw. es sich als recht schwierig erweist, diese Faktoren bei einer Person zu bewerten, ist es häufig möglich, die Gelegenheit in Hinblick auf die für eine Person interessanten Daten einzuschränken.

[6] zitiert nach Weck.

2.2.2 Informationen

Was sind die für einen Angreifer interessanten Informationen? Sicherlich nicht die aktuelle Uhrzeit oder der neueste Stand der Lieblingsaktie. Obwohl es doch einige Menschen geben soll, für die es nichts wichtigers als den Deutschen Aktien Index gibt, dreht es sich bei der Informationssicherheit nicht um Daten der eben genannten Art. Bei zu schützenden Daten handelt es sich vielmehr um Informationen, die nicht auf einfache Art und Weise beschafft werden können. Sie erhalten dadurch einen gewissen Wert, der sich sehr häufig in materiellen Größen auswirkt. Das können Forschungsdaten sein, die den technischen Vorsprung gegenüber einer konkurrierenden Firma gewährleisten, oder Gehaltsdaten, die mir eine bessere Verhandlungsbasis bei der Einstellung durch Kenntnis der Gehälter meiner zukünftigen Arbeitskollegen geben, oder aber – wofür es in der Geschichte unzählige Beispiele gibt – Truppenbewegungen, die über Sieg und Niederlage militärischer Verbände entscheiden und unter Umständen Leben retten können.
Bei der Erstellung eines Sicherheitskonzeptes muss man sich also die Frage stellen, welche Daten sind schützenswert? Können durch ihre Preisgabe Nachteile entstehen? Auch über die Dauer des Schutzes muss Klarheit herrschen. Häufig sind Informationen nur bis zu einem bestimmten Zeitpunkt als schützenswert einzustufen (die Messeneuheit gilt nur bis zu ihrer Präsentation als geheim, danach ist sie ohnehin allgemein bekannt).

2.2.3 Motive

Genauso wie man sich Gedanken über die Person eines potenziellen Angreifers in Abschnitt 2.2.1 gemacht hat, muss man sich auch Klarheit über dessen Motive verschaffen. Weck unterscheidet dabei fünf Hauptbeweggründe:

Habgier: Bei den meisten Fällen, in denen sich unrechtmäßig finanzielle Vorteile verschafft wurden, spielte dieses Motiv eine Rolle.

Finanzielle Probleme: Bei steigendem finanziellen Druck, steigt auch die Bereitschaft eines möglichen Täters, die ihm obliegenden Informationen zu Geld zu machen.

Zum Beispiel müssen beim militärischen Abschirmdienst (MAD) der Bundeswehr Angaben zu den Vermögensverhältnissen, insbesondere den Schulden, gemacht werden, um von Vornherein „gefährdete“ Personen bei deren möglichen Einsatz in sicherheitskritischen Bereichen auszuschließen.

Reiz an der Sache: Immer wieder einmal gehen Berichte von Personen oder Gruppierungen (beispielsweise dem Chaos-Computer-Club, CCC) durch die Medien, die sich in ein Computersystem gehackt haben, nur um dessen Schwachstellen aufzuzeigen. Aber auch Beweggründe wie die Steigerung des eigenen Selbstwertgefühl oder der Freude daran, ein vermeintlich sicheres System überwunden zu haben, spielen eine Rolle.

Robin-Hood-Syndrom: Dieser eher weltanschauliche Beweggrund, bei dem es zum Beispiel zu einer Umverteilung finanzieller Werte kommt, kann nur schwer in ein allgemeines Schema gepresst werden.

Rache: Die Tat, welche aus einem (vermeintlichen) Unrecht gegenüber dem Täter sein Rechtfertigung zieht, hat als primäres Ziel, Schaden zuzufügen und muss daher als das Motiv mit den schwerwiegendsten Folgen betrachtet werden.

Genausowenig wie sich diese Motive zuverlässig bei einzelnen Personen im Vorfeld ihrer Tat feststellen lassen, lässt sich Einfluss auf sie nehmen. Allerdings können mit der Kenntnis über deren Existenz bereits geeignete organisatorische Maßnahmen zur Reduzierung des Gefährdungspotenzials ergriffen werden. (Weck, 1984; S. 24)

2.2.4 Angriffsformen

Die Grundbedrohungen eines Computersystems wie unberechtigtes *Abhören* von Informationen, *Verfälschung* von Daten und Programmen, *Diebstahl* und das mutwillige Zerstören von Ressourcen (*Vandalismus*) treten in unterschiedlichen Ausprägungen auf, wobei sich prinzipiell folgende Angriffsformen unterscheiden lassen.

***Eavesdropping* (Lauschen)**
Damit wird der klassische Lauschangriff bezeichnet, der das unbemerkte Abhören und Mitschneiden von Kommunikationsströmen oder Speicherhinhalten umfasst.

***Brute-force* (rohe Gewalt)**
Diese Art von Angriffen kann man überall dort antreffen, wo Geheiminformation für die Ausführung einer Aktion benötigt werden (Anmeldung mit Passwort oder Entschlüsselung von Kryptotext). Mit dem Begriff bezeichnet man das Durchprobieren aller kombinatorischen Möglichkeiten, um an die geheime Information zu gelangen. Vorraussetzung ist natürlich eine Vergleichsmöglichkeit, mit der die Korrektheit der ausprobierten Kombination überprüft werden kann.

Spoofing
Entgegen der harmlos erscheinenden Übersetzung von *spoof* (= Ulk, Scherz, Parodie) ist solch ein Vorgehen keineswegs als harmlos einzustufen. Dabei wird dem Benutzer zum Beispiel eine Anmeldemaske angeboten (*spoof-login*), die dem Erscheinungsbild der echten bis ins Detail gleicht. Nur ihr „Innenleben“ verhält sich nicht ganz so wie das Original. Gutgläubig eingegebene Benutzerdaten (wie beispielsweise das Passwort) werden an den Angreifer weitergeleitet. Damit der rechtmäßige Benutzer von dem Angriff nichts mitbekommt – er könnte sonst sein Passwort ändern – wird häufig im Anschluss die eigentliche Funktion des Programms ausgeführt.

***Masquerading* oder *Impersonation* (Maskierung)**
Ein Angreifer gibt sich als jemand anderes aus, meist als ein berechtigter Benutzer. Da dies oft gleichbedeutend ist mit einer falschen Authentifizierung, erfolgen Maskierungen häufig durch den Diebstahl von Passwörtern.

***Trapdoors* (Falltüren)**
Damit bezeichnet man das Ausnutzen von Sicherheitslücken in einem System. So konnte man beispielsweise in UNIX-Systemen durch bewusst herbeigeführte Abstürze von bestimmten Programmen in den Besitz von Super-User-Rechten gelangen.

***Replay* (Wiederholung)**

Nachrichten werden gespeichert und später erneut abgeschickt, um damit zum Beispiel eine zeitlich begrenzte Autorisierung zu reaktivieren. Es ist dabei nicht notwendig, dass man die Nachricht auch lesen kann; Verschlüsselung alleine reicht gegen Replay-Attacken nicht aus.

***Tampering* oder *Forgery* (Verfälschung)**

Nachrichten werden im Verlauf einer wie auch immer gearteten Übertragung verfälscht, so dass der Empfänger die Daten anders erhält, als sie ursprünglich vom Sender abgeschickt wurden.

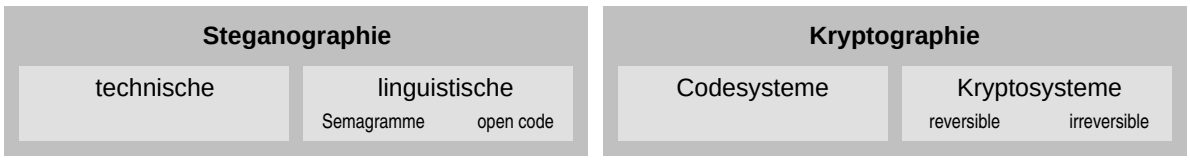
Dahingegen meint man bei dem Begriff *tamper proof* die zweite Bedeutung des englischen Verbs *to tamper* = herumhantieren. Er bezeichnet häufig eine bestimmte Art von Speicherbausteine, die durch unerwünschtes Auslesen ihrer Informationen gesichert sind; im Allgemeinen werden die Bausteine bei einem solchen Versuch zerstört.

***Denial-of-Service* (Verweigerung von Diensten)**

Diese Art von Angriffen wird häufig vergessen und richtet sich nicht so sehr gegen die Daten selbst, sondern vielmehr gegen die Verfügbarkeit von bestimmten Ressourcen. Durch bewusst herbeigeführte Überlastung des Systems oder die Deaktivierung von Diensten – zum Beispiel durch das Starten von sehr vielen Prozessen oder dem häufigen Anmelden mit falschem Passwort, so dass dessen Sperrung erfolgt – wird regulären Benutzern das Arbeiten mit bestimmten Ressourcen verwehrt.

2.3 Kryptologie

Die Isolation von Informationen, sprich ihre Geheimhaltung, kann man prinzipiell auf zwei Arten erreichen. Bei der einen verheimlicht man generell die Existenz der informationstragenden Daten – man versteckt sie gewissermaßen –, bei der anderen verbirgt man nur die Information selbst, indem man die Daten so verändert, dass sie für Eve (der unberechtigten Lauscherin) keinerlei Sinn ergeben. Methoden, die unter den erstgenannten Ansatz fallen, werden in dem Gebiet der *Steganographie* behandelt, die des zweiten in dem der *Kryptographie*. Der Sammelbegriff *Konzelationssysteme* leitet sich aus dem englischen Verb *to conceal* (= verbergen) ab und bezeichnet Methoden sowohl der Steganographie als auch der Kryptographie (siehe Darstellung 2.3).



Darstellung 2.3: Gliederung von **Konzelatiossystemen**.

In dieses Umfeld fallen auch die Begriffe der *Kryptoanalysis*, *Kryptoanalyse* und *Kryptographie*. Mit der Kryptanalysis bezeichnet man die Wissenschaft von den Methoden

der unbefugten Entschlüsselung von Daten zur Rückgewinnung der durch sie dargestellten Informationen. Die Kryptoanalyse beschäftigt sich mit der Untersuchung eines Kryptoverfahrens zum Zwecke einer Bewertung seiner kryptographischen Stärken und Schwächen. Kryptoanalyse und Kryptanalsis haben also den gleichen Untersuchungsgegenstand, lediglich ihre Intensionen sind verschieden (vgl. Horster, 1985). Einen Sammelbegriff für Kryptographie und Kryptoanalyse stellt die *Kryptologie* dar.

2.3.1 Steganographie

Steganographische Verfahren können in *linguistische* und *technische* eingeordnet werden (siehe Darstellung 2.3). Technische Methoden greifen dabei häufig auf chemische Mittel wie „unsichtbare Tinten“ zurück, mit denen die Nachricht meist zwischen die Zeilen einer unverfänglichen Mitteilung geschrieben wird. Aber auch Geheimfächer, doppelte Böden oder Mikroschriften zählen dazu.
Die linguistische Steganographie, auch als gedeckte Geheimschriften bezeichnet, kennt zwei Klassen von Verfahren: die eine besteht darin, die Daten offen und unverfänglich darzustellen (*open code*), die andere darin, Informationen durch grafische Details einer Schrift oder Zeichnung auszudrücken (beispielsweise indem kursive und aufrechte Punkte in einem Text als Morsezeichen interpretiert werden). Letztgenannte Verfahren sind auch unter dem Begriff *Semagramme* bekannt (vgl. Bauer, 1995). Heutzutage werden Informationen vorzugsweise in digitalen Multimedia-Daten wie Bildern oder Tondateien versteckt.

Da die Isolationsfunktion von steganographischen Verfahren nur auf der Geheimhaltung des Verhandenseins von Informationen an sich und des verwendeten Rekonstruktionsverfahrens beruht, sind sie wegen der im Folgenden erforderlichen Eigenschaften eines Konzelationssystems ungeeignet und sollen daher nicht weiter berücksichtigt werden.

2.3.2 Kryptographie

In der Kryptographie werden die ursprünglichen Daten mittels einer Funktion bzw. Abbildung so verändert, dass deren Ergebnis für eine unbefugte Person keinerlei Informationsgehalt mehr hat. Diese Funktion ist im Regelfall parametrisiert, wodurch trotz gleichen Ursprungsdaten unterschiedliche Funktionswerte erzeugt werden können. Trotz des mathematischen Hintergrundes werden in der Kryptographie eher die folgenden Bezeichnungen verwendet:

Ursprungsdaten	→ Klartext (*plain text*)
Funktion	→ Chiffre, Verschlüsselungsalgorithmus, Kryptofunktion, Kryptoalgorithmus, Chiffrierverfahren (*cipher*, *code*)
Funktionswert	→ Geheimtext, Kryptotext, Chiffrat (*ciphertext*)
Funktionsparameter	→ Schlüssel (*[encryption] key*)
Berechnung der Funktion	→ chiffrieren, verschlüsseln (*encrypt*)
Bildung der Umkehrfunktion	→ dechiffrieren, entschlüsseln (*decrypt*)

Das Gebiet der Kryptographie untergliedert sich in die *Codesysteme* und die *Kryptosysteme* (siehe Darstellung 2.3). Bei einem Codesystem werden Silben, (Teil-) Wörter oder Sätze durch weniger redundante Zeichenfolgen ersetzt. Darstellung 2.4 zeigt ein solches Codesystem.

var	Abstimmspruch wird getestet um ... Uhr
vis	Geben Sie zur Abstimmung Wettermeldung um ... Uhr
vwL	Befehlswelle
xdm	Schlüsselmittel und Nachrichtenunterlagen werden wegen drohender Gefahr des Feindverlustes vernichtet
xve	Bis auf weiteres Wettermeldung gemäß Funkbefehl testen
yde	Frage
sLk	Befehl
fin	beendet
eom	eigene Maschinen
sub	Suchbefehl
zrt	Es folgt Leuchtfeuer-Schaltbefehl

Darstellung 2.4: Ein Ausschnitt aus dem 1945 im Einsatz gewesenen deutschen **Luftwaffencode** (Quelle: Kahn, 1996; Seite 462).

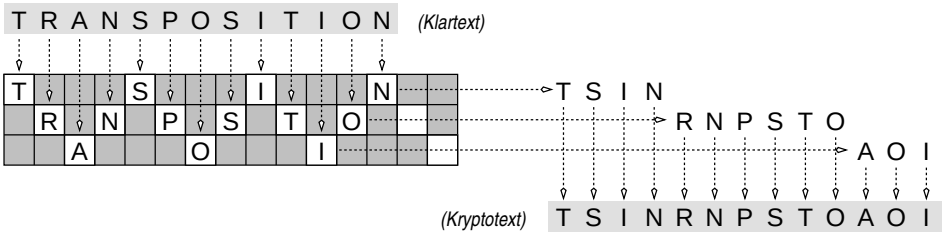
Während Codesysteme auf semantischen Spracheinheiten arbeiten, und somit nur zuvor festgelegte Sachverhalte ausdrücken können, arbeiten Kryptosysteme auf syntaktischen Einheiten. Dabei werden Einheiten fester Länge des Klartextes Einheiten fester Länge eines bestimmten Zeichenvorrates (Chiffrieralphabet) zugeordnet. Dieses Chiffrieralphabet muss nicht zwangsläufig dasselbe sein, wie das, in dem der Klartext verfasst ist.

Seit dem Artikel von Diffie und Hellman (1976) unterscheidet man Kryptosysteme in *symmetrische* und *asymmetrische* Verfahren. Symmetrische Systeme werden auch als reversible Chiffren bezeichnet oder unter dem Begriff klassische/konventionelle Kryptographie zusammengefasst (im Englischen *secret key* oder *single key* Systeme). Für die asymmetrischen Verfahren gibt es entsprechend das Synonym irreversible Chiffren oder den häufiger gebrauchten, englischsprachigen Begriff *Public-Key* Systeme.

Symmetrische Kryptosysteme

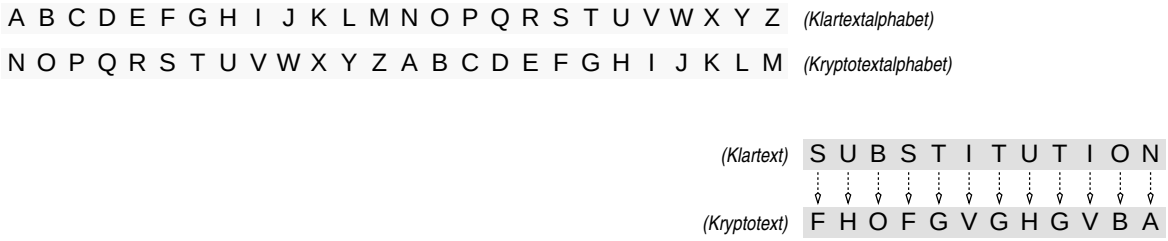
Charakteristisch für symmetrische Verschlüsselung ist, dass sie sowohl für die Verschlüsselung als auch für die Entschlüsselung den gleichen Schlüssel verwendet. Streng genommen trifft dies eigentlich nur für die sogenannten *involutorischen* Kryptosysteme zu, bei denen Ver- und Entschlüsselung mit dem gleichen Schlüssel und der gleichen Funktion durchgeführt werden. Für die meisten Chiffren wird jedoch ein *inverser* Schlüssel und/oder eine von der Verschlüsselungsfunktion verschiedene (inverse) Funktion verwendet. Da bei diesen Verfahren der inverse Schlüssel sehr leicht aus dem ursprünglichen Schlüssel berechnet werden kann, können die beiden Schlüssel als „gleich“ oder äquivalent angenommen werden.
Die klassischen Kryptosysteme setzen sich zusammen aus *Transpositions-Chiffren* und *Substitutions-Chiffren*. Bei Substitutions-Chiffren werden Zeichen oder Zeichenfolgen der zu verschlüsselnden Nachricht ersetzt, wohingegen bei den Transpositionen nur die Reihenfolge der Zeichen verändert wird; die ursprünglichen Zeichen und deren Häufigkeit

bleibt erhalten. Ein einfaches Beispiel für eine Transpositions-Chiffre zeigt Darstellung 2.5.



Darstellung 2.5: **Zick-Zack-Transposition**. Der Klartext wird im Zick-Zack in eine Matrix eingeschrieben und zeilenweise wieder ausgelesen. Die Höhe der Matrix (hier 3) bildet den Schlüssel.

Die Substitutionsverfahren lassen sich nach den Gesichtspunkten *mono-/polyalphabetisch* und *mono-/polygraphisch* gliedern: Eine Substitution nennt man monographisch (polygraphisch), wenn sie Einzelzeichen (Zeichenfolgen) ersetzt. Eine Substitution heisst monoalphabetisch, wenn jedem Zeichen (oder jeder Zeichenfolge) über einem Alphabet $\mathcal{A}$ eindeutig ein Zeichen (oder eine Zeichenfolge) über einem Alphabet $\mathcal{B}$ zugeordnet wird. Bei polyalphabetischen Substitutionen wird dagegen jedem Zeichen (oder jeder Zeichenfolge) in Abhängigkeit von seinem (ihrer) Position im Klartext ein Zeichen (oder eine Zeichenfolge) über Alphabeten $\mathcal{B}_1, \ldots, \mathcal{B}_k$ zugeordnet. Das Beispiel in Darstellung 2.6 zeigt eine recht einfache (monoalphabetische, monographische) Substitutions-Chiffre, bei der Klartext- $\mathcal{A}$ und Kryptotextalphabet $\mathcal{B}$ identisch sind.



Darstellung 2.6: **ROT13-Substitution**. Der Algorithmus ist eine zyklische Verschiebung mit dem festen Schlüssel 13 (daraus ergibt sich auch die involutorische Eigenschaft der Chiffre). Trotz der kaum vorhandenen Sicherheit, wird das Verfahren häufig im Internet verwendet, um beispielsweise News-Leser vor einem versehentlichen Mitlesen von möglicherweise beleidigenden Texten oder Rätsellösungen zu schützen.

Kryptosysteme im EDV-Bereich arbeiten meist nicht mit Alphabeten im herkömmlichen Sinne, sondern operieren auf der Bit/Byte-Ebene. Somit ist es für sie irrelevant, ob die Eingabedaten zum Beispiel aus ASCII-Zeichen oder Bilddaten bestehen.
Symmetrische Verfahren kann man in zwei Kategorien unterteilen. Die Einen verarbeiten den Klartext bit-/byteweise und werden *Stromchiffren* genannt. Die Anderen bearbeiten den Klartext in größeren Bitgruppen (oft 64 Bit) und werden daher als *Blockchiffren* bezeichnet

Blockchiffren als auch Stromchiffren können auf unterschiedliche Arten betrieben werden. Beispielsweise werden für den *Data Encryption Standard*[7] (DES) folgende Operationsmodi angeboten (FIPS PUB 74, Abschnitt 5.3): *Electronic Codebook* (ECB), *Cipher Block Chaining* (CDC), *Cipher Feedback* (CFB) und *Output Feedback* (OFB) – ECB und CBC betreiben den DES als Blockchiffre, OFB und CFB als Stromchiffre.

Der ECB-Modus zerlegt den Klartext in Blöcke von 64 Bit und chiffriert alle mit dem gleichen Schlüssel (Anm.: Der DES benötigt als Eingabe unabhängig von dem verwendeten Modus eine Blockgröße von 64 Bit). Schwachpunkt dabei ist, dass ein Angreifer einzelne Kryptotextblöcke verändern könnte, ohne dass der Empfänger dies bemerken würde. Daher wird im CBC-Modus vor der Verschlüsselung der Klartextblock mit dem vorhergehenden Kryptotextblock bitweise modulo 2 addiert; auf den ersten Klartextblock wird ein Initialisierungsvektor addiert. In den stromchiffrierten Modi wird der DES nur als Pseudozufallszahlengenerator verwendet, der bei jedem Durchlauf eine Anzahl von t ($1 \leq t \leq 64$) zufälligen Bits erzeugt, welche bitweise auf den Klartext addiert und dann übermittelt werden. Die Eingabe des DES bildet dabei ein Schieberegister, welches am Anfang mit einem Initialisierungsvektor geladen wird und bei jedem Durchlauf einer Verschiebung um t Stellen unterliegt. Abhängig von dem verwendeten Modus werden die „freien“ Stellen entweder direkt aus der DES-Ausgabe (OFB) oder aus dem Kryptotext nach erfolgter Addition mit dem Klartext (CFB) aufgefüllt.

Allen klassischen Chiffrierungen ist gemein, dass sie keinen nennenswerten Schutz bieten und es einige bekannte (teilweise sogar recht einfache) Verfahren gibt, mit denen sie gebrochen werden können[8]. Das Problem bei Substitutions-Chiffren liegt darin, dass sie die inneren Eigenschaften einer Sprache nur sehr schwer verstecken können. Man kann ihnen daher durch Mustererkennung, Häufigkeitsanalysen[9] oder speziellen sprachlich-statistischen Parametern zu Leibe rücken. Transpositions-Chiffren wurden schon seit jeher wegen ihrer geringen Sicherheit belächelt und können durch die Analyse sogenannter Kontakthäufigkeiten gebrochen werden. Ausgangspunkt hierfür ist die Verteilung von Bigrammen (Buchstabenpärchen) einer Sprache.

Obwohl die geringe Sicherheit der klassischen Kryptosysteme schon seit Jahrzehnten bekannt ist, werden sie auch heute noch, trotz verfügbarer stärkerer Algorithmen, eingesetzt. So waren beispielsweise die Zugangsdaten der T-Online-Software bis Version 2.031 nur durch eine einfache Übersetzungstabelle geschützt (Luckhardt, 1998).

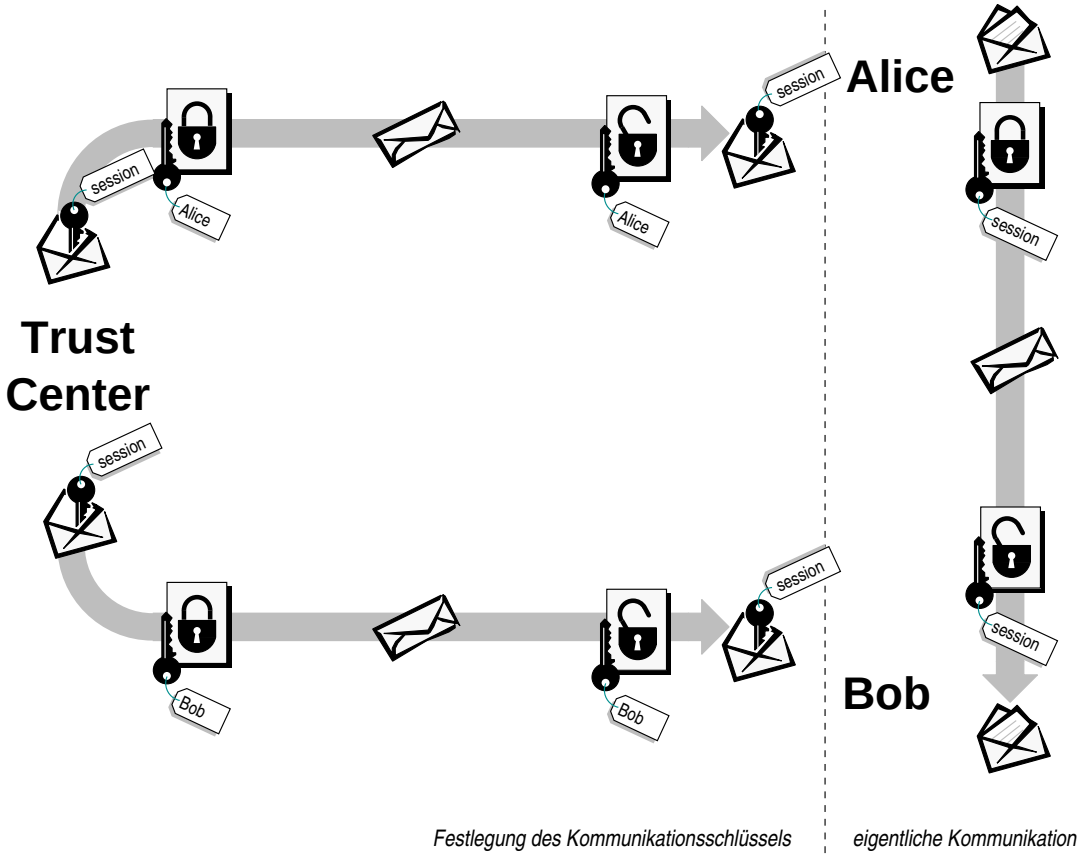
Deutlich bessere Ergebnisse liefert eine Kombination von Substitutions- und Transpositions-Chiffren, wie sie beispielsweise im DES oder dem *International Data Encryption Algorithm* (IDEA) vorkommt. Aber auch eine mehrfache Hintereinanderausführung eines Algorithmus (mit unterschiedlichen Schlüsseln) kann die Sicherheit erhöhen. So wurde nach einem erfolgreichen Angriff auf DES dieser als unsicher klassifiziert, eine Verschlüsselung mit Triple-DES hingegen, also eine dreimalige Anwendung von DES auf den Klartext, gilt bislang noch als sicher (siehe auch S. 72).

[7] Spezifikation in FIPS PUB 46-2; eine deutschsprachige Beschreibung kann man in Horster (1985, Kapitel 8, S. 114-157) nachlesen und eine Pascal-Implementierung findet sich beispielsweise in Tanenbaum (1992); vgl. auch 6.2.4.

[8] Bauer (1993) widmet knapp die Hälfte seines Buches der Kryptoanalyse.

[9] Eine recht anschauliche „Einführung“ dazu kann man in Poe (1989) nachlesen.

Die vertrauenswürdige Instanz – Schlüsselzentrale oder auch *trust center* (TC) genannt – vergibt an jeden Benutzer einen geheimen Schlüssel, der ausschließlich für die Kommunikation zwischen dem Benutzer und dem TC verwendet wird; in untenstehender Abbildung sind dies die Schlüssel mit der Beschriftung „Alice“ und „Bob“. Möchte Alice mit Bob kommunizieren ohne mit diesem direkt einen Kommunikationsschlüssel zu vereinbaren, wird dies in einer Nachricht dem TC mitgeteilt. Dieser erzeugt daraufhin einen zufälligen Sitzungschlüssel („session“), welcher an die Teilnehmer Alice und Bob jeweils mit deren TC-Kommunikationsschlüsseln chiffriert übermittelt wird. Auf diese Weise haben nur Alice und Bob (und natürlich der TC) Kenntnis über den bei der eigentlichen Kommunikation (rechter Teil der Abbildung) verwendeten Schlüssel.

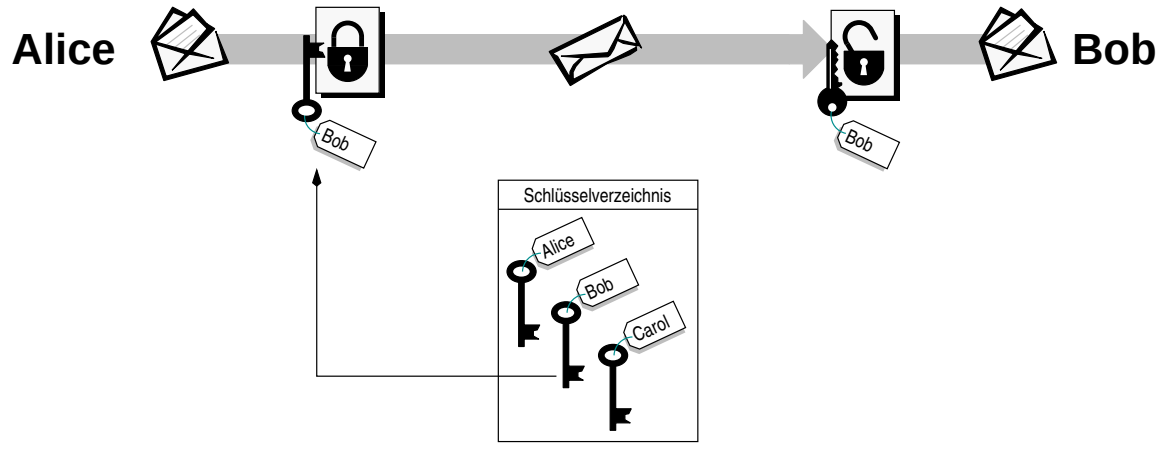


Darstellung 2.7: Funktionsweise eines **Trust Centers**.

Asymmetrische Kryptosysteme

Man stelle sich n Personen vor, die miteinander (unter Ausschluss der anderen) kommunizieren wollen. Dafür werden bei einem symmetrischen Kryptosystem $^1/_2 \cdot n \cdot (n-1)$ Schlüssel benötigt. Nicht nur, dass bei steigendem n die Anzahl der Schlüssel mit quadratischer Komplexität wächst, so gibt es auch rein praktische Probleme bei der sicheren Übermittlung der Schlüssel an die entsprechenden Partner. (Ein Reduktion der Schlüssel lässt sich durch die Einführung einer vertrauenswürdigen dritten Instanz erreichen; vgl. Darstellung 2.7.)

Aus dieser Motivation heraus beschritten Diffie und Hellman 1976 einen neuen Weg, indem sie sich von der Vorstellung lösten, dass zur Ver- und Entschlüsselung jeweils der gleiche Schlüssel verwendet werden muss. Ihr sogenanntes *Public-Key* Kryptosystem hält für jeden Benutzer (hier mit X bezeichnet) ein Schlüsselpaar bestehend aus einem öffentlichen Schlüssel k_X zur Verschlüsselung und einem privatem Schlüssel $\overline{k}_X$ zur Entschlüsselung bereit, die die Eigenschaft haben, dass eine mit k_X verschlüsselte Nachricht nur mit $\overline{k}_X$ entschlüsselt werden kann und ausserdem bei Kenntnis von k_X keinerlei Rückschlüsse auf $\overline{k}_X$ gemacht werden können. Dadurch kann k_X öffentlich bekannt gegeben werden. Für einen exklusiven Nachrichtenaustausch zwischen n Personen werden nur noch n Schlüsselpaare (also $2n$ Einzelschlüssel) benötigt; der quadratische Schlüsselaufwand wurde auf einen linearen reduziert, und die geheime Übermittlung der (öffentlichen) Schlüssel entfällt.



Darstellung 2.8: Funktionsweise eines **Public-Key** Kryptosystems. In einem für jeden zugänglichen Schlüsselverzeichnis werden die öffentlichen Schlüssel aller Teilnehmer verwaltet. Möchte Alice mit Bob in Kontakt treten, besorgt sie sich aus dem Verzeichnis dessen öffentlichen Schlüssel (k_{Bob}). Die so verschlüsselte Nachricht kann nur mit Bobs privaten Schlüssel ($\overline{k}_{\mathsf{Bob}}$) entschlüsselt werden.

Die Bedingung, dass von dem öffentlichen Schlüssel k_X keine Rückschlüsse auf den privaten $\overline{k}_X$ gemacht werden können, hängt nicht so sehr mit den Schlüsseln, sondern vielmehr mit der Beschaffenheit der asymmetrischen Verschlüsselungsfunktion zusammen, die die Eigenschaften einer *Trapdoor*-Einwegfunktion aufweisen muss (vgl. Köbler, 1999). Die Einweg-Eigenschaft garantiert, dass die Verschlüsselungsfunktion zwar effizient berechnet werden kann, ihre Umkehrung aber unter Zuhilfenahme des bekannten öffentlichen Schlüssels gar nicht oder nur sehr schwer durchführbar ist (daher auch die Bezeichnung irreversibles Kryptosystem). Und durch die Trapdoor-Eigenschaft wird gefordert, dass diese Umkehrung, also die Entschlüsselung, sehr leicht zu vollziehen ist, falls man die Geheiminformation (Falltürinformation) in Form des privaten Schlüssels besitzt.

Populärster Vertreter dieser Art von Kryptosystemen ist der RSA (siehe auch Abschnitt 6.2.6), bei dem die Trapdoor-Funktion auf dem Problem der Faktorisierung von großen Zahlen basiert: die Multiplikation zweier großer Zahlen (mehrere hundert Stellen) lässt sich effizient berechnen, die Faktorisierung dieses Produkts ist allerdings recht

schwierig, vor allem, wenn es sich bei den beiden Faktoren um gleichgroße Primzahlen handelt.

2.3.3 Kryptographische Hashverfahren

Eigentlich gehören die Einweg-Hashfunktionen, wie diese Verfahren auch genannt werden, nicht unmittelbar in das Gebiet der Konzelationssysteme, werden aber häufig in Verbindung mit ihnen verwendet/benötigt.

Eine Hashfunktion macht nichts anderes als Eingaben variabler Länge auf eine Bitfolge fester Länge (häufig im Bereich von 128 Bit) abzubilden. Dabei sollten bei der Berechnung die „charakteristischen Eigenschaften“ der Eingabe mit in das Ergebnis einfließen, so dass der Funktionswert einer Art Identifikationsnummer oder Fingerabdruck der Eingabedaten gleichkommt (oft auch als *message digest* bezeichnet). Neben oben genannter Kompressionseigenschaft, sollte sich die Funktion effizient berechnen lassen, wohingegen die Bildung ihrer Umkehrfunktion nur mit unvertretbar hohem Aufwand möglich sein sollte (Einweg-Eigenschaft).[10]

Ein einfaches, wenn auch kryptographisch irrelevantes Hashverfahren wäre zum Beispiel eine Funktion, die einen Eingabestrom entgegennimmt und ein Byte zurückliefert, das sich aus der XOR-Verknüpfung aller Bytes der Eingabe berechnet.

Das Einsatzgebiet kryptographischer Hashfunktionen liegt in der Sicherung der Integrität von Nachrichten, weswegen sie auch als *message integrity codes* (MIC) bezeichnet werden.[11] Dazu wird für eine erhaltene Nachricht die Hashfunktion berechnet und mit einem Referenzwert verglichen. Stimmen beide Werte überein, wurde die Nachricht nicht verändert. Allerdings muss sichergestellt werden, dass der Referenzwert, den der Absender der Nachricht erstellt hat, auf einem sicheren Kanal übermittelt wurde, da sonst ein Angreifer neben der Veränderung in der Nachricht auch den Hashwert anpassen könnte.
Hängt der Ausgabewert der Hashfunktion neben dem Eingabetext zusätzlich von einem symmetrischen Schlüssel ab, kann man sogar die Authentizität eines Textes garantieren (daher auch die Bezeichnung *message authentication code*, MAC). Bei der Überprüfung kann ein übereinstimmender Hashwert nur dann erzeugt werden, wenn die Nachricht nicht verändert wurde und der gleiche Schlüssel vorliegt, der auch bei der Erzeugung des *messsage digests* verwendet wurde.
Setzt man in dem Hashverfahren anstatt eines symmetrischen einen asymmetrischen Schlüssel ein, kann darüber hinaus die Urheberschaft der Nachricht gegenüber Dritten nachgewiesen werden (Unabstreitbarkeitsbedingung). So ist es nicht möglich, dass der Empfänger einer Nachricht diese verändert, die Hashfunktion mit dem ihm bekannten (symmetrischen) Schlüssel neu berechnet und als Urheber den Absender der ursprünglichen Nachricht angibt, da diese Person ebenfalls im Besitz des symmetrischen Schlüssels ist und somit die Nachricht abgeschickt haben könnte. Diese Funktionalität kommt der einer Unterschrift gleich, weswegen man in diesem Fall nicht mehr von Hashverfahren sondern von digitalen Signaturen spricht (siehe Abschnitt 6.4).

[10] Eine sehr ausführliche Einführung in kryptographische Hashverfahren findet sich in Köbler (1999).
[11] Man trifft allerdings häufiger auf das Akronym MDC, das für *manipulation detection code* oder *modification detection code* steht.

2.4 Java

Die Programmiersprache Java[12] wurde 1991 von Sun Microsystems Inc. im Rahmen eines Forschungsprojektes zur Entwicklung von Software für Verbraucherelektronikgeräte entwickelt, mit der Absicht, eine Sprache zu entwerfen, die kompakt, schnell, effizient und leicht auf unterschiedliche Hardware-Plattformen portierbar ist. Pate dafür standen einige renommierte Programmiersprachen, so dass die Ähnlichkeit von Java mit Konzepten aus C/C++, Ada und Pascal nicht zufällig ist.

Die offizielle Einführung der Java-Technologie erfolgte im Mai 1995 auf der SunWorld-Konferenz in San Francisco. Seither sorgte Java für so viel Aufsehen, dass Sun einen eigenen Geschäftsbereich (JavaSoft) für die Belange der Programmiersprache gründete. Mit ihrer rasanten Entwicklung ist kaum Schritt zu halten; nach der Veröffentlichung des *Java Development Kit* (JDK) 1.1 Anfang 1997 und der Folge eines halben Dutzend Zwischenversionen gibt es seit Dezember 1998 die Version 1.2, welche mit einigen Neuerungen, vor allem im Bereich der Sicherheit, aufwarten kann.[13]

Java lässt sich nicht wie die meisten übrigen Programmiersprachen in eine der Kategorien Übersetzer- oder Interpretersprache einordnen. Durch die Plattformunabhängigkeit bedingt stellt sie eine Hybridform dar: Der Quellcode wird durch den Java-Compiler in sogenannten *Bytecode* übersetzt. Dieser Code ähnelt Maschinencode, ist allerdings nicht auf einen speziellen Prozessor zugeschnitten. Für die Ausführung von Bytecode auf einer realen Maschine wird ein Interpreter – die *Java Virtual Mashine* (JVM) – benötigt, welcher die Umsetzung der virtuellen Maschinenbefehle auf die des realen Prozessors vornimmt. Java-Programme können also auf jedem Rechner ausgeführt werden, für dessen Betriebssystem eine JVM erhältlich ist.

Wie man auf den Web-Seiten von JavaSoft nachlesen kann, wurde Java von Anfang an auf den Gebrauch in einer heterogenen Netzwerkumgebung ausgerichtet. Dadurch wurde neben Eigenschaften wie Portabilität, Multithread-Fähigkeit[14], Robustheit und Modularität (durch Objektorientierung) auch sehr großer Wert auf Sicherheit gelegt – Sicherheit in Bezug auf missbräuchliche Verwendung der Sprache und der mit ihr geschriebenen Programme (hauptsächlich der Applets).

Java wird oft in Zusammenhang gebracht mit sogenannten *Applets*. Dabei handelt es sich um Java-Programmcode, der in Web-Seiten eingebettet von der JVM eines Browsers ausgeführt werden kann, sofern diese die zur Programmierung des Applets verwendete JDK-Version unterstützt.

Anfangs wurden Applets häufig nur zur Animation von Bildern oder für erweiterte Navigationsmöglichkeiten auf Web-Seiten verwendet. Wie aber das Projekt MEntAs zeigt (und es stellt dabei keinen Einzelfall dar), können Applets für die Bewältigung weitaus komplizierterer Aufgaben eingesetzt werden.

---

[12] Offizielle Internet-Site von Java ist `http://java.sun.com/`.

[13] Da die Entwicklung von MEntAs jedoch noch unter dem JDK 1.1 erfolgte, soll in dieser Arbeit nicht weiter auf die Neuerungen eingegangen werden. Dem interessierten Lesern können dazu die Bücher von Knudsen (1998) und Oaks (1998) empfohlen werden.

[14] Threads sind „leichtgewichtige“ Prozesse, die (im Gegensatz zu Prozessen in herkömmlichen Betriebssystemen) einen Adreßraum miteinander teilen. Damit ist es möglich, dass ein Prozess mit mehreren Kontrollflüssen mehrere Threads hat, die quasi-parallel laufen (Tanenbaum, 1994).

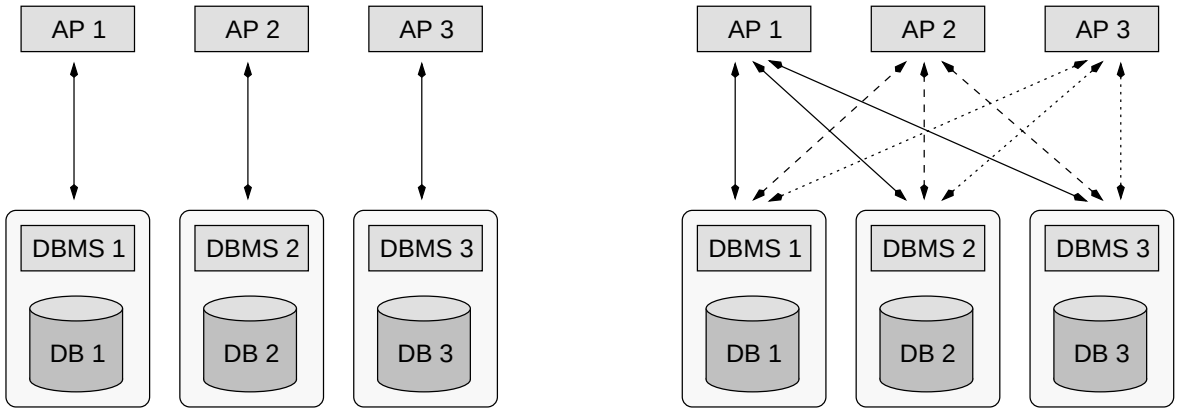
2.5 Datenbank-Middleware

Der Begriff der Datenbank-Middleware (in den folgenden Kapiteln nur noch als Middleware bezeichnet) ist relativ neu und die Definitionen in der Literatur sind teilweise noch vage. Eine prägnante Darstellung findet sich in Hergula (1998), an die sich die beiden folgenden Abschnitte anlehnen.

2.5.1 Problematik heterogener Schnittstellen

Um einem Anwendungsprogramm (AP) den Zugriff auf eine Datenbank zu ermöglichen, stellt jedes Datenbankmanagementsystem (DBMS) eine Schnittstelle zur Verfügung (beispielsweise Embedded SQL). In der Regel unterscheiden sich diese Schnittstellen zwischen den verschiedenen Datenbankherstellern, so dass der Zugriff mit steigender Zahl heterogener Datenbanken immer komplexer wird.

Greift eine Anwendung nur auf Datenbanken eines Herstellers zu, so wird einfach diese Schnittstelle implementiert (siehe Darstellung 2.9 a). Sobald die Anwendung jedoch auf DBMSe verschiedener Hersteller zugreifen möchte, müssen alle Schnittstellen berücksichtigt werden. Aus Darstellung 2.9 b ist ersichtlich, dass, sobald jede Anwendung auf jede Datenbank zugreifen soll, ein quadratischer Aufwand an Schnittstellenimplementierungen entsteht.[15]



(a) Zugriff auf *eine* Datenbank (b) Zugriff auf *alle* Datenbanken

Darstellung 2.9: **Anwendungsschnittstelle** bei Datenbanksystemen.

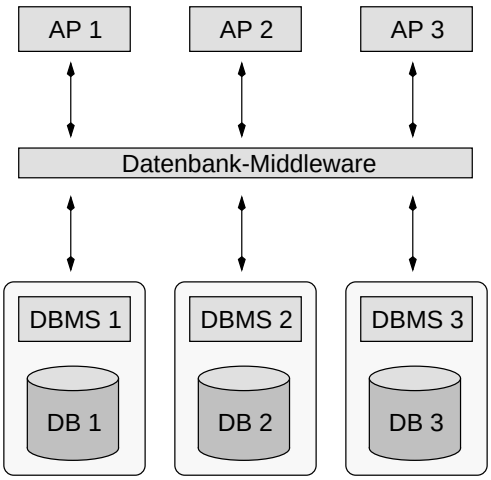
Falls in eine solche bestehende Architektur nun die Datenbank eines neuen Herstellers integriert werden soll, hat das zur Folge, dass alle betroffenen Anwendungen um eine zusätzliche Schnittstelle erweitert werden müssen. Mit Hilfe von Datenbank-Middleware Technologie kann dieser Aufwand reduziert werden.

2.5.2 Middleware

Der Begriff Middleware beschreibt im Allgemeinen eine Software-Schicht (meist in einer Client/Server-Umgebung), die das Netzwerk mit seinen Heterogenitäten sowie die ver-

[15] Man beachte die Analogie zu der geheimen Kommunikation zwischen mehreren Teilnehmern unter Verwendung eines symmetrischen Verschlüsselungssystems (siehe Abschnitt 2.3).

schiedenen Kommunikationsprotokolle vor Benutzern und Anwendungen verbirgt. Mit Hilfe einer eigenen Programmierschnittstelle zu der Middleware brauchen Entwickler die verschiedenen Netzprotokolle nicht mehr zu berücksichtigen, und können Anwendungen für verschiedene Plattformen erstellen.



Darstellung 2.10: **DB-Middleware**.

Im Bereich der Datenbanken ermöglicht der Einsatz von Middleware-Produkten das Verbergen von Unterschieden zwischen Anwendung und Datenbank. In einer Multi-Protokoll- und Multi-Hersteller-Umgebung müsste ein Programmierer die Anwendung normalerweise so schreiben, dass sie mit jedem unterstützten Protokoll und System arbeiten kann. Mit Hilfe von DB-Middleware passt der Programmierer seine Anwendungen nur an die Schnittstelle der Middleware an und lässt diese die verschiedenen Protokolle und Systeme bewältigen. Die Middleware bietet also eine einheitliche Schnittstelle nach oben an und verbirgt Heterogenitäten verschiedener DBMSe (siehe Darstellung 2.10).

Werden Datenbanken aus einer integrierten Umgebung wieder herausgenommen oder sollen neue hinzugenommen werden, so ist auch hier die Anwendung vor größeren Änderungen geschützt, da der Zugriff auf die DB-Middleware dieselbe bleibt, während die Middleware das Ansprechen der neuen bzw. geänderten Datenbankumgebung übernimmt.

2.6 Zusammenfassung

In diesem Kapitel wurden die Basisbausteine einer sicheren Informationsverarbeitung beschrieben. Dabei wurde insbesondere die Terminologie und Funktionsweise von Konzelationssystemen erläutert. In einer knappen einführenden Weise wurde desweiteren zum Verständnis der folgenden Kapitel der Begriff der Datenbank-Middleware erläutert sowie auf die Besonderheiten der Programmiersprache Java eingegangen.

Kapitel 3

MEntAs, der Motorentwicklungsassistent

In diesem Kapitel wird der Motorentwicklungsassistent vorgestellt. Nach der Darlegung der Ziele dieses Projektes, wird die Software-Architektur beschrieben, um darauf aufbauend in den folgenden Kapiteln den Einsatz von Schutzmechanismen in den einzelnen Komponenten zu untersuchen.

3.1 Ziele[1]

Seit 1997 gibt es im Bereich Forschung und Technologie von DaimlerChrysler unter Beteiligung der Abteilungen FT1/MK, FT1/MP, FT1/MS, FT1/TS und FT3/EK das Projekt Motorentwicklungsassistent, kurz MEntAs. Gegenstand dieses Projektes ist die Entwicklung eines Werkzeuges, welches die Ingenieure der Mercedes-Benz Motorentwicklung bei der Umsetzung der folgenden strategischen Ziele unterstützen soll:

- Steigerung sowohl der Reaktionsfähigkeit auf Veränderungen am Markt als auch der Innovationskraft,
- Reduzierung der Entwicklungszeit und demzufolge auch der Entwicklungskosten, sowie
- Parallelisierung von Entwicklungsarbeiten.

Derzeit ist die Motorentwicklung bei Mercedes-Benz von isolierten Software-Systemen und Datenbanken verschiedener Hersteller geprägt. Diese Systeme bestehen großteils aus Berechnungs- und Simulationswerkzeugen sowie CAD-Systemen, die einander nicht „verstehen“ und auch nicht miteinander kommunizieren können. Zudem sind die Datenbanksysteme nicht in der Lage, Informationen aus heterogenen Datenquellen abzurufen. Im Wesentlichen stellt jede in die Entwicklung involvierte Abteilung mit ihren eigenen Werkzeugen und eigenen Datenquellen eine *Informationsinsel* dar. Der Informationsaustausch zwischen den Abteilungen erfolgt meistens form- und formatlos über Telefonate oder die Erteilung von Anfrageaufträgen.

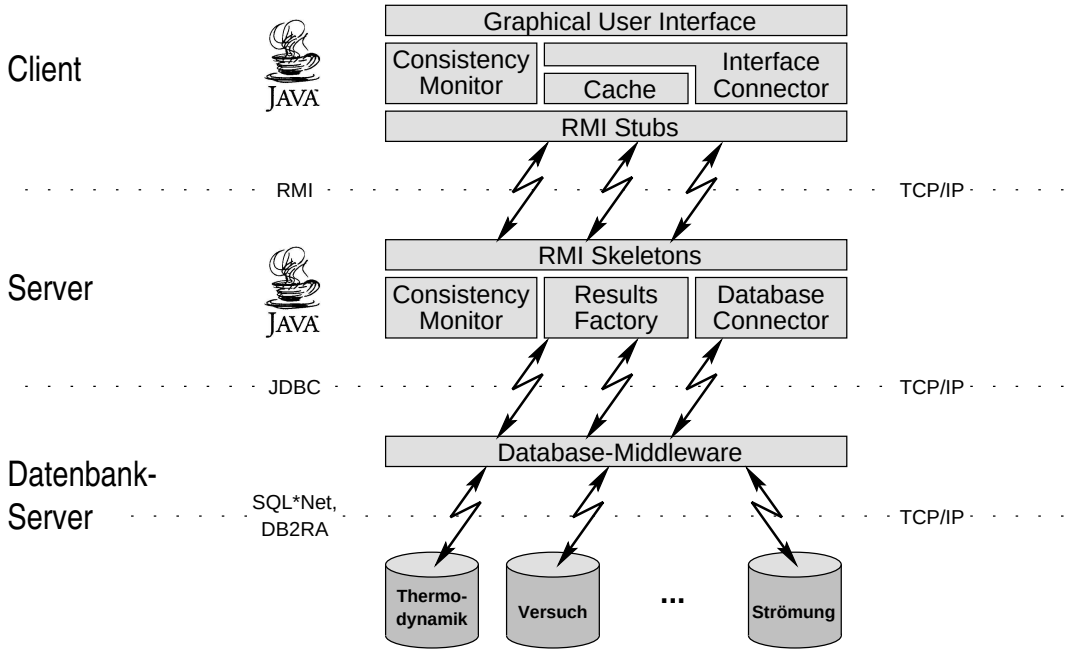
MEntAs hat sich zum Ziel gesetzt, eine vernetzte, kundenorientierte Arbeitsumgebung zur schnellen Konzeption und vergleichenden Bewertung von Motorkonzepten zu schaf-

[1] Zum Teil übernommen aus Hergula und Rezende (1998).

fen. Um dies zu erreichen, soll ein automatischer, integrierter Zugriff auf heterogene Datenbanken und Simulationswerkzeuge unter einer gemeinsamen Oberfläche zur Verfügung gestellt werden. Diese erlaubt es den Konstrukteuren, Berechnungs- und Versuchsingenieuren, eine Auswahl und Bewertung der Daten vorzunehmen, sowie die Werkzeuge – durch die Datenbankschnittstelle unterstützt – mit entsprechenden Eingabewerten zu versorgen. Dabei soll die lokale Autonomie der Bereichsdatenbanken auch nach der Integration erhalten bleiben.

3.2 Architektur

Unter Datensicherheitsaspekten betrachtet, besteht der Aufbau von MEntAs aus den folgenden drei Komponenten: dem (MEntAs-) *Client*, dem (MEntAs-) *Server* und dem *Datenbank-Server* (siehe Darstellung 3.1).



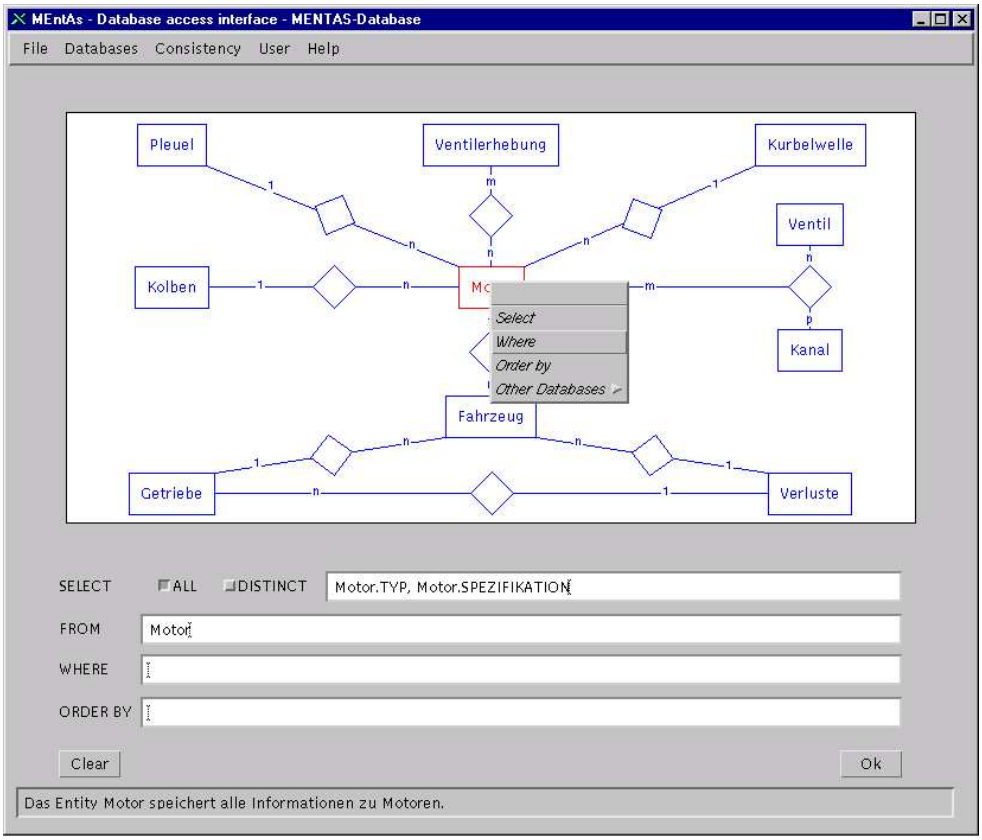
Darstellung 3.1: Die **Client/Server-Architektur** der Datenbankschnittstelle von MEntAs.

Die Implementierung der MEntAs-Software erfolgte in der Programmiersprache Java. Grund dafür war die Tatsache, dass wegen der Plattformunabhängigkeit von in Java geschriebenen Anwendungen – für die meisten Betriebssysteme gibt es eine JVM – die Client-Software auf möglichst vielen verschiedenen Rechnerplattformen eingesetzt werden kann. Ein weiterer Vorteil ist, dass durch die Einbindung der Client-Software in Form eines Applets innerhalb einer HTML-Seite beim Kunden die Vorort-Installation von neuen Versionen entfällt; bei jedem Aufruf der Web-Seite wird die neueste auf dem Server verfügbare Version heruntergeladen und ausgeführt.[2]

[2] Eingehendere Betrachtungen zu Applets erfolgen in Abschnitt 4.2.

MEntAs präsentiert sich dem Benutzer über eine grafische Benutzeroberfläche (*graphical user interface*, GUI). Über sie erfolgt die Anmeldung bei dem System[3], werden die Aufrufe der Entwicklungswerkzeuge durchgeführt und können Anfragen an die Datenbanken gestellt werden.

Der Datenaustausch zwischen GUI und dem Server erfolgt durch den *Interface Connector* des Client sowie den RMI[4]-Modulen über das Intranet. Nachdem der Client gestartet wurde, wird die GUI über den Interface Connector mit den Meta-Daten der integrierten Datenquellen (wie Tabellennamen, Attributnamen, Datentypen der Attribute und Hilfetexte zu Tabellen und Attributen) versorgt und stellt mit diesen Informationen das Datenbankschema grafisch dar. Über dieses Schema, dessen Darstellung sich an dem Entity-Relationship-Modell[5] orientiert, kann der Benutzer mit Hilfe der Maus seine Datenbankanfragen in SQL formulieren („zusammenklicken"; siehe Darstellung 3.2).



Darstellung 3.2: Über die **MEntAs-GUI** können SQL-Anfragen grafisch formuliert werden.

Sobald sich eine Anfrage über mehrere der integrierten Datenbanken erstreckt, kommt der *Consistency Monitor* (sowohl im Client als auch im Server vertreten) ins Spiel. Da die Abteilungen der Mercedes-Benz Motorentwicklung teilweise gleiche Datenbestände, aber in verschiedenen Darstellungsformen pflegen, ist ein Abgleich sowohl bei den Anfragen als auch den Ergebnismengen erforderlich.

[3] Zum Thema Zugangskontrolle siehe Kapitel 5.

[4] RMI = *remote method invocation*; ermöglicht in Java den Zugriff auf entfernte Objekte (und damit auch die Ausführung von deren Methoden) über die Versendung von Nachrichten (*messages*).

[5] Zum Entity-Relationship-Modell (ER-Modell) siehe Chen (1976).

Beispielsweise wird ein Motor durch die Attribute Typ, Spezifikation und Baumuster eindeutig identifiziert. Leider wurde nicht in allen integrierten Datenbanken diese Darstellungsform eingehalten. In einer Datenbank wurde Motortyp mit Baumuster durch einen Punkt getrennt zu einem Attribut verknüpft, in einer anderen wurde anstelle des Punktes ein Bindestrich verwendet, um den Typ mit der Spezifikation zu verknüpfen, und in einer dritten gibt es überhaupt keine Punkte oder Bindestriche und der Motortyp wurde zusammen mit der Spezifikation zu einem neuen Attribut verschmolzen, wobei Baumuster in einem völlig anderen Format dargestellt wird. (Rezende et al., 1998)
Der *Consistency Monitor* übernimmt in diesem Fall die Umsetzung zwischen den verschiedenen Darstellungen.

Der *Database Connector* des Servers stellt über die JDBC[6]-Schnittstelle eine Verbindung mit der Datenbank-Middleware her. Die *Results Factory* bereitet die über diese Verbindung erhaltenen Daten für die Darstellung im Client auf. Dabei wird nur eine vordefinierte Anzahl von Tupeln aus der Ergebnismenge der Anfrage von der Datenbank angefordert und zum Client übertragen. Während dieser mit der Anzeige der Daten beschäftigt ist, erfolgt asynchron ein *prefetch* für die nächste Teilmenge. Die übertragenen Daten werden vorübergehend in dem *Cache* des Client zwischengespeichert. Auf diese Weise ergeben sich für den Benutzer keine unnötigen Wartezeiten, während er durch das Ergebnis seiner Anfrage blättert.

Der Aufbau des *Datenbank-Servers* stellt – isoliert betrachtet – ebenfalls eine Client/Server-Architektur dar, wobei die Middleware die Rolle des Client einnimmt und die Bereichsdatenbanken (Thermodynamik, Versuch, Strömung; vgl. Darstellung 3.1) die Rolle des Servers übernehmen. Bei den integrierten Datenbanken handelt es sich um relationale Datenbanksysteme (DB2 und ORACLE).
Als kommerzielles Middleware-Produkt, über welches die Integration der heterogenen Abteilungsdatenbanken realisiert wird, wurde DataJoiner von IBM gewählt.[7] Die Integration der Bereichsdatenbanken erfolgt über Verweise in der in DataJoiner enthaltenen DB2-Datenbank. (Einzelheiten zu deren Zugangskontrolle werden in Abschnitt 4.3.5 besprochen.) Die Middleware und die DBMSe der Abteilungsdatenbanken verständigen sich über die jeweiligen Netzwerkschnittstellen der Datenbankanbieter; im Fall MEntAs sind das die Protokolle SQL*Net für die ORACLE-Datenbanken und DB2RA für die DB2-Datenbanken.

Sowohl RMI, JDBC als auch die verteilten Datenbankprotokolle SQL*Net und DB2RA basieren auf TCP/IP (*transmission control protocol/internet protocol*), den Protokollen für die Transport- bzw. Netzwerkschicht des OSI-Schichtenmodells. Die Problematik einer sicheren Übertragung auf Basis dieser Protokolle beschreibt Abschnitt 6.6.

Die Architektur in Darstellung 3.1 soll nun sowohl auf Server- als auch Client-Seite um einen *Security Controller* erweitert werden. Um dabei eine durchgängige Sicherheitskette zu realisieren, muss auch die Umgebung, in der die Komponenten ablaufen, berücksichtigt werden (Kapitel 4).

[6] JDBC = *Java database connectivity*; Schnittstelle, die die meisten Datenbankhersteller für die Integration ihres DBSe in Java-Anwendungen anbieten.
[7] Die Auswahl dieses Produktes erfolgte basierend auf den Untersuchungen von Hergula (1998).

3.3 Zusammenfassung

In diesem Kapitel wurde das Projekt MEntAs vorgestellt. Ziel dieses Projektes ist es, die Motorentwicklung bei Mercedes-Benz zu beschleunigen, indem den Ingenieuren eine vernetzte Arbeitsumgebung zur Verfügung gestellt wird. Des Weiteren wurden die Komponenten der Client/Server-Architektur von MEntAs vorgestellt, an Hand derer im Folgenden die Gewährleistung einer durchgängigen Datensicherheit untersucht werden soll.

Kapitel 4

Systemumgebung

In diesem Kapitel werden gewissermaßen die Rahmenbedingungen für den Betrieb von MEntAs betrachtet. Nach einer kurzen Einführung in die bestehende Sicherungsinfrastruktur von DaimlerChrysler, werden Sicherheitsaspekte zu der verwendeten Programmiersprache Java und den Datenbanksystemen beschrieben.

4.1 Sicherungsinfrastruktur

Da der Einsatz von MEntAs in einem bestehenden Firmennetzwerk erfolgt, sollen bereits vorhandene Sicherungsmechanismen wenn möglich verwendet werden.

Innerhalb seiner Sicherungsinfrastruktur bietet der Konzern neben einem organisatorischen Rahmen (auf den hier nicht weiter eingegangen werden soll) und der generellen Standortabsicherung durch eine Firewall auch diverse Werkzeuge zum Ver- und Entschlüsseln jeglicher Art von elektronischen Daten, zum Überprüfen der unverfälschten Übertragung mittels digitaler Signatur und zur Absicherung der Kommunikation im Client-Server-Umfeld an.
Für die Ver- und Entschlüsselung wird das Public-Key Verfahren RSA mit variabler Schlüssellänge eingesetzt. Die Versorgung mit öffentlichen Schlüsseln erfolgt über ein Konzernschlüsselverzeichnis, welches für alle Unternehmensbereiche von dem Trust Center der debis Systemhaus IT Security Services GmbH verwaltet wird. Zugriff auf das Schlüsselverzeichnis (enthält alle öffentlichen Schlüssel der an dem Verfahren teilnehmenden Benutzer) erhält man über das Intranet. Von dort kann es heruntergeladen werden und auf dem lokalen Dateisystem bei dem Endnutzer abgelegt werden.
Die Gültigkeitsdauer eines Schlüssels beträgt maximal drei Jahre. Trägermedium des geheimen Schlüsselteils ist ein Diskette (Chipkarten sind in Planung), auf der der Schlüssel vor unberechtigter Einsichtnahme durch eine PIN gesichert gespeichert ist. Die PIN wird wie bei den Bankkarten dem Benutzer postalisch gesondert zugestellt.

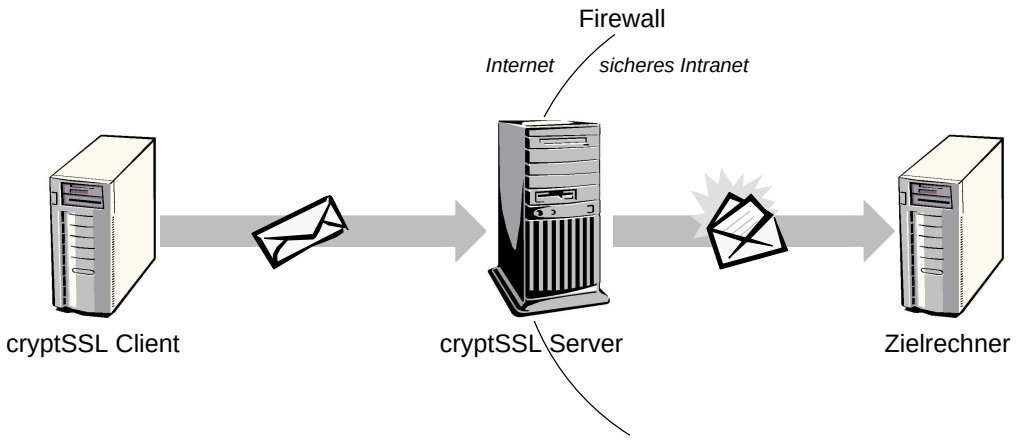
Zur Chiffrierung von Windows NT Dokumenten (beispielsweise von Microsoft-Word Dateien) werden sogenannte Plug-In-Module angeboten, die direkt in das dokumenterzeugende Programm integriert werden, wo über ein neues Menü die zusätzlichen Sicherheitsfunktionen (Ver- und Entschlüsselung, Zugriff auf das Konzernschlüsselverzeichnis) zur Verfügung stehen.

Ob und für welche Dokumente eine Verschlüsselung erfolgen soll, liegt im Verantwortungsbereich des Einzelnen. Zwar gibt es wohl eine Klassifizierung für die Geheimhaltung bestimmter Dokumente, allerdings erfolgt keine Überprüfung bzw. automatische Chiffrierung.

Für die Absicherung von Intranet-Anwendungen über Secure Socket Layer (SSL) steht ebenfalls im Intranet ein CA-Zertifikat zur Übernahme in den eigenen Browser zur Verfügung.

Eine generelle Verschlüsselung von beliebigen TCP/IP-Verbindungen kann mit dem Produkt *EADI cryptSSL* der Firma Giesecke & Devrient (welches in einer Beauftragung der DaimlerChrysler AG erstellt wurde) realisiert werden. Es basiert auf dem SSL-Protokoll (siehe Abschnitt 6.5.2) und der SSLeay-Implementierung von Eric Young.

Das Produkt besteht aus einer Client- und einer Server-Komponente. Der Server wird dabei als eine Art Zwischenstation zwischen den Client und den eigentlichen Zielrechner der Verbindung geschaltet. Der Datenaustausch zwischen Client und Server erfolgt chiffriert, der zwischen Server und Zielrechner unchiffriert. Darstellung 4.1 zeigt ein Anwendungsszenario für diese Architektur.



Darstellung 4.1: Client/Server-Modell von **cryptSSL**.

Der Client kann dabei so flexibel konfiguriert werden, dass nur bestimmte Dienste (auf bestimmten Ports; beispielsweise FTP oder telnet) zu bestimmten Rechnern verschlüsselt durchgeführt werden. Die Umleitung erfolgt transparent für die oberhalb der TCP/IP-Ebene liegenden Schichten, da bei der Installation der Software Änderungen an den relevanten Betriebssystemteilen vorgenommen werden, die beim Ansprechen eines zu verschlüsselnden Dienstes dann automatisch das Paket an den cryptSSL-Server weiterleitet. Das Paket enthält Zusatzinformationen, die dem Server mitteilen, an welchen Zielrechner das Paket nach der Entschlüsselung weiterzuleiten ist.

4.2 Schutzmechanismen in Java

Eines der Hauptziele bei der Konzeptionierung von MEntAs war es, die Software auf eine einfache Art und Weise einem möglichst großen Personenkreis zugänglich zu machen. Das bestehende Intranet und die weit verzweigten Zugriffsmöglichkeiten darauf sowie

die Möglichkeit, in Java geschriebene Anwendungen in Web-Seiten zu integrieren, bilden dafür nahezu ideale Voraussetzungen. Neben dem Aspekt der Plattformunabhängigkeit bietet Java allerdings noch weitere Vorteile gegenüber anderen Programmiersprachen – insbesondere im Bereich der Anwendungssicherheit.

In Java sind verschiedene Sicherheitsstufen integriert, die auf unterschiedlichen Ebenen der Programmausführung zum Tragen kommen. Schon durch den objektorientierten Ansatz ergeben sich diverse Schutzmechanismen während der Programmerstellung, ergänzt durch den bewußten Wegfall von Zeigern und die Bereitstellung von Klassen mit sicherheitsunterstützenden kryptographischen Funktionen. Applets laufen in einer gesicherten Umgebung, dem sogenannten Sandkasten, wodurch sich für sie einige Einschränkungen ergeben, deren Einhaltung sowohl bei ihrer Erstellung als auch zur Ausführungszeit überprüft werden. Generell wird von der JVM ausgeführter Code zur Laufzeit auf Zugriffsverletzungen hin analysiert.

Zeigerstrukturen

Die häufig gemachte Aussage, dass es in Java keine Zeiger gibt, ist genau genommen nicht richtig, denn die Objekte werden weiterhin über Zeiger referenziert. Vielmehr meint man damit, dass keine Zeiger angelegt werden können, die auf den Hauptspeicher verweisen. Dies wird dadurch erreicht, dass es keine Zeigerarithmetik gibt. Indem der direkte Zugriff auf den Speicher vollständig unterbunden wird, wird eine große und sehr unangenehme Klasse von Sicherheitsattacken völlig unterbunden (Flanagan, 1998) und nebenbei wird eine häufige Quelle von Programmierfehlern beseitigt.

Sandkastenmodell

Mit Hilfe der in Java unterstützten Sicherheit können Applets vom Web oder Intranet geladen werden, um sie anschließend risikolos auf dem Benutzerrechner abzuspielen. Dies wird gewährleistet, indem die Applet-Aktionen auf den sogenannten *Sandkasten* beschränkt werden. Dieser Sandkasten stellt den Bereich des Web-Browsers dar, der dem Applet zur Verfügung gestellt wird. Innerhalb dieses Sandkastens, kann das Applet nach Belieben agieren, es kann jedoch keine Daten (des lokalen Systems) lesen oder manipulieren, die ausserhalb des Sandkastens liegen. Die Idee des Sandkastenprinzips ist, dass unsicherer Code in einer sicheren Umgebung ablaufen kann. Sollte der Benutzer also versehentlich ein feindseliges Applet heruntergeladen haben, so kann es auf dem lokalen Rechner keinen Schaden anrichten. (Fritzinger und Mueller, 1996) Die Einhaltung der Restriktionen wird durch den SecurityManager (siehe unten) erzwungen.

***Signierte Applets** (trusted applets)*

Für manche Anwendungen ist das Sandkastenmodell zu restriktiv. Beispielsweise möchte der Benutzer aus dem – ihm wohlbekannten – Applet heraus Daten auf seiner Festplatte speichern. Durch sogenannte *Signierte Applets*, die auf einer verschlüsselten, digitalen Signatur basieren, können die Sandkasten-Einschränkungen aufgehoben werden. Software-Entwickler können mittels eines Signaturschlüssels Klassen-Dateien und Java-Archiv-Dateien signieren. Diejenige Person, die das Applet ausführt, kann mit Hilfe eines bekannten Public-Keys die korrekte Signatur des Applets – und damit dessen Authentizität und Herkunft – nachprüfen. Ist

dies der Fall, kann es als sicher eingestuft werden (*trusted applet*). Für JDK 1.1 bedeutet das, dass das Applet mit allen Rechten abläuft, über die auch lokaler, als sicher eingestufter Code verfügt.

Leider werden die von *javakey*, dem Signier-Werkzeug der Sun JDK-Distribution, erzeugten Signaturen nur von dem HotJava-Browser unterstützt, so dass für die Verwendung von *trusted applets* in anderen Browser deren eigenes Signier-Werkzeug verwendet werden muss (beispielsweise *signtool* für den Netscape-Browser).

Bytecode-Verifier und SecurityManager

Ein Problem, welches bei der Übermittlung von vorübersetzten Klassen (im Gegensatz zu Quelldateien) auftritt, ist, dass die Einhaltung der Java-Richtlinien durch den Compiler nicht sichergestellt ist; der Klassencode könnte sogar manuell erzeugt worden sein.

Ein weiteres Problem besteht darin, dass Java-Programme in der Regel aus einer Vielzahl von Klassen bestehen, deren Instanzen untereinander kommunizieren. Ein Aufruf, der zum Übersetzungszeitpunkt gültig war, kann zwischenzeitlich ungültig sein, wodurch unvorhergesehene Effekte bei der Ausführung eintreten könnten.

Aus diesem Grund enthält die JVM einen Bytecode-Verifier, der vor Ausführung des Codes die Klassen auf Typverträglichkeit (richtige Anzahl von Argumenten, Belegung von Feldern mit typkonsistenten Werten, ...) prüft.

Sind die Klassen durch den Verifier überprüft worden, so steht fest, dass die Java-Richtlinien erfüllt sind. Damit ist aber noch nicht geklärt, ob die Aktionen – auch wenn sie korrekt formuliert sind – erlaubt sind. Ein Objekt, dass bei jeder Ressourcennutzung entscheiden kann, ob der Zugriff erlaubt ist, ist der *SecurityManager*. Im Falle einer Verletzungen der Zugriffsrechte signalisiert er dies durch eine Ausnahmebedingung (*exception*), welche von dem Client-Programm weiterverarbeitet werden kann. (In Anlehnung an Rueß, Ludwig und Zapf, 1998.)

Java Cryptography Architecture (JCA)

Das Konzept der Signierung von Java-Code ist eine externe Lösung, mit der nur die Java-Klassen (vor Veränderungen) geschützt werden können, aber nicht die Daten, die die Java-Anwendung verwendet.

Die *Java Cryptography Architecture* (JCA) beschreitet einen neuen Weg, Implementierungen für Algorithmen anzubieten: Es wird keine verbindliche Implementierung mitgeliefert, sondern vielmehr werden Schnittstellen angeboten, über die zur Laufzeit auf konkrete Implementierungen zugegriffen werden kann. Das macht schon deshalb Sinn weil es eine Vielzahl von verschiedenen Algorithmen gibt, die die gleiche Aufgabe erledigen. (Rueß, Ludwig und Zapf, 1998)

Leider sind diese Schnittstellen unter dem JDK 1.1 noch recht spärlich, lediglich Message-Digest-Funktionen und Schlüsselpaargeneratoren werden unterstützt. Abhilfe wurde in der Version 1.2 geschaffen, die auch *Interfaces* für (Block-) Chiffren und ein erweitertes Schlüsselmanagement bietet.

4.3 Datenbanksicherheit

In dem Prototyp von MEntAs sind derzeit nur relationale Datenbanksysteme integriert. Im Einzelnen sind dies drei ORACLE7-, eine DB2-Datenbank und das Datenbank-Middleware Produkt DataJoiner. Da es sich bei DataJoiner um eine eigenständige DB2-Datenbank handelt, die um Komponenten zur Unterstützung der Middleware-Funktionalität erweitert wurde, treffen die im Folgenden für DB2 gemachten Aussagen auch für DataJoiner zu. Die speziellen sicherheitsrelevanten Aspekte bei den Erweiterungen zur Middleware werden in Abschnitt 4.3.5 gesondert betrachtet. Auch auf die Auditing-Möglichkeiten von Datenbanksystemen wird an anderer Stelle eingegangen (Abschnitt 7.3).

Alle DBSe operieren mit der Datenbankanfragesprache SQL (*structured query language*). Trotz dieses gemeinsamen Standards ergeben sich dennoch geringfügige Unterschiede gerade bei den Autorisierungsbefehlen `GRANT` und `REVOKE`. Zum Teil hängt dies auch mit der unterschiedlichen Realisierung der Zugangskontrolle und des Gruppenkonzeptes zusammen.

4.3.1 Gruppen und Rollen

Was man bei DB2 als Gruppe bezeichnet (in Anlehnung an die vom Betriebssystem verwendeten Benutzergruppen) entspricht bei ORACLE einer Rolle. Die unterschiedlichen Bezeichnungen sollten nicht verwirren, sie meinen dasselbe grundlegende Konzept: Benutzer mit gleichen Befugnissen können in einer Gruppe zusammengefasst werden – sie nehmen eine bestimmte Rolle ein –, so dass man nur einmal für die Gruppe Rechte an bestimmten Objekten vergeben muss, anstatt jedem Mitglied einzeln die Rechte zu erteilen. Jeder Benutzer kann in mehreren Gruppen Mitglied sein, wodurch seine Autorisierung die Vereinigung der Rechte aller Einzelgruppen ist, zusätzlich zu den Rechten, die direkt an seine Person gebunden sind.

Eine Besonderheit von ORACLE7 ist, dass man Rollen wiederum andere Rollen zuweisen kann. Mit dieser, im Gegensatz zu DB2 flexibleren Vorgehensweise ist es auch möglich, baumartige Gruppenzugehörigkeiten zu modellieren.

Ähnlich dem Benutzerverzeichnis bei einem Betriebssystem-Account befindet sich der Benutzer bei dem Datenbanksystem in seinem persönlichen Arbeitsbereich – dieser wird Schema genannt. Alle Tabellen, die nur über ihren normalen Namen, beispielsweise `dummy` angesprochen werden, werden in diesem Schema gesucht. Will man auf die Tabelle eines anderen Benutzers zugreifen, muss man vor dem Tabellennamen noch den Schemanamen, der gleichzeitig auch der Benutzername ist, angeben, zum Beispiel `alice.dummy`. So ist es möglich, dass mehrere Benutzer Tabellen (oder Sichten) mit dem gleichen Namen haben.

4.3.2 Zugangskontrolle

ORACLE7 bietet eine eigene Benutzerverwaltung an. Mit ihr können Benutzer und Rollen angelegt werden. Die Anmeldung bei dem DBMS erfolgt – wie bei den meisten Betriebssystemen – über einen Benutzernamen (*login*) und ein Passwort. Diese können völlig verschieden von dem Namen und dem Passwort des Betriebssystems sein.

Das Anlegen und Löschen von Benutzern und Rollen erfolgt über die Befehle `CREATE USER` und `CREATE ROLE` bzw. `DROP USER` und `DROP ROLE`.
Das Ändern von Benutzercharakteristika – insbesondere des Passwortes – erfolgt über den Befehl `ALTER USER`. Während man für das Ändern seines eigenen Passwortes keine weiteren Rechte benötigt, braucht man für die anderen Optionen des Befehls bzw. bei Änderungen, die einen anderen Benutzer betreffen, die `ALTER USER` Systemprivilegien (vgl. ORA-SQL, 1992, S. 4-84).

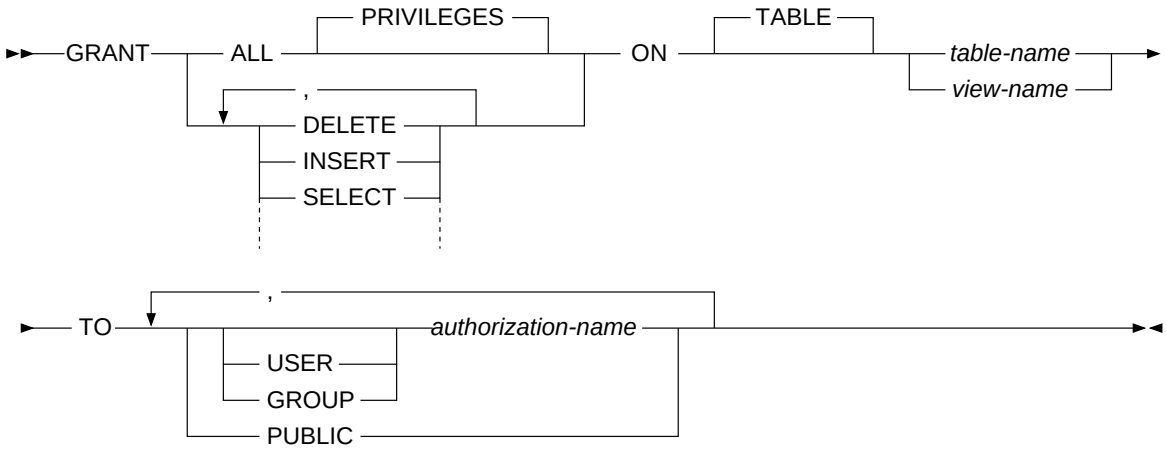
Bei DB2 hingegen erfolgt die Zugangskontrolle über das Betriebssystem. Das bedeutet, jeder Benutzer, der Zugang zu dem System hat, kann auch eine Verbindung zu dem DBS aufbauen. Inwieweit er damit dann arbeiten kann, hängt von den ihm erteilten Rechten ab.

4.3.3 Autorisierung

Die Rechte eines Benutzers an den Datenbankobjekten werden über die Befehle `GRANT` und `REVOKE` geregelt. DB2 unterscheidet im Gegensatz zu ORACLE, wo es nur zwei Arten von Privilegien gibt (System und Objekt), vier verschiedene Bereiche, deren Objekte mittels `GRANT` autorisiert werden können:

- Database Authorities
- Index Privileges
- Package Privileges
- Table or View Privileges

Das Syntaxdiagramm in Darstellung 4.2 zeigt exemplarisch den `GRANT`-Befehl zur Rechtevergabe auf Tabellen und Sichten.[1]



Darstellung 4.2: Der SQL-Befehl `GRANT` (DB2-Version)

[1] Im wesentlichen ähnelt sich die Syntax der `GRANT`-Befehle von ORACLE7 und DB2. Trotz der Dominanz der ORACLE Datenbanken soll hier die DB2-Variante vorgestellt werden, da es sich bei den ORACLE Datenbanken nur um integrierte Datenquellen handelt, an denen wegen deren geforderter Autonomie keine Änderungen von Seiten des Motorentwicklungsassistenten vorgenommen werden dürfen.

Bei `DELETE`, `INSERT` und `SELECT` handelt es sich um die Rechte zum Löschen, Einfügen und Lesen einer Tabelle (oder Sicht). Die vollständige Liste der Privilegien kann in DB2-SQL (1995, S. 428) nachgeschlagen werden. Aus dem Diagram ist ersichtlich, dass sowohl einzelnen Benutzern, als auch Gruppen (oder allen Benutzern: `PUBLIC`) Rechte erteilt werden können.
Wichtig ist, dass mit dem SQL-Befehl `GRANT` nur Rechte für Datenbankobjekte (Tabellen, Sichten, Indexe, etc.) vergeben werden können. Eine direkte Autorisierung auf der Ebene von Tupeln ist mittels `GRANT` nicht möglich. Wie man eine Ausblendung bestimmter Tabellenzeilen dennoch erreichen kann, zeigt Abschnitt 4.3.4.

Eine dennoch etwas feinere Abstufung als auf Relationenebene lässt ORACLE7 zu, wo es unter bestimmten Voraussetzungen möglich ist, Rechte auf einzelnen Tabellenspalten zu vergeben (vgl. ORA-SQL, 1992, S. 4-311 ff).
Ausserdem können dort im Gegensatz zu DB2 Rechte mit der Option vergeben werden, das erhaltene Recht an andere Benutzer weiterzugeben (`WITH GRANT OPTION`). Dabei übernimmt das DBMS selbständig die Kontrolle darüber, dass Rechte wieder entzogen werden, falls derjenige, der die Rechte an einem Objekt weitergegeben hat, selbst die Rechte für das Objekt entzogen bekommt.

Beispielsweise ist Alice Eigentümerin der Tabelle `pleuel`. Sie vergibt dafür das Leserecht an Bob mit der Option, dieses Recht weiterzugeben:

```
GRANT SELECT ON pleuel TO bob WITH GRANT OPTION;
```

Bob möchte seinerseits, dass auch Carol die Tabelle lesen kann und gibt somit das Leserecht weiter (wozu er explizit ermächtigt wurde):

```
GRANT SELECT ON alice.pleuel TO carol;
```

Kommt Alice nun zu der Erkenntnis, dass es keine so gute Idee war, Bob Einblick in die Pleueltabelle zu gewähren, kann sie ihm das Recht dazu wieder entziehen:

```
REVOKE SELECT ON pleuel FROM bob;
```

Das DBMS entzieht dann automatisch auch Carol das Leserecht, welches sie ja von Bob erhalten hat.

ORACLE ist bei der Rechteverwaltung so flexibel, dass es unterscheiden kann, von wem welche Rechte an welchem Objekt erteilt wurden, auch wenn die Autorisierung von mehreren Seiten her erfolgt ist.

Angenommen Alice hat wie in vorherigem Beispiel das Recht zum Lesen an ihrer Tabelle `pleuel` nicht nur an Bob (diesem wieder mit der Option der Rechteweitergabe), sondern auch an Carol vergegeben.
Bob kommt nun durch eigenen Entschluss zu der Überzeugung, dass auch Carol einen Blick in die Daten werfen sollte und gibt seinerseits das Leserecht an Carol weiter, ohne zu wissen, dass sie es bereits von Alice erhalten hat. Entzieht nun Alice Carol das (direkte) Leserecht wieder, hat Carol immer noch Zugriff auf die Tabelle, da sie rechtmäßig von Bob dazu ermächtigt wurde. Eine Tatsache, die für die Eigentümerin Alice nicht unbedingt offensichtlich ist, nachdem sie doch gerade Carol das Recht entzogen hat.[2]

[2] Für Alice besteht die Möglichkeit mittels einigen Systemtabellen herauszufinden, wer welche Rechte an ihren Objekten besitzt.

4.3.4 Sichtenkonzept[3]

Eine Sicht (*view*) ist ein Datenbankobjekt, das (mit Einschränkungen) die gleiche Funktionalität wie eine Tabelle hat. Es lassen sich auf ihr die gleichen Operationen ausführen, wie auf einer Tabelle.
Die Sicht ist eine auf den Benutzer zugeschnitte Darstellung von Daten aus einer oder mehrerer Basistabellen. Basistabellen können dabei entweder Tabellen oder auch selbst Sichten sein.
Eine Sicht enthält selbst keine Daten, vielmehr kann man sie sich als eine gespeicherte Anfrage vorstellen, die auf den Basistabellen ausgeführt wird.

Dadurch ist es neben einigen anderen Zwecken, wie beispielsweise der Umbenennung von Spalten oder einer eingeführten Transparenz bei der Verknüpfung von Relationen, auch möglich, den Zugriff auf einzelne Spalten oder Zeilen zu beschränken, indem man durch die Sicht einfach nur die „freigegebenen“ Informationen selektiert oder unerwünschte Spalten wegprojeziert.
Beispielsweise bestehe eine Tabelle `personal` aus den Attributen *Personalnummer*, *Name*, *Vorname*, *Abteilung* und *Gehalt*. Alice soll Zugriff auf diese Tabelle erhalten, allerdings darf sie die empfindlichen Gehaltsdaten nicht sehen. Nur mit Hilfe des `GRANT`-Befehls wäre ein solcher Zugriff nicht zu realisieren. Anders mit einer Sicht:

```
CREATE VIEW personal_alice AS
  SELECT Personalnummer, Name, Vorname, Abteilung
    FROM personal;
```

Man kann nun für Alice den Zugriff auf die Basistabelle `personal` sperren und ihr Rechte auf `personal_alice` einräumen. Damit kann sie nur die für sie freigegebenen Informationen einsehen.

Zur Sperrung einzelner Zeilen nehmen wir als Beispiel Bob, der nur Daten von Mitarbeitern der Abteilung FT3/EK bearbeiten darf. Die für ihn autorisierte Sicht würde dann wie folgt definiert werden:

```
CREATE VIEW personal_bob AS
  SELECT *
    FROM personal
   WHERE Abteilung='FT3/EK';
```

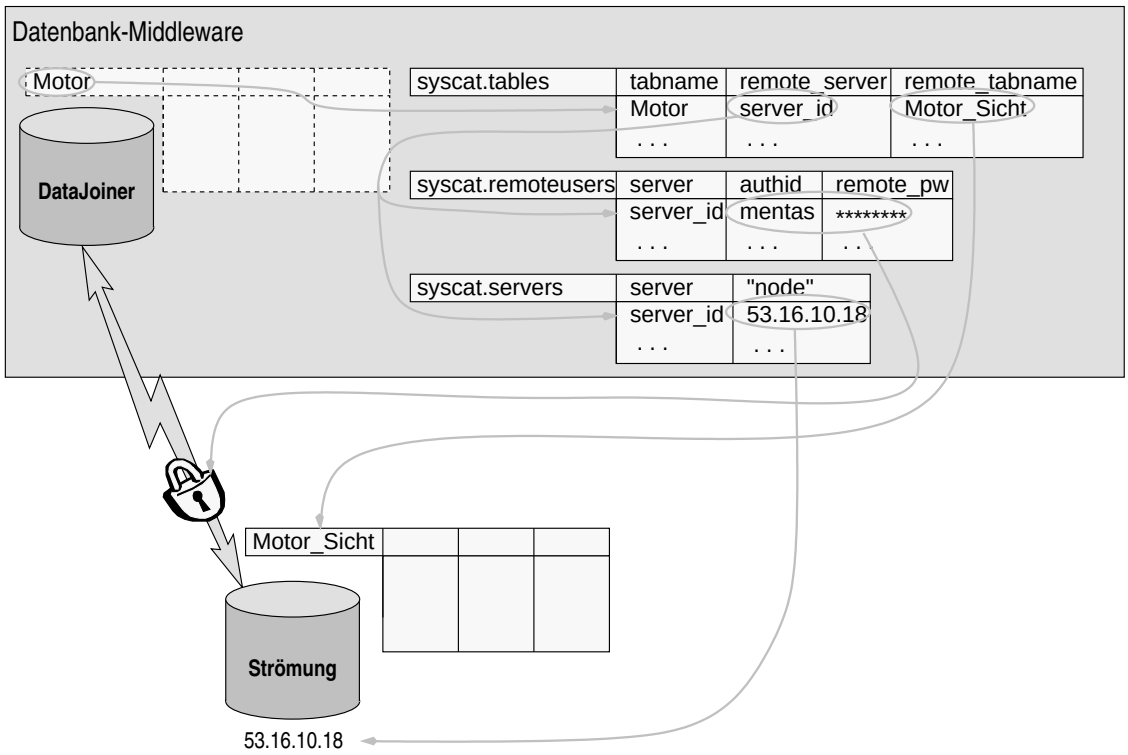
Natürlich lassen sich die beiden obigen Benutzerssichten auch zu einer *view* verknüpfen.

4.3.5 Datenbank-Middleware

Der Zugang zur Middleware erfolgt durch die integrierte DB2 Datenbank, also über das Betriebssystem. Zugriff auf entfernte Tabellen oder Sichten erhält man über sogenannte *nicknames*, einem Objekt, das von der Middleware her wie eine gewöhnliche Relation angesprochen werden kann. Dadurch gestaltet sich der Zugriff für den Benutzer nach der Integration weitestgehend transparent. Er hat die Möglichkeit Sichten auf der Tabelle zu definieren und Rechte mit dem `GRANT`-Befehl zu vergeben.

[3] In Anlehnung an ORACLE (1993).

Für die Integration der Quelldaten müssen einige Vorarbeiten geleistet werden, bei denen im Wesentlichen die Server-Adresse der Datenbank und der Zugang (Kennung und Passwort) festgelegt wird. Dies geschieht mit den Befehlen `CREATE SERVER MAPPING`, `CREATE USER MAPPING` und `CREATE NICKNAME`.
Mit `CREATE SERVER MAPPING` wird vereinfacht gesagt eine Zuordnung zwischen einem selbst definierten, DB-internen Bezeichner für den Server (der zu integrierenden Datenbank) und dessen TCP/IP-Adresse vorgenommen. Dieser Bezeichner wird aufgegriffen, wenn mittels `CREATE USER MAPPING` festgelegt wird, mit welcher Benutzerkennung und welchem Passwort auf den Server zugegriffen wird.[4]
Schließlich wird mit `CREATE NICKNAME` eine Art virtuelle Tabelle angelegt, deren Schema dem der zu integrierenden gleicht. Darstellung 4.3 verdeutlicht den Zugriff auf eine solche Tabelle.



Darstellung 4.3: **Zugriff der Middleware** (vgl. Abbildung 3.1 auf Seite 26, Datenbank-Server). Der Zugriff auf die DB2 Tabelle `Motor_Sicht` erfolgt von DataJoiner aus über den *nickname* `Motor`. Dazu holt sich das DBMS aus der Systemtabelle `syscat.tables` die Informationen für das referenzierte Objekt. Über die Server-Kennung kann aus den Daten in `syscat.servers` eine TCP/IP-Verbindung zu der entfernten Datenbank aufgebaut werden, bei der sich unter Zuhilfenahme des entsprechenden Eintrags in `syscat.remoteusers` DataJoiner anmelden kann.

Wie in der Darstellung angedeutet, wird die Passwortinformation für den Zugang zur Quelldatenbank beim Auslesen maskiert dargestellt. Die Information selbst wird verschlüsselt auf der Platte abgelegt, so dass selbst eine Untersuchung der Datenbankdateien mit einem Hex-Editor das Passwort nicht im Klartext verrät.

[4] Die Zuordnungen werden in Systemtabellen der Middleware abgelegt: Server und Adresse in `syscat.servers`, Server und *User-Mapping* in `syscat.remoteusers`.

4.4 Autorisierung und Zugriffskontrolle in MEntAs

Zu MEntAs soll nicht jeder Systembenutzer Zugang haben. Daher muss eine Zugangsregelung für autorisierte Benutzer realisiert werden. (Nähres dazu in Abschnitt 5.4 ab Seite 64.)

Die Benutzerverwaltung ist so zu entwerfen, dass ein Gruppenkonzept unterstützt wird: Eine Gruppe fasst mehrere Benutzer zusammen, denen bestimmte Befugnisse gemeinsam sind. Dabei soll die Mitgliedschaft in mehreren Gruppen möglich sein. Die gesamten Befugnisse eines Benutzers ergeben sich aus der Vereinigung aller Einzelbefugnisse, die aus der Mitgliedschaft in einer bestimmten Gruppe gewährt werden.

Wenn von Befugnissen die Rede ist, sind Rechte wie Lesen, Einfügen, Ändern und Löschen gemeint. Befugnisse beziehen sich auf einzelne Motorkonzepte. Ein Motorkonzept enthält die charakteristischen Werte eines (Versuch-) Motors. Solche Werte sind beispielsweise Typ, Bauart, Leistung, Kolbenhub oder Ventildurchmesser. Diese Attribute sind auf mehrere Tabellen (Motor, Kolben, Ventil, etc.) verteilt, welche mittels Primär- und Fremdschlüsselattributen zueinander in Beziehung gesetzt werden können. Auf diese Weise wird die redundante Speicherung von Motorbestandteilen – ein Ventil kann in verschiedenen Motoren vorkommen – vermieden. Ein bestimmtes Motorkonzept ergibt sich also aus der Verknüpfung (*join*) aller Konzepttabellen, wobei die Motortabelle gewissermaßen als Einstiegspunkt gilt, da in ihr der Typ und die Bezeichnung sowie die Verweise auf der im Motor verwendeten Teile (Ventil, Kolben, usw.) gespeichert sind.

Motorkonzepte werden von den Ingenieuren entsprechend den Anforderungen für einen neu zu entwickelnden Motor erstellt, beispielsweise „Erstelle ein Konzept für einen Dreizylindermotor mit einem Benzinverbrauch von nicht mehr als 4 Litern auf 100 km“. Ein Konzept wird vorerst von dem Ingeneur in einem lokalen Bereich, auf den nur er Zugriff hat, bearbeitet. Er hat die Möglichkeit, nach entsprechendem Fortschritt seines Entwurfs, das Konzept freizugeben, wodurch es der bei der Freigabe zugeordneten Gruppe zugänglich gemacht wird. Ein Konzept hat somit genau einen Eigentümer und gehört genau einer Gruppe an.

Bereits freigegebene Konzepte dürfen von allen Gruppenmitgliedern eingesehen und bearbeitet werden. Bearbeiten bedeutet in diesem Fall, dass das Konzept zur Erstellung einer neuen (Konzept-) Version herangezogen werden kann. Damit nicht mehrere Benutzer gleichzeitig von dem selben Konzept eine neue Version erstellen können, die sie später beide einbringen wollen, muss die Möglichkeit bestehen, ein Konzept bis zur Freigabe der neuen Version zu sperren.

4.4.1 Status-quo

Sowohl ORACLE7 als auch DB2 unterstützen das Gruppenkonzept (DB2 über das Betriebssystem, ORACLE7 über eine eigene Benutzerverwaltung) in welchem ein Benutzer Mitglied in verschiedenen Gruppen sein kann.

Rechte an einem Datenbankobjekt können einzelnen Benutzern oder auch einzelnen Gruppe zugeteilt werden.
Die Granularität der Rechte reicht allerdings nicht bis auf die Tupel-Ebene, es können nur Rechte an ganzen Tabellen oder Sichten vergeben werden.

Um dennoch den Zugriff nur auf einzelne Tupel einer Tabelle zu beschränken, müssen Sichten angelegt werden, wodurch allerdings die Unterstützung des Gruppenkonzepts durch das DBMS verloren geht. Ein Beispiel verdeutlicht dies:

Angenommen man hat eine Basisrelation `Motor`, in der Benzin-, Diesel- und Brennstoffzellenmotoren gespeichert sind. Auf ihr sollen verschiedene Gruppen von Mitarbeitern zugreifen. Zu diesem Zweck werden Sichten angelegt, die den Datenbestand filtern. Beispielsweise sind die Sichten `Benzinmotor` und `Dieselmotor` definiert, die die entsprechenden Tupel anzeigen. Weiter gibt es die DBMS-Gruppe *Benzin* und *Diesel*, welche der Aufgabe entsprechend bestimmte Rechte an der jeweiligen Sicht inne haben (*Benzin* nur an `Benzinmotoren` und *Diesel* nur an `Dieselmotoren`).
Alice soll sich nun ausschließlich mit den Benzinmotoren beschäftigen während Bob nur ausschließlich auf die Dieselmotoren zugreifen darf. Carol soll als deren Vorgesetzte die Arbeit beider kontrollieren können.
Dazu wird Alice Mitglied in *Benzin*, Bob Mitglied in *Diesel* und Carol Mitglied in beiden Gruppen. Während Alice und Bob nun bei dem Zugriff auf ihre jeweiligen Sichten alle für sie relevanten Tupel angezeigt bekommen, ergibt sich für Carol das Problem, dass sie auf zwei Sichten zugreifen muss, damit sie alle für sie relevanten Tupel zu sehen bekommt. Das ist ein Problem der Namenstransparenz, welches dadurch auftritt, dass bei der Definition einer Sicht – welche gewissermaßen die Rechte einer Gruppe darstellt – ein neuer Name vergeben werden muss. Für keinen der drei Benutzer ist es in diesem Szenario möglich, auf eine für alle gleichlautende Tabelle oder Sicht (beispielsweise `Motordaten`) zuzugreifen, um die für ihn relevanten Tupel selektiert zu bekommen.

Abhilfe kann geschaffen werden, wenn man die Sichten in dem lokalen Schema des jeweiligen Benutzers anlegt. Durch den Schemazusatz sind die Sichten systemweit eindeutig, dennoch kann jeder Benutzer (durch Weglassen seines Schemabezeichners) auf den gleichen Sichtnamen zugreifen. Beispielsweise würde die bei allen Benutzern auf den (lokalen) Namen `Motordaten` lautende Tabelle/Sicht systemweit auf die eindeutigen Bezeichner `alice.Motordaten`, `bob.Motordaten` und `carol.Motordaten` abgebildet werden.

Die Zugehörigkeit zu einer Gruppe würde sich dann allerdings nicht mehr zwangsläufig über die reale Gruppenzugehörigkeit beim DBMS ergeben, sondern über die Definition der im lokalen Schema liegenden Sicht.
Dabei besteht die Möglichkeit die Gruppenzugehörigkeit direkt über Konstanten in der Sicht zu verankern (wodurch allerdings bei jeder Änderung der Mitgliedschaft in einer Gruppe die Sicht neu angelegt werden müsste) oder aber mit Hilfstabellen, in denen die Gruppenzugehörigkeit abgelegt wird und über die der Zugriff auf die Tupel geregelt wird.
Wegen der höheren Flexibilität ist der zweite Ansatz vorzuziehen. Wie die Definition der Sichten und Operationen darauf im Einzelnen aussieht, beschreibt der nächste Abschnitt. In ihm wird eine Realisierung der für die an MEntAs gestellten Anforderungen vorgeschlagen. Dabei wird auch ersichtlich, dass nicht alle Integritätsbedingungen durch das Datenbanksystem garantiert werden können und eine Unterstützung durch das Anwendungsprogramm erfolgen muss.

4.4.2 Aufbau der Konzeptdatenbank

Der Einfachheit halber wurde das tatsächlich in MEntAs verwendete Motorkonzeptschema auf die drei Tabellen `Motor`, `Ventil` und `Kolben` und einige exemplarische Attribute beschränkt. In den Tabellendefinitionen steht PK für den Primärschlüssel (*primary key*) und FK für den Fremdschlüssel (*foreign key*), der den entsprechenden Primärschlüssel referenziert.

```
CREATE TABLE mentas.Motor (
  Konzept      VARCHAR(20) NOT NULL,
  Version      VARCHAR(10) NOT NULL,
  Benutzer     VARCHAR(8)
               CONSTRAINT cs_mot_fk_benutzer
                 REFERENCES mentas.Benutzer(PK_Benutzer),
  Gruppe       INTEGER
               CONSTRAINT cs_mot_fk_gruppe
                 REFERENCES mentas.Gruppe(PK_Gruppe),
  Freigabe     SMALLINT
               CONSTRAINT cs_freigabe
                 CHECK (Freigabe = 0 OR Freigabe = 1),
  Sperrung     SMALLINT,
  Leistung     INTEGER,
  FK_Ventil    INTEGER
               CONSTRAINT cs_fk_ventil
                 REFERENCES mentas.Ventil(PK_Ventil),
  FK_Kolben    INTEGER
               CONSTRAINT cs_fk_kolben
                 REFERENCES mentas.Kolben(PK_Kolben),
  CONSTRAINT cs_pk_motor
    PRIMARY KEY (Konzept, Version),
  CONSTRAINT cs_sperrung
    CHECK(Sperrung = 0 OR (Sperrung = 1 AND Freigabe = 1))
  );
```

`Leistung` stellt exemplarisch ein Attribut der Nutzdaten des Motorkonzeptes dar. Die übrigen Attribute dienen weitestgehend der Konzeptverwaltung. Ein Konzept wird eindeutig über die beiden Primärschlüsselattribute `Konzept` und `Version` angesprochen. `Benutzer` und `Gruppe` geben den Besitzer des Tupels an. Die `cs_sperrung`-Beschränkung in der Definition verhindert, dass noch nicht freigegebene Konzepte gesperrt sind.

```
CREATE TABLE mentas.Ventil (
  PK_Ventil    INTEGER NOT NULL
               CONSTRAINT cs_pk_ventil
                 PRIMARY KEY,
  Hub          FLOAT
  );
```

```
CREATE TABLE mentas.Kolben (
  PK_Kolben    INTEGER NOT NULL
               CONSTRAINT cs_pk_kolben
                 PRIMARY KEY,
  Durchmesser  FLOAT
  );
```

Schließlich folgen noch die beiden Tabellen zur Verwaltung der Benutzer- und Gruppendaten.

```
CREATE TABLE mentas.Benutzer (
  PK_Benutzer  VARCHAR(8) NOT NULL
               CONSTRAINT cs_pk_benutzer
                 PRIMARY KEY,
  Name         VARCHAR(20) NOT NULL,
  Vorname      VARCHAR(20)
  );
```

`PK_Benutzer` bezeichnet die Benutzerkennung, mit der die Anmeldung bei dem DBMS erfolgt. Da sie systemweit eindeutig ist – sie ist ja die Betriebssystemzugangskennung – fungiert sie gleichzeitig als Primärschlüssel.

```
CREATE TABLE mentas.Gruppe (
  PK_Gruppe    INTEGER NOT NULL
               CONSTRAINT cs_pk_gruppe
                 PRIMARY KEY,
  Bezeichnung  VARCHAR(20) NOT NULL
  );
```

Schließlich folgt noch die Tabelle `Mitglied`. In ihr wird festgehalten, welcher Benutzer welcher Gruppe angehört. Diese Tabelle ist notwendig, da es sich bei der Beziehung zwischen Benutzer und Gruppe um eine n:m-Beziehung handelt (ein Gruppe kann mehrere Mitglieder haben, ein Benutzer kann aber auch Mitglied in mehreren Gruppen sein).

```
CREATE TABLE mentas.Mitglied (
  FK_Benutzer  VARCHAR(8)
               CONSTRAINT cs_mit_fk_benutzer
                 REFERENCES mentas.Benutzer(PK_Benutzer)
                 ON DELETE CASCADE,
  FK_Gruppe    INTEGER
               CONSTRAINT cs_mit_fk_gruppe
                 REFERENCES mentas.Gruppe(PK_Gruppe)
                 ON DELETE CASCADE
  );
```

Die `ON DELETE CASCADE` Zusätze bei den Fremdschlüsseln gewährleisten, dass beim Löschen eines Benutzers oder einer Gruppe auch die Mitgliedschaften des Benutzers bzw. in der Gruppe beendet werden.

```
CREATE VIEW mentas.MotorView AS
  SELECT *
  FROM mentas.Motor
  WHERE Benutzer = USER
     OR (Freigabe = 1 AND Gruppe IS NULL)
     OR (Freigabe = 1 AND Gruppe IN (
     SELECT FK_Gruppe
       FROM mentas.Mitglied
      WHERE FK_Benutzer = USER)
  ) WITH CHECK OPTION;
```

Mit dieser Sicht werden alle Konzepte angezeigt, von denen man Besitzer ist (also alle freigegebenen und nicht freigegebenen Konzepte; vgl. unten), sowie alle freigegebenen Konzepte, die keiner Gruppe angehören und damit öffentlich sind (`Gruppe IS NULL`), oder aber einer Gruppe angehören, in der man Mitglied ist.

`USER` ist ein spezielles Register der Schema-Umgebung, in dem die jeweilige Benutzerkennung (der Schemaname) vermerkt ist. Bei jeder Operation, die auf der Sicht `mentas.MotorView` ausgeführt wird, wird also `USER` durch die Kennung des Benutzers ersetzt, wodurch er Zugriff auf die für ihn relevanten Daten erhält. Damit erübrigt sich die Eingangs vorgeschlagene Erzeugung einer individuellen Sicht im Schema jedes Benutzers.

Die Beschränkung `WITH CHECK OPTION` stellt sicher, dass keine Tupel in die Basistabelle eingefügt werden können, die nach der Einfügung ausserhalb der Sicht liegen würden. Bedingt durch die Oder-Verknüpfungen wird allerdings nur ein Teilbereich aller unzulässigen Kombinationen bei Einfüge- oder Änderungsoperationen unterbunden. Beispielsweise könnte von Alice ein Tupel eingefügt werden, in deren Gruppe sie nicht Miglied ist, da bereits der Vergleich mit ihrer Benutzerkennung das Tupel nach der Einfügung selektieren würde. Daher sind für korrekte Einfügeoperationen weitere Maßnahmen zur Konsistenzsicherung notwendig. Dazu später mehr.

Schließlich müssen noch die Rechte auf die Datenbankobjekte vergeben werden: auf die Basistabellen erhält nur der Administrator (Benutzer `mentas`) Zugriff (`REVOKE ALL PRIVILEGES ON TABLE mentas.`⟨*Tabellenname*⟩ `FROM PUBLIC`), während Lese-, Einfüge- und Änderungsrechte für die Sicht `mentas.MotorView` an alle Benutzer vergeben werden.[5]

4.4.3 Datenmanipulation

In den folgenden SQL-Anweisungen werden in Anlehnung an *embedded SQL* individuell zu ergänzende Werte – sei es durch eine Konstante oder den Inhalt einer Hostvariablen – mit einem vorangestellten Doppeltpunkt wie bei `:benutzer_neu` dargestellt.

[5] Da es prinzipiell für jeden Benutzer mit einer Kennung auf dem Rechner des Datenbank-Servers möglich ist, sich bei DB2 anzumelden, wäre es besser, alle Benutzer von MEntAs in einer dafür eingerichteten Betriebssystemgruppe zusammenzufassen und dieser dann die Rechte an `MotorView` zu erteilen: `GRANT SELECT, INSERT, DELETE ON TABLE mentas.MotorView TO GROUP` ⟨*mentasgrp*⟩

Benutzerverwaltung

Anlegen von Benutzern. Beim Betriebssystem wird ein neuer Benutzer für den Zugang zu der Middleware angelegt (`PK_Benutzer`-Attribut). Seine Daten werden zusammen mit der Zugangskennung in die Tabelle `Benutzer` gespeichert. In der Tabelle `Mitglied` werden für ihn die Gruppenzugehörigkeiten festgelegt.

Anmerkung: Es sollte darauf geachtet werden, dass alle `PK_Benutzer`-Angaben in Großbuchstaben gemacht werden, da das Register `USER` eine solche Zeichenkette zurückliefert und beispielsweise ein Vergleich mit `'alice'` zu einem falschen Ergebnis führen würde.

Ändern von Benutzern. Probleme treten hier nur auf, wenn die Benutzerkennung (`PK_Benutzer`) geändert wird (was, wenn überhaupt, nur sehr selten der Fall sein dürfte).[6] Eine einfache Änderung ist nicht möglich, da u. U. ein Fremdschlüssel aus der Mitglied-Tabelle darauf referenziert, welcher seinerseits nicht einfach geändert werden kann, da ein dort eingetragener Wert bereits in der Benutzer-Tabelle vorhanden sein muss. Um diesen „Ringschluss“ zu umgehen, ohne dass alle Mitglied-Einträge erst gelöscht und später dann mit der neuen Benutzerkennung wieder eingetragen werden müssen, wird zuerst eine Kopie des Benutzers mit neuem `PK_Benutzer`-Wert erstellt, danach ein `UPDATE` auf den Mitglied-Einträgen durchgeführt und schließlich der alte Benutzereintrag gelöscht.

```
INSERT
  INTO mentas.Benutzer
VALUES
  SELECT :login_neu, Name, Vorname
    FROM mentas.Benutzer
   WHERE PK_Benutzer = :login_alt

UPDATE mentas.Mitglied
   SET FK_Benutzer = :login_neu
 WHERE FK_Benutzer = :login_alt;

UPDATE mentas.MotorView
   SET Benutzer = :login_neu
 WHERE Benutzer = :login_alt;

DELETE
  FROM mentas.Benutzer
 WHERE PK_Benutzer = :login_alt;
```

Löschen von Benutzern. Die Löschung von Benutzern wirkt sich hauptsächlich auf die Konzepte aus, von denen sie Eigentümer sind. Einträge, die bereits freigegeben wurden, sollen auch weiterhin erhalten bleiben, es wird nur die Benutzerkennung

[6] Der Name einer Betriebssystemkennung kann nicht einfach geändert werden. Es ist notwendig, einen Benutzer mit dem neuen Namen anzulegen und den alten zu löschen.

gelöscht (erhält den Wert `NULL`). Noch nicht freigegebene Konzepte werden entweder einfach gelöscht, oder können an einen anderen Benutzer übergeben werden. An wen die Übergabe erfolgen soll, kann durch einen Benutzerdialog ermittelt werden, in dem beispielsweise nur Personen angezeigt werden, die in einer Gruppe Mitglied sind, der auch die zu löschende Person angehört.

```
UPDATE mentas.MotorView
   SET Benutzer = NULL
 WHERE Benutzer = :benutzer
   AND Freigabe = 1;

DELETE
  FROM mentas.MotorView
 WHERE Benutzer = :benutzer
   AND Freigabe = 0;

DELETE
  FROM mentas.Benutzer
 WHERE PK_Benutzer = :benutzer;
```

Wegen der `ON DELETE CASCADE` Bedingung in der Definition der Mitgliedrelation werden die relevanten Mitgliedtupel automatisch beim Löschen des Benutzers mitgelöscht.

Gruppenverwaltung

Anlegen von Gruppen. Das Anlegen einer neuen Gruppe bringt ausser der Eindeutigkeit der Tupelwerte keine Probleme mit sich.

```
INSERT
  INTO mentas.Gruppe (PK_Gruppe, Bezeichnung)
VALUES (:gruppe, :bezeichnung);
```

Ändern von Gruppen. Bei der Änderung des Primärschlüsselattributes `PK_Gruppe` ergeben sich die gleichen Probleme wie bei der Änderung von `PK_Benutzer`. Bis auf das Anlegen einer neuen Betriebssystemkennung ist hier die Vorgehensweise analog zu dem Fall der Benutzerverwaltung.

Ändern der Gruppenzugehörigkeit. Das Hinzufügen der Mitgliedschaft eines Benutzers in einer neuen Gruppe verläuft reibungslos, ihm wird lediglich Zugriff auf eine größere Anzahl von Konzepten gewährt.

Bei der Löschung einer Gruppenzugehörigkeit treten allerdings gewisse Inkonsistenzen auf, die dadurch bedingt sind, dass beim Einfügung eines neuen Konzeptes (vgl. unten) nur Gruppen verwendet werden dürfen, in denen der Eigentümer des Konzepts auch Mitglied ist. Durch die lokale Motorsicht würde der Benutzer aber immer noch sein Konzept sehen, das jetzt allerdings einer Gruppe angehört, in der er nicht mehr Mitglied ist.

Für diesen Fall können zwei Lösungsansätze verfolgt werden. Der eine geht von dem Standpunkt aus, dass ein Konzept für eine bestimmte Gruppe erstellt wurde. Da der Eigentümer jetzt nicht mehr Mitglied in dieser Gruppe ist, gehört das Konzept nur noch der Gruppe. (`:gruppe` bezeichne die Gruppe, aus der der Benutzer `:benutzer` herausgenommen wurde.)

```
UPDATE mentas.MotorView
   SET Benutzer = NULL
 WHERE Benutzer = :benutzer
   AND Gruppe = :gruppe;
```

Alternativ könnten die Konzepte an einen anderer Benutzer aus der Gruppe übergeben werden (`SET Benutzer = :anderer_benutzer`).

Der andere Ansatz stellt mehr den Eigentümer in den Vordergrund. Er hat das Konzept erstellt, also bleibt es auch in seinem Besitz. Es wird einfach die Gruppenzugehörigkeit gelöscht.

```
UPDATE mentas.MotorView
   SET Gruppe = NULL
 WHERE Benutzer = :benutzer
   AND Gruppe = :gruppe;
```

Nun ist das Konzept für jeden zugänglich, was unter Umständen nicht erwünscht war. Man könnte es daher „unfrei“ machen, wodurch es nur noch der Eigentümer sieht, und diesen beispielsweise durch e-mail über die erfolgte Umstellung informieren.

Da an der Erstellung eines Konzeptes in der Regel mehrere Ingenieure beteiligt sind, tritt neben der eben skizzierten, etwas aufwendigen Vorgehensweise zudem noch der Effekt einer zerstückelten Entwicklungshistorie auf. Nimmt jeder aus der Entwicklungsgruppe ausscheidende Ingenieur seine Konzepte mit, treten Lücken in der Versionskette auf.

Aus diesem Grund ist der erste Lösungsvorschlag (Löschen oder Übergabe des Eigentümers) vorzuziehen. Dieser kann durch das DBS mit folgendem Trigger[7] unterstützt werden:

```
CREATE TRIGGER delete_member AFTER DELETE ON mentas.Mitglied
  REFERENCING OLD AS old
  FOR EACH ROW MODE DB2SQL
  UPDATE mentas.Motor SET Benutzer = NULL
   WHERE Benutzer = old.FK_Benutzer AND Gruppe = old.FK_Gruppe;
```

Löschen von Gruppen. Auch hier ist die Vorgehensweise analog zu der beim Löschen von Benutzern. Die Konzepte gehören damit keiner Gruppe mehr an und sind für jeden verfügbar.

Optional wäre denkbar, einen Benutzerdialog zu führen, ob die Tupel einer anderen Gruppe zugeordnet oder aber mitgelöscht werden sollen.

[7] Ein Erläuterung des Trigger-Konzepts findet sich in Abschnitt 7.3.2 auf Seite 97.

Konzeptverwaltung

Lesen von Konzepten. Dazu wird einfach die Motorsicht verwendet, die alle für einen Benutzer relevanten Konzepte anzeigt. Alternativ kann nach bestimmten Kriterien gefiltert werden:

- Anzeigen aller Konzepte (freigegebene und nicht freigegebene) des Benutzers sowie alle diejenigen, zu dessen Gruppe er gehört.

```
SELECT *
  FROM mentas.MotorView;
```

- Nur die eigenen Konzepte (freigegebene und nicht freigegebene) werden angezeigt.

```
SELECT *
  FROM mentas.MotorView;
 WHERE Benutzer = USER;
```

- Ausgabe der nicht freigegebenen (eigenen) Konzepte. Die Satzbedingung `AND Benutzer = USER` ist nicht notwendig, da durch die Sichtdefinition bedingt eh nur solche Konzepte – unter der Bedingung `Freigabe = 0` – angezeigt werden.

```
SELECT *
  FROM mentas.MotorView;
 WHERE Freigabe = 0;
```

- Auflistung aller gerade gesperrten Konzepte (Versionen).

```
SELECT *
  FROM mentas.MotorView;
 WHERE Sperrung = 1;
```

Erstellung eines neuen Konzepts. Der Befehl für diese Operation sieht folgendermaßen aus:

```
INSERT
  INTO mentas.MotorView (Konzept, Version, Benutzer, Gruppe,
                         Freigabe, Sperrung, ...)
VALUES (:konzept, :version, USER, NULL, 0, 0, ...);
```

Bei einer generellen Einfügeoperation in die Motortabelle sind die Werte für `Konzept` und `Version` benutzerbestimmt. Das DBS übernimmt lediglich die Kontrolle darüber, dass eine Kombination aus diesen beiden Werten nicht bereits in der Datenbank vorhanden ist (Primärschlüsselbeschränkung). Die Nutzdaten `Leistung` sowie die Verweise auf die verwendeten Bestandteile werden ebenfalls von dem Ingenieur festgelegt; einzige Bedingung ist hier, dass die referenzierten Motorteile in der Datenbank vorhanden sein müssen. Auch diese Einschränkung wird von dem DBS durch die Definition der Fremdschlüssel geprüft.

Bei den übrigen Attributen `Benutzer`, `Gruppe`, `Freigabe` und `Sperrung` handelt es sich um Verwaltungsdaten, deren korrekte Werte beim Einfügen durch nachfolgenden Trigger[8] gewährleistet werden sollen.

Die zugrundeliegende Annahme dabei ist, dass ein Konzept vorerst lokal eingefügt wird (nicht freigegeben). Die Zuordnung zu einer konkreten Gruppe erfolgt erst zum Zeitpunkt der Freigabe (daher vorerst einmal ein Null-Wert). Ausserdem darf der Ingenieur keine andere als seine eigene Benutzerkennung als Wert für das Attribut `Benutzer` angeben.

```
CREATE TRIGGER insert_trigger
  NO CASCADE BEFORE INSERT ON mentas.Motor
  REFERENCING NEW AS new
  FOR EACH ROW MODE DB2SQL
  WHEN ((new.Benutzer <> USER)
     OR (new.Freigabe <> 0)
     OR (new.Gruppe IS NOT NULL))
    SIGNAL SQLSTATE '70001' ('Insertion violation.');
```

Der Wert für `Sperrung` wird durch die Forderung `Freigabe` = 0 und die Beschränkung `cs_sperrung` bei der Definition der Motortabelle automatisch als 0 erwartet.

Erstellung einer neuen Konzeptversion. Ausgangspunkt dabei ist ein bereits bestehendes Konzept. Damit von diesem aber nicht unabhängig voneinander zwei Benutzer eine neue Version erstellen können, muss das Konzept bis zur Freigabe der neuen Version gesperrt bleiben.

```
UPDATE mentas.MotorView
   SET Sperrung = 1
 WHERE Konzept = :konzept
   AND Version = :version;
```

Innerhalb des Anwendungsprogramms muss darauf achten, dass der Wert des Attributes `Konzept` vom Ingenieur nicht verändert werden kann (er erstellt ja eine neue Version und kein neues Konzept). Dadurch wird er beim Einfügen der neuen Version gezwungen, eine neue Versionsnummer anzugeben (resultiert aus der Primärschlüsseleigenschaft).

```
INSERT
  INTO mentas.MotorView (Konzept, Version, Benutzer, Gruppe,
                         Freigabe, Sperrung, ...)
VALUES (:konzept, :version, USER, NULL, 0, 0, ...);
```

Die Aufhebung der Sperrung des Ausgangskonzepts darf erst erfolgen, wenn eine Folgeversion des gesperrten Konzepts freigegeben wurde.

[8] Trigger können nicht auf Sichten definiert werden, daher wird dieser direkt auf die Motortabelle gesetzt.

```
UPDATE mentas.MotorView
   SET Sperrung = '0'
 WHERE Bezeichnung = :bezeichnung
   AND Version = :version;
```

Freigabe von Konzepten. Bei der Freigabe eines Konzeptes muss das Anwendungsprogramm beim Benutzer eine Gruppenzugehörigkeit erfragen. Ihm werden alle diejenigen Gruppen zur Auswahl angeboten, in denen er Mitglied ist.

```
UPDATE mentas.MotorView
   SET Freigabe = '1', Gruppe = :gruppe
 WHERE Konzept = :konzept,
   AND Version = :version;
```

Folgende Trigger erzwingt diese Gruppenzugehörigkeit:

```
CREATE TRIGGER release
  NO CASCADE BEFORE UPDATE ON mentas.Motor
  REFERENCING NEW AS new
  FOR EACH ROW MODE DB2SQL
  WHEN  ((new.Benutzer <> USER)
      OR (new.Gruppe NOT IN (SELECT FK_Gruppe
                               FROM mentas.Mitglied
                              WHERE FK_Benutzer = USER)))
    SIGNAL SQLSTATE '70002' ('Invalid user- or group-id while
                              releasing conecpt');
```

Da die Änderungsoperation auf der Motorsicht ausgeführt wird, können nur die lokalen Tupel freigeben werden, die einem selbst gehören (andere werden ja überhaupt nicht angezeigt). Alle übrigen angezeigten Datensätze wurden bereits von anderen Personen freigeben. Es ist also darauf zu achten, dass keine Änderungsoperationen an bereits freigebenen Konzepten vollzogen werden.

Wegen der administrativen Aspekte – beispielsweise wenn ein Benutzer gelöscht wird, wird das Benutzerattribut seiner Konzepte auf einen Nullwert geändert – kann kein generelles Verbot für alle Änderungsoperation, die nicht eine Freigabe darstellen, erteilt werden.

Generell muss bei der Freigabe geprüft werden, ob das eingefügte Tupel eine neue Konzeptversion war. Für den Fall, dass nämlich die Vorgängerversion kein lokales, sondern ein schon freigegebenes Konzept war, muss die Sperrung bei diesem Ausgangskonzept aufgehoben werden.

Löschen von Konzepten. Gelöscht werden dürfen nur Konzepte, die einem selbst gehören, oder aber die keinen Besitzer mehr haben.

```
DELETE
  FROM mentas.MotorView
 WHERE Konzept = :konzept,
   AND Version = :version;
```

Die Einhaltung obiger Forderung wird abermals durch einen Trigger realisiert.

```
CREATE TRIGGER delete_trigger
  NO CASCADE BEFORE DELETE ON mentas.Motor
  REFERENCING OLD AS old
  FOR EACH ROW MODE DB2SQL
  WHEN ((old.Benutzer <> USER) AND
        (old.Benutzer IS NOT NULL))
    SIGNAL SQLSTATE '70003' ('Not owner of concept');
```

Verwaltung der Motorkomponenten

Bisher wurde nur der Zugriff auf das zentrale Konzeptelement – die Motortabelle – berücksichtigt. Wie im Abschnitt über den Datenbankaufbau beschrieben, gibt es aber noch weitere Tabellen, in denen die Komponenten eines Motors (Ventil und Kolben) verwaltet werden. Durch eine Fremschlüsselbeziehung werden sie mit der Motortabelle in Verbindung gebracht.

Entstanden sind diese Tabellen, indem – zur Vermeidung von Redundanzen bei der Speicherung – eine Selektion mit Tupelelimination auf die jeweiligen (Ventil-, Kolben-) Attribute einer formals einzigen (Motor-) Tabelle durchgeführt wurde. Die Eindeutigkeit eines Eintrages in den Komponententabellen ergibt sich also im Gegensatz zu der Motortabelle nicht aus der Konzeptionierung eines bestimmten Teils, sondern vielmehr aus der Unterschiedlichkeit aller in der Tabelle aufgenommen Maße.[9] Aus diesem Grund würde eine Benutzer-/Gruppenverwaltung bzw. Versionierung dieser Daten nur zu einer Verkomplizierung des Zugriffs und der Verwaltung führen.
Probleme entstehen nämlich schon dann, wenn beispielsweise für ein Ventil die Gruppenzugehörigkeit gewechselt wird und damit unter Umständen nicht mehr für ein Motorkonzept, das einer anderen Gruppe angehört, sichtbar ist.
Vielmehr sollte man sich die Komponententabellen als eine Art *Repository* vorstellen, aus denen die Ingenieure die Grundbestandteile ihres Motors auswählen und das sie gegebenenfalls ergänzen können.

Eine Verwaltung der Komponententabellen könnte so aussehen, dass von Zeit zu Zeit eine Löschoperation durchgeführt wird, die all diejenigen Tupel löscht, die in keinem Motor (lokalen oder freigegebenen) verwendet werden.
Durch das bewusste Weglassen der `ON DELETE CASCADE` Klausel bei der Definition der Motortabelle wird sichergestellt, dass nicht versehentlich eine Komponente (und damit dann auch das Motortupel selbst) gelöscht wird, welche in einem Motorkonzept noch Verwendung findet.

[9] Dass ein Tupel der Komponententabellen seine Eindeutigkeit bereits aus dem Primärschlüssel `PK_Ventil` bzw. `PK_Kolben` bezieht, hat nichts mit der zugrundeliegenden Datenstruktur zu tun; dieser Schlüssel wurde nur zu kompakten Referenzierung in der Motortabelle eingeführt.

4.5 Zusammenfassung

Dieses Kapitel befasste sich mit den Rahmenbedingungen des Motorentwicklungsassistenten: der Sicherungsinfrastruktur, in der MEntAs eingebettet ist, der Programmierprache Java, in der sowohl Client als auch Server programmiert wurden und den verwendeten/integrierten Datenbanken. Abschließend wurde ein Datenbankkonzept für die Verwaltung der Motorkonzeptdaten vorgestellt, mit dem sich eine Rechtevergabe auf Tupelebene realisieren lässt.

Kapitel 5

Zugangskontrolle

In Kapitel 2, den Grundlagen zu dieser Arbeit, wurden die drei verschiedenen Aspekte eines Menschen, mit denen er seine Identität bestätigen kann beschrieben. In diesem Kapitel sollen nun konkrete Methoden zu diesen Aspekten behandelt werden, sowie eine Abwägung deren Stärken und Schwächen erfolgen. Den größten Anteil nehmen dabei die Betrachtungen zu der im EDV-Bereich weit verbreiteten Authentifizierung durch Passwörter ein.

5.1 Biometrische Systeme

Die körperspezifischen Merkmale eines Benutzers sind extrem schwer zu verändern oder nachzubilden, wodurch eine relativ hohe Sicherheit bei der Authentifizierung über diese Eigenschaften gegeben ist.
Solche Merkmale sind zum Beispiel der Fingerabdruck in klassischer oder genetischer Form, die Stimme, die Netzhaut (Retina) oder Regenbogenhaut (Iris) des Auges. Neueste Forschungen beschäftigen sich mit den Proportionen der menschlichen Hand, von denen man vermutet, dass sie ähnliche Authentifizierungseigenschaften haben wie eben genannte Körpermerkmale. Aber auch Eigenschaften wie die Dynamik beim Benutzen einer Tastatur oder bei der Ausführung der handschriftlichen Signatur werden als individuelle Charakteristika angenommen.

Egal welche Eigenschaft man zugrunde legt, gemeinsam ist allen biometrischen Verfahren, dass sie eine Einheit zur Erfassung der Körpermerkmale benötigen. Gerade darin liegt deren Schwierigkeit. Während bei den wissensbasierten Systemen die Eingabe direkt über eine bereits vorhandene Peripherie (Tastatur oder Maus) erfolgt und bei den token-basierten ein wahlweise „maschinennaher“ Gegenstand geprüft wird (Chipkarte, Diskette), erfordern biometrische Systeme das Lesen von „unstrukturierten“ Daten. Das bedeutet, dass für die Autentifizierung neben der eigentlichen Prüfung der Übereinstimmung mit dem hinterlegten Referenzwert eine Möglichkeit gefunden werden muss, die unstrukturierten Daten in eine maschinenlesbare Form zu bringen. Dies erfolgt in einem ersten Schritt über Sensoren (Kamera, Mikrofon). Der zweite Schritt besteht darin, aus den Eingabedaten, die durch das Aufzeichnungsverfahren bedingt gewissen Schwankungen unterliegen,[1] die charakteristischen Werte herauszufiltern.

[1] Solche Schwankungen sind beispielsweise die Helligkeit bei Kameras oder Umgebungsgeräusche bzw. Heiserkeit des Sprechers bei Mikrofonen.

Die Vorteile biometrischer Systeme liegen auf der Hand: Fälschungen sind nur schwer durchzuführen, Zugangscodes müssen sich nicht mehr gemerkt werden und auch gegen den Verlust des Tokens durch Diebstahl oder einfaches Vergessen ist diese Art von Systemen immun.

Es gibt aber auch Nachteile. Da sind zum Beispiel die zusätzlichen Kosten für die Eingabeeinheit. Diese treten zwar bei den token-basierten Systemen in der Regel auch auf (Chipkartenleser), allerdings sind dort die Kosten im Allgemeinen niedriger anzusetzen, da der Lesevorgang keine so komplexe Technologie erfordert.

Weiter darf die Benutzerakzeptanz nicht unberücksichtigt bleiben. Anfängliche Authentifizierungsversuche durch den Fingerabdruck scheiterten daran, dass der Finger, um störende Lichteinflüsse beim Lesen auszuschließen, in eine Öffnung gesteckt werden musste. Ein Vorgang, der vielen Testpersonen suspekt war (Was passiert mit meinem Finger in der Maschine?). Auch das Abtasten der Netzhaut mit einem Laserstrahl, dürfte bei den meisten Menschen trotz zugesicherter Gefahrlosigkeit ein psychisch bedingtes Unbehagen verursachen. Deshalb wird dieses Verfahren nur für Bereiche mit höchsten Sicherheitsanforderungen, wie beispielsweise dem Militär oder Geheimdienst, relevant sein.
Gänzliche Verweigerung dürfte eine DNS-Analyse mit sich bringen – wer möchte schon jedesmal eine Blut- oder Speichelprobe abgeben, nur um sich bei einem Programm zu authentifizieren. Daher wird diese Methode sicherlich nur in Ausnahmefällen zur Anwendung kommen.

Trotz der eben genannten Nachteile sind biometrische Systeme dennoch im Kommen. Bedingt durch den steigenden Bedarf an Netzwerksicherheit und die stetig fallenden Hardware-Kosten auch auf dem Sektor biometrischer Systeme, wird 1999 ein weltweiter Markt von 60 Millionen Dollar prognostiziert; dazu im Vergleich: 1997 waren es 24 Millionen Dollar (Lawton, 1998).

5.2 Token-basierte Systeme

Bei dieser Variation der Authentifizierung erfolgt oft eine Autorisierung ohne direkte Identifikation, wie beispielsweise bei einem Wohnungsschlüssel oder einer Zugangskarte zum Firmengelände. Die eigentliche Identifizierung wurde bereits bei der Vergabe des Gegenstandes duchgeführt, unter der Annahme, dass von nun an der alleinige Besitz maßgeblich für die Authentizität der Person ist.

Wie auch bei den biometrischen Systemen erfordert die token-basierte Vorgehensweise eine Leseeinheit für den Gegenstand. Ein gängiges Beispiel ist der Chipkartenleser. Ihn gibt es als interne Rechnerkomponente oder zum externen Anschluss an den seriellen Port. Auch Versionen, in denen das Gerät in die Tastatur integriert ist, sind auf dem Markt erhältlich.[2]

Eine Möglichkeit, schon bereits vorhandene Hardware zu nutzen, wäre die Authentifizierung mittels einer Diskette. Auf ihr wird der zur Überprüfung notwendige Schlüssel, das Passwort oder eine ID gespeichert. Problematisch erweist sich diese Vorgehensweise

[2] Auf der letzten CeBIT wurden auch Tastaturen vorgestellt, auf denen ein kleines Feld zum Lesen eines Fingerabdrucks integriert war.

bei Workstations. Zum Einen sind oft nicht alle Maschinen mit einem Laufwerk ausgerüstet, zum Anderen besteht bei einer Workstation die Möglichkeit, sich von einem anderen Rechner aus anzumelden (*rlogin*), wodurch ein Angreifer Zugriff auf das lokale Diskettenlaufwerk erhält und sich die Zugangsdaten einfach kopieren kann.

Eine andere Möglichkeit bieten sogenannte *Dongles*. Das sind kleine Hardware-Einheiten von der Größe eines Steckers, die meist an den Parallelport des Rechners angeschlossen und von der Authentifizierungs-Software abgefragt werden. Eigentlich kommen diese Apparate aus dem Bereich des Software-Schutzes, wo sie gegen Raubkopien und Software-Piraterie eingesetzt werden. Prinzipiell kann man sie sich aber auch zur Authentifizierung von Benutzern vorstellen.

Neben den zusätzlichen Kosten für eine Leseeinheit gibt es noch die Nachteile, die bei einem Verlust oder beim Vergessen des Gegenstandes entstehen: Liegt dem Benutzer das Token zum Anmeldezeitpunkt nicht vor, kann er keinen Zugang zu dem System erlangen. Gravierender ist aber, dass der Dieb des Gegenstandes sich problemlos unter einer fremden Identität anmelden kann (weshalb Tokens häufig nur in Kombination mit einem der anderen Authentifizierungsverfahren Verwendung finden).

5.3 Wissensbasierte Systeme

Wie in den Grundlagen dieser Arbeit erläutert, werden bei der Authentifizierung durch ein wissensbasiertes System dem Benutzer mehrere Fragen gestellt, deren richtige Beantwortung ihn als authentisch bestätigen (Frage-Antwort-Spiel). Richtig bedeutet in diesem Fall nicht immer logisch. Es ist durchaus denkbar, dass auf die Frage nach dem Namen der Mutter, der Name des Vaters eingegeben werden muss. Unter „richtig“ wird vielmehr verstanden, dass die für eine Frage zugehörige, gespeicherte Antwort gegeben werden muss. Dabei ist es nicht zwingend erforderlich, dass die Antwort direkt gespeichert ist. Denkbar wäre auch die Speicherung eines Algorithmus, der die richtige Antwort berechnet. Beispielsweise setzt die überprüfende Instanz dem Benutzer eine zufällige Zahl vor, die dieser nach einem nur ihm (und der Instanz) bekannten Algorithmus transformiert und das Ergebnis zur Authentifizierung eingibt. Ein Nachteil ist, dass, damit der Benutzer sich den Algorithmus merken kann, dieser recht einfach sein muss, wodurch er aber auch leicht erraten werden kann. (Weck, 1984)

Wenn die Aufgaben des Benutzers sich nicht unmittelbar mit der Systemsicherheit beschäftigen, werden Anmeldevorgänge eher als lästiger *overhead* empfunden. Ein Umstand, der sich bei dem derzeitigem Integrations-Boom in Bestrebungen eines *single logon* ausdrückt: Die durch eine heterogene Produktlandschafte geprägten immer wiederkehrenden Anmeldevorgänge (beim Betriebssystem, beim Datenbanksystem, beim Mail-Programm usw.), bei denen zum Teil systembedingt unterschiedliche Benutzerkennungen und Passwörter verwendet werden müssen, sollen durch eine (für den Benutzer) einmalige Identifizierungs- und Authentifizierungsprozedur zu Beginn der Arbeitssitzung erledigt werden.
Ein Umstand, der verdeutlicht, dass mehrstufige Passwortverfahren – und nichts anderes stellt funktional betrachtet ein Frage-Antwort-Spiel dar – von Benutzerseite her wenig Akzeptanz finden. Diese Bequemlichkeit des Benutzers dürfte einer der Gründe

sein, warum sich die Frage-Antwort-Spiele trotz offensichtlich größerer gebotener Sicherheit gegenüber den einfachen Passwortverfahren nicht durchgesetzt haben. Ausserdem können unter Berücksichtigung einiger Regeln Passwortverfahren einen durchaus akzeptablen Schutz bieten. Diese Regeln sollen im folgenden Abschnitt anhand der gängigsten Passwortbedrohungen erläutert werden.
Da manche dieser Gegenmaßnahmen auf den guten Willen des Benutzers angewiesen sind, dieser aber (aus welchen Gründen auch immer) nicht immer vorausgesetzt werden kann, sollte eine software-seitige Unterstützung der Maßnahmen in Erwägung gezogen werden.

Ein weiterer Nachteil von Passwortverfahren ist, dass der Diebstahl eines Passwortes nicht unmittelbar entdeckt wird. Der rechtmäßige Benutzer kann sich weiterhin anmelden und merkt unter Umständen garnicht oder erst sehr spät, dass noch jemand anderes unter seiner Kennung am Arbeiten ist.[3]
Diesem Problem kann man nur dahingehend begegnen, dass man häufig sein Passwort wechselt. Dieser Wechsel liegt entweder in dem Verantwortungsbereich des Benutzers (was sehr häufig mit keinem Passwortwechsel gleichbedeutend ist) oder wird von dem System erzwungen. Die ausgeprägteste Form, d. h. Wechsel bei jeder Anmeldung, wird mit sogenannten Einmal-Passwörtern erreicht, die in Abschnitt 5.3.3 betrachtet werden.

5.3.1 Angriffe auf Passwortverfahren

Geht man davon aus, dass die Passwortinformation verschlüsselt in dem System abgespeichert wird, und das Verschlüsselungsverfahren selbst als sicher gilt, d. h. man aus dem verschlüsselten Passwort nicht die Originaleingabe zurückgewinnen kann, gibt es immer noch diverse Möglichkeiten, in ein durch Passwörter geschütztes System einzudringen. Es sind im Einzelnen

- Brute-force Angriffe,
- Wörterbuchangriffe,
- Replay/Lausch-Angriffe,
- Spoof-Logins und
- Angriffe gegen die Passwortdatei.

Brute-Force Angriffe

Mittels systematischem Durchprobieren aller Zeichenkombinationen versucht der Angreifer sich Zugang zu verschaffen. Da die Anzahl aller möglichen Passwörter nur in den seltensten Fällen so gering ausfällt, dass man sie kurz von Hand durchprobieren könnte, werden dazu Programme eingesetzt.
Nimmt man als Möglichkeiten für ein Zeichen in einem Passwort alle Groß- und Kleinbuchstaben (`a-z`, `A-Z`; 52 Zeichen), die zehn Ziffern (`0-9`) und die Sonderzeichen `!"#$% &'‘()*+,-./:;<=>?@[\]^_{|}~` (32 Zeichen), so ergibt sich eine Gesamtanzahl von 94 möglichen Zeichen. Allerdings werden `\`, `#` und `@` von manchen Systemen als spezielle Steuerzeichen (Escape-Zeichen) interpretiert, ihre Verwendung bei Passwörtern sollte deshalb vermieden werden. In UNIX haben Passwörter eine Länge zwischen 6 und 8

[3] Möglichkeiten dies herauzufinden sind die Zeit der letzten Anmeldung, die Liste aller Prozesse eines Benutzers oder der Zeitpunkt, zu dem bestimmte Dateien geändert oder Programme ausgeführt wurden.

Zeichen. Daraus und unter der Verwendung von 91 möglichen Einzelzeichen ergeben sich 4 754 769 247 339 293 Möglichkeiten für die Bildung eines Passwortes.

Gegenmaßnahmen. Nimmt man die Zeit von 0,4 Millisekunden für die Überprüfung eines Passwortes an,[4] würde man rein rechnerisch im Mittel immer noch 950 953 849 467 Sekunden benötigen, um die richtige Zugangskombination zu finden; das entspricht 30 154 Jahren. Selbst bei einer Vertausenfachung der Rechengeschwindigkeit (der Testrechner war ja nicht sonderlich schnell) müßte man noch nicht ganz ein halbes Menschenleben auf das Result warten. –
Der Schutz vor Brute-Force Angriffen besteht also einfach darin, die mittlere Zeit bis zum Finden eines Passwortes, möglichst lang werden zu lassen, so dass von vornherein eine solche Vorgehensweise wegen ihrer Aussichtslosigkeit nicht durchgeführt wird.

Die eine Möglichkeit, um dies zu erreichen, besteht darin, möglichst viele verschiedene Zeichen zu verwenden und das Passwort so lang wie möglich zu machen. Schon wenn man anstelle eines 8-stelligen Passwortes nur ein 6-stelliges verwendet, schränkt man den Suchraum bereits auf 0,012% des 8-stelligen ein. Werden ausserdem nur noch Kleinbuchstaben verwendet, braucht es im Mittel lediglich 154 457 888 Überprüfungen bis das richtige Passwort gefunden ist; ein Vorgang, der nach obigen Annahmen nur 17 Stunden benötigt. Diese Zahl ist keineswegs nur ein konstruierter *worst-case*. In einer von Garfinkel und Spafford (1996, S. 65) angeführten Studie wurde herausgefunden, dass nur 6% aller Passwörter Interpunktionszeichen und Leerzeichen enthielten.

Eine andere Möglichkeit die Laufzeit des Angriffs in die Höhe zu treiben, besteht darin, bei der Überprüfung des Passwortes eine Verzögerung einzubauen, falls eine falsche Eingabe erkannt wurde. Dabei sollte man aber darauf achten, dass die Verzögerung im Überprüfungsalgorithmus selbst vorgenommen wird und nicht beispielsweise in dem *rlogin*-Programm. In diesem Fall könnte ein Angreifer sich mit Hilfe der Systembibliotheken, die die eigentliche Überprüfung vornehmen, ein eigenes Programm erstellen, dass die eingebaute Verzögerung von *rlogin* umgeht und somit in kürzerer Zeit mehr Kombinationen durchprobieren kann.
Allerdings sollte die Verzögerung auch nicht zu lange dauern. Ein rechtmäßiger Benutzer, der sich bei der Eingabe seines Passwortes nur vertippt hat, wäre mit einer Wartezeit von mehrere Minuten, um einen erneuten Versuch zu unternehmen, sicherlich nicht einverstanden. Zumal ein Angreifer den Überprüfungsvorgang nach dem Prinzip eines *Timeout* frühzeitig beenden könnte, wodurch die Maßnahme nur im begrenzten Umfang greifen würde.
Denkbar wäre auch, die Verzögerungszeit mit der Anzahl der Fehlversuche zu erhöhen, wodurch sich allerdings zusätzlich die Möglichkeit eines *Denial-of-Service*-Angriffs ergibt. Beispielsweise müsste bei einer Anfangsverzögerung von einer Sekunde und einer Verdoppelung der Wartezeit bei jedem Fehlversuch ein Angreifer nur 10 falsche Eingaben produzieren, um den rechtmäßigen Benutzer knappe 17 Minuten von seinem Zugang abzuhalten.

[4] Zur Ermittlung der Zeitangabe wurde die Dauer für die Abarbeitung einer Schleife (= Verschlüsselung des inkrementell veränderten Passwortes mittels der Betriebssystemfunktion `crypt` und Vergleich des Resultats mit Referenzwert) mit drei verschiedenen maximalen Durchlaufwerten (10 000, 100 000 und 1 000 000) mehrmals gemessen und daraus ein Mittelwert gebildet. Der Test wurde auf einem *standalone* Linux-PC mit einem Intel Pentium 166 MHz durchgeführt.

Ähnlich verhält es sich auch mit der radikalsten Version einer „Verzögerung“: nach einer festgelegten Anzahl von Fehlversuchen wird die Benutzerkennung einfach gesperrt. Anmeldevorgänge sind erst wieder möglich, nach einer Freigabe durch den Systemadministrator. Besonderes Augenmerk muss dabei dem *root*-Passwort gewidmet werden, das wegen den aus einer Kompromittierung resultierenden uneingeschränkten Systemrechten häufiges Angriffsziel ist. Eine Sperrung dieses Accounts würde über kurz oder lang einem Systemausfall gleichkommen, da dieses nicht mehr wartbar ist, und die *root*-Kennung auch nicht freigegeben werden kann.

Wie eben gezeigt, spielen die Passwortlänge und die verwendeten Zeichen eine entscheidende Rolle. Da der durchschnittliche Benutzer aber bekanntermaßen bequem ist, wird er selten diesen Forderungen nachkommen. Daher bedarf es gewisser Unterstützung von Seiten der Software. Die beim Festlegen des Passworts überprüfte Mindestlänge sollte eine Grundfunktion sein. Weiterhin wäre denkbar, das Passwort auf die darin verwendeten Zeichen hin zu untersuchen; zum Beispiel mindestens zwei Zahlen, ein Sonderzeichen und ein Großbuchstabe. Dabei muss aber bedacht werden, dass durch eine solche (allgemein bekannte) Restriktion der Suchraum aller möglichen Passwörter auch eingeschränkt wird.

Neben einer Mindestlänge wird bei Passwortsystemen häufig auch eine Maximallänge festgelegt. Dies ist jedoch aus sicherheitstechnischen Gesichtspunkten unzweckmäßig, denn je länger das Passwort ist, desto geringer ist die Möglichkeit eines erfolgreichen Brute-Force Angriffes. Der Hauptgrund für die feste Länge dürfte dabei auf programmtechnischer Seite liegen. Bei der Programmierung der Eingabemaske und für die Speicherung im System ist es einfacher, eine feste Größe an Speicher zu reservieren. Für eine feste Länge spricht auch, dass es die Auswahl eines Benutzers, gegen den der Angriff geführt werden soll, erschwert, wenn alle Benutzer einen gleichlangen Eintrag haben. Wäre beispielsweise ersichtlich, dass Alice ein Passwort mit 8 (unbekannten) Zeichen hat und Bob nur eins mit 4 Zeichen, erübrigt sich die Frage, gegen wen der Brute-Force Angriff gestartet werden soll.
Trotz dieser Überlegungen können dennoch unterschiedliche Passwortlängen verwendet werden. Dafür läßt man die Eingabe durch eine Einweg-Hashfunktion laufen, die einen beliebig langen Zeichenstrom auf einen zu speichernden Block fester Länge abbildet. Für die Eingabemasken könnten beispielsweise pauschal 256 Zeichen reserviert werden, was immerhin über drei Zeilen Text auf einem 80-Zeichen Bildschirm entspricht und einer vergleichbaren Schlüssellänge von 1 666 Bits gleichkommt.

Wörterbuchangriffe

Um den Aufwand eines Brute-Force Angriffes zu reduzieren, werden bei der Überprüfung möglicher Passwörter nur „wahrscheinliche“ Zeichenketten berücksichtigt. Da der Benutzer das Passwort sich merken können muss, verwendet er häufig real existierende Worte.
Geht man von der geschätzten Wortanzahl einer Sprache mit 150 000 aus, und nimmt man zudem noch geringfügige Veränderungen wie das Hinzufügen einer Ziffer am Anfang oder Ende des Wortes, Großschreibung oder Rückwärtsschreibung hinzu, ergeben sich 12 000 000 Zeichenketten.

Unter Verwendung der zehn gängigsten Sprachen, ergibt sich damit eine Gesamtanzahl von 120 000 000 Worten. Das sind gerade einmal 0,000 000 025% des ursprünglichen Passwortsuchraums aus dem vorherigen Abschnitt, also eine ganz erhebliche Einsparung bei den zu überprüfenden Zeichenketten. Zeitlich wären diese Worte in knappen 13,5 Stunden durchprobiert; mit einem 400 MHz PC wäre das Passwort im Mittel rein rechnerisch sogar in weniger als drei Stunden gefunden.

Gegenmaßnahmen sind recht einfach zu ergreifen: keine Wörter verwenden, die aus irgendeinem Wörterbuch stammen. Auch jegliche Art von Namen – seien es Spitznamen, Rechner- oder Betriebssystemnamen, der Name des Chefs oder des Haustiers –, Autonummern oder Geburtsdaten sind zu vermeiden, da sie leicht mit einem bestimmten Benutzer in Verbindung gebracht werden können.
In der Literatur gibt es viele Anregungen zur Wahl von guten Passwörtern. Da Zeichenfolgen, die keinem Wörterbuch entstammen, schwerer zu merken sind, ist ein Vorschlag die Bildung von Akronymen. Der Benutzer wählt einen für ihn leicht zu merkenden Satz und verwendet von den einzelnen Worten nur den ersten oder letzten Buchstaben (oder im Wechsel). Aus „To be, or not to be“ würde beispielsweise das Passwort `tbontb` abgeleitet; eine Buchstabenkombination, die so sicherlich in keinem Wörterbuch zu finden ist.
Bei der Bildung von Akronymen sollte man immer auch darauf achten, dass man nicht zufällig eine Buchstabenkombination erhält, die es bereits als gängige Abkürzung gibt. Die – zugegebenermaßen konstruierte – *Pass-phrase* „Das brennt mit Sicherheit“ (→ DBMS) wäre also nicht sehr vorteilhaft.

Wie auch in dem Fall der *Brute-Force* Angriffe können schwache Passwörter schon bei deren Festlegung von der Sicherheitskomponente durch den Abgleich mit hinterlegten Wörterbüchern erkannt und zurückgewiesen werden.
Der Umstand, dass durch das Nichtzulassen von ungeeigneten Passwörtern der ursprüngliche Suchraum eingeschränkt wird, beeinträchtigt die Sicherheit kaum, da die Einschränkungen vernachlässigbar gering sind und nach obiger Rechnung immer noch 99,999999975% der anfänglichen Zeichenfolgen zugelassen sind.

Replay- und Lauschangriffe

Diese Verfahren zeichnen sich dadurch aus, dass bei einem Anmeldevorgang die Authentifizierungsnachricht (Passwort) des Benutzers abgefangen/kopiert wird, um sie später zur Erlangung eines unrechtmäßigen Zugangs erneut an die überprüfende Instanz zu schicken. Dabei ist es irrelevant, ob die Nachricht verschlüsselt ist oder nicht. Erhält man ein unverschlüsseltes Passwort, kann man es direkt zur Anmeldung verwenden (Lauschangriff). Ist es dagegen verschlüsselt, braucht man sich nicht unbedingt die Mühe zu machen, es zu entschlüsseln, da das Paket als Ganzes, wie zuvor von dem unbescholtenen Anwender, an die Authentifizierungsinstanz geschickt werden kann (*replay*).

Gegenmaßnahmen. Eigentlich braucht man nur sicherstellen, dass die Authentifizierungsinformation nur für diesen einen Anmeldevorgang gültig ist.
Dazu kann man dem Passwort einen Zeitstempel befügen, der gewissermaßen eine Art Frischedatum darstellt, bis zu dem die Passwortinformation gültig ist. Darüberhinaus

muss natürlich gewährleistet sein, dass der Zeitstempel nicht aus der Nachricht entfernt oder verändert werden kann.

Zwei Probleme ergeben sich bei diesem Ansatz. Das Erste besteht darin, das Zeitfenster für die Gültigkeit des Passworts richtig zu dimensionieren: Ist es zu groß, sind weiterhin Replay-Attacken möglich; ist es zu klein, kann bedingt durch die erhöhten Übertragungszeiten in einem Netzwerk das Passwort fälschlicherweise als ungültig verworfen werden.
Das andere Problem liegt in der einheitlichen Zeit. Um zu ermitteln, ob ein Passwort noch gültig ist, muss der Zeitstempel mit der aktuellen Zeit verglichen werden. Diese holen sich die meisten Rechner aus ihrer internen Uhr, die systembedingt gewissen Schwankungen unterliegt. Dadurch haben selten zwei Rechner genau die gleiche Uhrzeit. Mitunter differieren diese sogar erheblich, wodurch es zu Fehlinterpretationen des erhaltenen Zeitstempels kommen kann: sie fallen rein rechnerisch aus dem Zeitfenster oder haben eine Absendezeit aus der Zukunft.

Spoof-Login

Wie schon in Abschnitt 2.2.4 angesprochen, wird bei dieser Art von Angriff dem Benutzer eine Bildschirmmaske angeboten, die der regulären Anmeldeprozedur völlig gleicht. Intern wird allerdings das im guten Glauben eingegebene Passwort (im Klartext!) an den Angreifer weitergeleitet und anschließend, damit die Attacke nicht auffliegt, die eigentliche Anmeldung vollzogen.
Solche *Spoof-Logins* können Kopien von HTML-Seiten sein, oder auch in das System eingeschleuste trojanische Pferde. Leichtes Spiel wird den Angreifern dabei durch das Internet und die Applet-fähigen Browser gemacht. Fremde Software wird schon beim Aufruf einer Seite heruntergeladen und ausgeführt, oft ohne dass der Benutzer davon Kenntnis nimmt.[5]

Gegenmaßnahmen. Gerade um der Gefahr von Java-Applets zu begegnen, warnt ein entsprechender Hinweis in der Titelleiste aller von einem Applet aus geöffneten Fenster. Die Nachbildung einer bekannten Anmeldemaske bekommt dadurch ein optisch störendes Element.
Allerdings ist diese Art von Schutz recht passiv. Sie verhindert nicht, dass ein sehr unachtsamer Benutzer troztdem in gutem Glauben sein Passwort verrät.

Das eigentliche Problem liegt auch vielmehr darin, dass bei den meisten Passwortverfahren zwar eine Authentifizierung des Benutzers gegenüber dem System erfolgt, der Benutzer aber nur vermuten kann, dass er sich gerade bei der von ihm gewünschten Instanz anmeldet.
Schon mit einer zusätzlichen Authentifizierung des Systems gegenüber dem Benutzer kann der Gefahr eines *Spoof-Logins* begegnet werden.

[5] Im Netscape Navigator 4.04 wird in der Fußzeile des Browser-Fensters beim Start eines Applets eine entsprechende Meldung ausgegeben. Sie verschwindet jedoch recht schnell wieder, so dass sie leicht übersehen werden kann. Es besteht jedoch über Einstellungen in den *Preferences* die Möglichkeit, die Ausführung von Java-Code generell zu unterbinden, wodurch aber auch erwünschte Programme nicht gestartet werden.

Off-line Angriff auf Passwortdatei

Damit das System eine Authentifizierung durchführen kann, muss es Zugriff auf einen Vergleichswert zu den Passwörten (im Allgemeinen die Passwörter selbst) haben. Diese werden häufig in einer Datei gespeichert. Da über diese Datei alle Benutzer kompromittiert werden können, stellt sie häufig das primäre Angriffsziel dar.
Sind die Passwörter in der Datei unverschlüsselt abgelegt, muss ein bedeutend größerer Aufwand für den Schutz der Datei selbst betrieben werden. Aber auch wenn sie darin verschlüssel stehen, ist die Gefahr nicht gebannt, da der Algorithmus für ihre Verschlüsselung als bekannt vorauszusetzen ist und durch einfaches Kopieren der Datei eine passive (off-line) Attacke auf den Passwörtern durchgeführt werden kann.

Gegenmaßnahmen. Neben einer Verschlüsselung des Passwortes kann und sollte zusätzlich eine Einweg-Hashfunktion darauf angewendet werden. Diese bildet beliebig lange Eingaben auf einen Block fester Länge ab. Damit können keine Rückschlüsse auf die Länge des ursprünglichen Passworts gemacht werden.
Zur Überprüfung der Authentifizierungsinformation wird das vom Benutzer eingegebene Passwort verschlüsselt und mit dem gespeicherten Referenzwert verglichen. Dabei sollte darauf geachtet werden, dass die Verschlüsselung auf der Client-Seite stattfindet (Passwort wird nicht im Klartext zum Server übertragen) und der Vergleich beim Server vollzogen wird, da der Client sonst dem Server eine erfolgreiche Überprüfung verspielen könnte.

Die Passwörter müssen nicht unbedingt in einer Datei gespeichert werden. Sie können auch in einer Datenbank abgelegt werden. Die Einträge können so zusätzlich durch die von dem Datenbanksystem angegebotenen Schutzmechanismen gesichert werden. Aber Vorsicht, auch ein Datenbanksystem arbeitet letztendlich auf Basis des Dateisystems. Mit etwas Mühe ist es möglich, die Einträge aus den Datensätzen der *tablespaces* herauszulesen, da diese bei gängigen Datenbanksystemen unverschlüsselt auf der Festplatte abgelegt werden. Allerdings sind diese Dateien, beispielsweise bei einem UNIX-System und im Gegensatz zu dessem Passwortdatei, vor dem Zugriff durch die Allgemeinheit geschützt.

5.3.2 Zusätzliche Vorkehrungen

Neben den eben besprochenen Gegenmaßnahmen können noch weitere Vorkehrungen getroffen werden, um ein wissensbasiertes System zu stärken.

Zeitpunkt der Anmeldung

Ähnlich wie bei den Zeitschaltuhren von Banktresoren, kann man Anmeldevorgänge nur zu bestimmten Zeiten zulassen, beispielsweise nur zu den Bürostunden.[6] Man geht dabei von der Annahme aus, dass jemand, der ausserhalb dieser Zeiten Zugang erlangen möchte, dem System gegenüber nicht freundlich gesinnt sein kann, da er sonst sein Vorhaben ja auch während der regulären Arbeitszeit erledigen könnte.

[6] Dies trifft nicht nur für die wissensbasierten Systeme zu; auch biometrische und tokenbasierte Systeme können sich dieser Einschränkung bedienen.

Ausnahmen sollte es für Administratoren geben, damit auch Aufgaben erledigt werden können, die während der regulären Betriebszeit nicht möglich sind (Software-Updates, Umbau des Netzes oder auch die Durchführung von Benchmarks).

Eingabe des Passwortes

Für den Fall, dass ein neugieriger Kollege einem über die Schulter sieht, sollte die Eingabe des Passwortes verdeckt erfolgen. Dabei bringt die häufig genutzte Maskierung der Eingabe durch das Sternsymbol (*) zwar den Vorteil, dass für die eingebende Person eine Kontrolle der gedrückten Tasten möglich ist, auf der anderen Seite verrät sie aber auch die Länge des Passwortes. Wie bereits in dem Abschnitt über die Brute-Force-Angriffe erläutert, können somit kurze Passwörter ausspioniert und bevorzugt angegriffen werden.

Änderung des Passwortes

Um zu Verhindern, dass in Ermangelung der Passwortkenntnis ein Angreifer einfach das Passwort eines Benutzer, der eben mal seinen Arbeitsplatz verlassen hat, um eine Tasse Kaffe zu holen,[7] auf einen neuen Wert setzt, muss der Änderungsprozess die Eingabe das alten (bestehenden) Passwortes erzwingen. Der rechtmäßige Benutzer würde die Änderung unter Umständen erst sehr viel später merken, nämlich dann, wenn er sich erneut bei dem System anmelden will. Diese Zeit reicht einem Angreifer oftmals schon für sein Vorhaben aus. Erfreulicherweise wird diese Vorgehensweise in heutigen System verfolgt. Fast immer wird auch eine Bestätigung (zweite Eingabe) des neuen Passwortes gefordert, um möglichen Tippfehlern (die Ausgabe des neuen Passwortes auf dem Bildschirm wird ebenfalls unterdrückt) vorzubeugen. Bei einer Falscheingabe des alten Passwortes oder keiner Übereinstimmung mit der Bestätigung wird der Änderungsvorgang abgebrochen.
Dies kann auch dann der Fall sein, wenn das neue Passwort nicht bestimmten Kriterien für ein „gutes“ Passwort genügt (vgl. Abschnitt 5.3.1).

Problematisch bei der systemseitig erzwungenen Änderung von Passwörtern nach Ablauf einer bestimmten Zeitspanne ist die Tatsache, dass der Benutzer aus Bequemlichkeit gerne sein altes Passwort wieder eingibt, wodurch der Zweck der Maßnahme umgangen wird. Daher ist darauf zu achten, dass wenn das System schon zeitlich begrenzte Passwörter unterstützt, auch eine Historie über alte Passwörter verwaltet wird, um ein solches Benutzerverhalten auszuschließen.
Manchmal ist es jedoch unumgänglich, dass Passwörter für bestimmte Benutzerkennungen in Programmcode oder Anwendungsprogrammen „hart verdrahtet“ eingegeben werden. Um die Funktionsweise dieser Programme über längere Zeit nicht zu beeinträchtigen, sollte die Zeitschaltung für Passwörter auch deaktivierbar sein.

[7] Es kommt auch häufig vor, dass eine Abmeldung bei dem System vergessen wird, sei es dass der Bildschirmschoner die Anzeige auf „Schwarz“ geschaltet hat, oder dass mal wieder vergessen wurde nach Beendigung des Window-Manager (OpenWindows) sich auch aus der Login-Shell zu verabschieden.

5.3.3 Einmal-Passwörter

Ein Hauptproblem bei der Authentifizierung liegt darin, dass für jeden Anmeldevorgang immer die gleiche Information überprüft wird. Was von Seiten des Benutzers oder der überprüfenden Instanz als Vorteil zu werten ist (der Benutzer muss sich nur diese Information merken bzw. zur Verfügung stellen; das System muss nur diese Information als Referenzwert speichern und von dem Anwender erfragen), erweist sich unter Betrachtung der gebotenen Sicherheit als potenzieller Schwachpunkt. Die immer wiederkehrende Übermittlung der Information während der Anmeldungsphase offenbaren einer feindlich gesinnten Person, die die Kommunikation auf dem Netz mitverfolgen kann, unterschiedliche Angriffsmöglichkeiten.
Wurde das übertragene Passwort nicht verschlüsselt, hat es der Angreifer am Einfachsten, er kann die Information direkt nutzen. Aber auch durch das Abfangen verschlüsselter Passwörter unterliegt der rechtmäßige Benutzer der Gefahr eines *Off-line* Angriffs oder einer Replay-Attacke.

Die Lösung des Problems scheint nur möglich, wenn für jeden Anmeldevorgang ein anderes, neues Passwort verwendet wird. Informationen die von einem Angreifer aufgeschnappt wurden, sind für die nächste Authentifizierung nicht mehr zu gebrauchen.

Eine naheliegende Realsierung wäre die Erzeugung einer Liste von Zufallswerten, die sowohl beim Benutzer als auch der Prüfinstanz (als Referenzwert) gespeichert wird, ähnlich einem One-time Pad (siehe Seite 70).

Auch das Anhängen eines Zeitstempels – die „klassische“ Maßnahme gegen Replay-Angriffe (vgl. S. 59) – machen die Passwortinformation bei jedem Anmeldevorgang einmalig.

Ein Verfahren, dass die Problematiken dieser beiden Ansätze umgeht, schlug Lamport (1981) vor. Die Speicherung der Listen wird durch eine einfache Berechnung der Passwörter vermieden, und das Problem einer synchronisierten verteilten Zeit bzw. der Zeitfensterdimensionierung (vgl. S. 60) wird in einer übertragenden Weise durch eine absteigende Zahlenfolge ersetzt.
Für das Verfahren wird die Hilfe einer Einweg-Hashfunktion benötigt, die sicherstellt, dass Passwörter, die zu einem späteren Zeitpunkt verwendet werden, nicht aus Informationen der vorhergehenden Anmeldevorgänge gewonnen werden können. Das Protokoll sieht im Einzelnen wie folgt aus:[8]

1. Es wird ein zufälliges Passwort p_0 erzeugt, im Folgenden auch als Kern bezeichnet.
2. Durch iterierte Anwendung der Einweg-Hashfunktion H werden n weitere Passwörter erzeugt: $p_i = H(p_{i-1})$ mit $1 \leq i \leq n$.
3. Der Anwender erhält die Passwörter p_0 bis p_{n-1}; bei der überprüfenden Instanz wird nur p_n gespeichert.
4. Zur Authentifizierung übermittelt der Benutzer das Passwort mit dem höchsten Index an die Prüfinstanz und streicht es danach aus seiner Liste (wodurch jedes Passwort nur einmal verwendet wird).

[8] Wegen seiner guten Schutzfunktion wird dieses Protokoll auch beim Online-Banking verwendet, dort heisst das Passwort allerdings Transaktionsnummer (TAN).

5. Die Prüfinstanz vergleicht, ob die Hashfunktion angewendet auf das von dem Benutzer erhaltene Passwort den gleichen Wert liefert, wie das gespeicherte Passwort. Stimmen sie überein, hat der Benutzer sich authentifiziert, und das System ersetzt den gespeicherten Referenzwert durch das vom Benutzer übermittelte Passwort

6. Sind alle n Passwörter verbraucht, erzeugt das System ein neues p_0.

Beispielsweise wird $n = 100$ gewählt. Aus dem zufälligen p_0 werden die Passwörter p_1 bis p_{100} generiert. Der Benutzer erhält p_0 bis p_{99}, auf dem Server wird lediglich p_{100} gespeichert. Möchte sich der Benutzer nun authentifizieren, schickt er p_{99} (sein Passwort mit dem höchsten Index) an den Server und streicht es aus seiner Liste. Der Server berechnet $H(p_{99})$ was entsprechend der Generierung p_{100} ergeben muss. Stimmen die Werte überein, hat sich der Benutzer authentifiziert und der Server ersetzt p_{100} durch p_{99}. Für den nächsten Authentifizierungvorgang übermittelt der Benutzer dann p_{98}, dessen Hashwert der Server mit dem zuvor erhaltenen Passwort p_{99} vergleicht, und so fort.

Wird n nicht zu groß gewählt, reicht es aus, bei dem Benutzer nur p_0 und den aktuellen Index i zu speichern, da aus diesen Werten das unter Punkt 4 benötigte Passwort schnell berechnet werden kann.

Es ist leicht ersichtlich, wie die Sicherheit des Verfahrens von der Güte der Einweg-Hashfunktion abhängt: Dadurch, dass die Passwörter entgegen der Reihenfolge, in der sie erzeugt wurden, angewendet werden, können für zukünftige Authentifizierungsvorgänge benötigt Passwörter nur aus der Umkehrung von H gewonnen werden: $p_{i-1} = H^{-1}(p_i)$ unter der Annahme, dass der Angreifer p_i (und alle davor benutzten Passwörter) abgefangen hat. Aber gerade dass eine solche Umkehrung nicht möglich ist, machen die Forderungen an eine Einweg-Hashfunktion aus. Je sicherer sie diesbezüglich ist, desto sicherer ist auch das Einmal-Passwortverfahren.

5.4 Zugangskontrolle in MEntAs

Eines der Hauptziele war die Plattformunabhängigkeit der Anwendung, damit eine möglichst große Anzahl von Benutzern erreicht werden kann. Allerdings erlaubt derzeit die Rechnerinfrastruktur keine weit verbreitete Überprüfung von physischen Eigenschaften. Sowohl das maschinelle Lesen von Körpermerkmalen als auch von beispielsweise Chipkarten erfordert zusätzliche Hardware. Die Kosten-Nutzen-Relation wäre eher ungünstig, zumal unter der Berücksichtigung obiger Betrachtungen schon bei der Verwendung von wissensbasierten Systemen ein durchaus akzeptabler Schutz gewährleistet werden kann.
Eine Identifikation über eine spezielle mit einem geheimen Schlüssel versehene Diskette scheidet ebenfalls aus, da es trotz einer weiten Verbreitung von Diskettenlaufwerken dennoch einige Maschinen ohne solche gibt (teilweise die Maschinen von Sun, Hewlett Packard und Silicon Graphics; letztgenannte wird aber sehr häufig im CAD-Bereich eingesetzt).

Diese Aspekte, sowie die in den vorhergehenden Abschnitten herausgearbeiteten Nachteile von biometrischen bzw. token-basierten Systemen und die Vorteile des wissens-

basierten Ansatzes, sprechen für eine Realisierung des Authentifizierungsmechanismus' durch Passwörter in der Intranet-Anwendung MEntAs.
Hinzu kommt noch die weitestgehende Benutzerakzeptanz von Passwortsystemen.

Um sich vor Replay-Attacken zu schützen, ist die Verwendung eines Einmal-Passwortsystems nach dem oben vorgestelltem Verfahren anzuraten.
Für den Benutzer gestaltet sich das Verfahren völlig transparent; er merkt keinen Unterschied zu einer Authentifizierung durch ein gewöhnliches Passwortsystem. Die konkrete Funktionsweise sieht wie folgt aus:

- Auf der Server-Seite ist das Passwort p_{i+1} sowie der Index i gespeichert. Der Index i bezeichnet das größte (im Sinne der Generierungsreihenfolge) noch zu keiner Authentifizierung verwendete (gültige) Passwort. p_{i+1} ist das bei der letzten Authentifizerung vom Benutzer übermittelte und zu Referenzzwecken gespeicherte Passwort.

- Der Benutzer gibt zur Authentifizierung wie bei einem normalen Passwortsystem seine Benutzerkennung und das persönliche, immer gleiche Passwort ein. Dieses Passwort bildet den Kern p_0.

- Die Client-Software erfragt nun bei dem Server den zu der Benutzerkennung gehörigen Index i. Dieser kann unverschlüsselt über das Netz geschickt werden, da er durch einen Angreifer, welcher den Netzverkehr abhören kann, durch einfaches Mitzählen sowieso ermittelt werden könnte. Einziger Punkt, der sichergestellt sein muss, ist, dass das übermittelte i auch wirklich von der Server-Komponente stammt. Ist dies nicht der Fall, wäre es möglich, dass ein Angreifer, der vorgibt der Server zu sein, einen immer um 1 verringerten Index an den Client weitergibt, um somit unter falschem Namen Zugriff auf den Server zu erhalten: Angenommen der aktuelle Index sei 5, der Angreifer übermittelt an den Client aber 4. Der Client schickt daraufhin p_4 an den vermeintlichen Server woraus dieser $p_5 = H(p_4)$ berechnet und sich damit gegenüber dem richtigen Server erfolgreich authentifizieren kann.

  Die Echtheit des Indexes i kann mittels eines Public-Key Systems erreicht werden. Dazu verschlüsselt der Server den Index i mit seinem nur ihm bekannten geheimen Schlüsselteil des Public-Key Paars. Entschlüsselt wird diese Nachricht mit dem in der Client-Software (Applet) enthaltenen öffentlichen Schlüssel.[9] Wie bereits erwähnt, ist die Nachricht nicht geheim, da sie von jedem, der das Applet hat – und damit auch den öffentlichen Schlüssel des Servers – entschlüsselt werden kann. Allerdings ist auf diese Weise sichergestellt, dass die Nachricht nur vom Server stammen kann, da nur er seinen geheimen Schlüsselteil kennt.

- Die Client-Software berechnet nun aus dem vom Benutzer eingegebenen p_0 durch iterierte Anwendung der Einweg-Hashfunktion das aktuelle Passwort p_i und übermittelt diesen Wert an den Server.

[9] Dass dieses Applet auch wirklich von dem authentischen Server stammt, kann über die Applet-Signierung (vgl. 33) garantiert werden.
Dieser öffentliche Schlüssel wird auch für die Datenübertragung zwischen dem MEntAs-Client und dem MEntAs-Server benötigt (vgl. Abschnitt 6.6.3).

- Der Server prüft $H(p_i) \stackrel{?}{=} p_{i+1}$. Bei Erfolg ersetzt er den gespeicherten Referenzwert p_{i+1} durch das eben erhaltene Passwort p_i und reduziert den Index i des Benutzers um 1.

- Unterschreitet i nach der Reduktion den Wert 1, wurden alle von dem Kern abgeleiteten Einmal-Passwörter aufgebraucht. In diesem Fall schickt der Server an den Client die Aufforderung, ein neues Passwort vom Benutzer zu erfragen.

- Eine benutzerinitiierte Änderung des Passwortes ist auch vor dem Aufbrauchen der n erzeugten Einmal-Passwörter möglich. Dazu erfragt die Client-Software von dem Benutzer ein neues p_0, berechnet daraus p_n und schickt diesen Wert an den Server. Der Server ersetzt in seiner Datenbank den alten Referenzwert einfach durch das neue p_n und setzt den Passwortindex auf $n - 1$.

Damit die Berechnung des aktuellen Passworts p_i einen akzeptablen Zeitrahmen, der für die Authentifizierung benötigt wird, nicht überschreitet, darf n nicht zu groß gewählt werden. Auch aus dem Umstand, dass ein Passwort von Zeit zu Zeit – und damit ist wirklich ein Zeitraum gemeint, und nicht die Anzahl der Anmeldevorgänge – gewechselt werden sollte, macht es wenig Sinn n sehr groß zu wählen. Meistens meldet sich ein Benutzer morgens an seinem System an und erst abends wieder ab. Würde man beispielsweise n auf 1000 setzen, wäre rein rechnerisch das Benutzerpasswort (der Kern p_0) fast drei Jahre gültig. Daher sollte in der Benutzerdatenbank auch der Zeitpunkt der letzten Passwortänderung (Speicherung von p_n) festgehalten werden, um nach einem bestimmten Zeitraum systeminitiiert von dem Benutzer die Änderung seines Passwortes zu verlangen.

Optional kann auch verhindert werden, dass ein neu eingegebenes Benutzerpasswort p_0 nicht den gleichen Wert wie ein schon vormals verwendeter Kern annimmt. Dazu müssen auf der Server-Seite die alten p_n zu Vergleichszwecken aufbewahrt werden. Da dem Server p_0 nicht bekannt ist, kann ein solcher Vergleich nur über diesen letzten gespeicherten Wert p_n erfolgen.

Bisher wurde noch nichts darüber gesagt, wo und wie das Referenzpasswort p_{i+1} und der Index i auf der Server-Seite gespeichert werden. Wie in Abschnitt 4.4 beschrieben, ist es notwendig, dass jeder Benutzer sich mittels einer eigenen Kennung bei dem Datenbanksystem anmeldet. Die Authentifizierung erfolgt dabei im vorliegenden Fall (DB2) über das Betriebssystem (UNIX). Es ist also eine „Transformation“ des übermittelten Einmal-Passwortes auf das Betriebssystempasswort (BSpwd) notwendig, da sich das übermittelte Passwort ja bei jedem Anmeldevorgang ändert. Es ist auch wünschenswert, dass das BSpwd nicht den gleichen Wert wie das eingegebene Benutzerpasswort p_0 annimmt, da sich sonst der Benutzer direkt bei dem Server und somit auch der Datenbank anmelden könnte. Ihm wären damit Änderungen an dem Datenbestand möglich, die unter Umständen zu Inkonsistenzen in der Datenbank führen könnten (vgl. Abschnitt 4.4).

Für die notwendige Umsetzung des Benutzerpasswortes auf das BSpwd können zwei Ansätze verfolgt werden:

- Sowohl der Index i als auch das Referenzpasswort des Servers werden neben der Benutzerkennung und dem BSpwd in der Datenbank abgelegt. Führte die Authentifizierung mit dem Referenzpasswort und dem Index i zum Erfolg, wird das

BSpwd des Datensatzes zur Anmeldung bei der Datenbank verwendet. Da der Server eine eigene Verbindung zur Datenbank unterhält, kann er die Werte in der Benutzertabelle nachschlagen.

Nachteil dabei ist, dass das BSpwd in dem System unverschlüsselt gespeichert wird und ein zusätzlicher Zwischenschritt eingeführt wird.

- Die zweite Möglichkeit umgeht dies, indem sie nur i in der Datenbank speichert und direkt das übermittelte Passwort als BSpwd verwendet. Da p_i das Resultat einer Einweg-Hashfunktion ist (beispielsweise MD5), die im Allgemeinen eine Ausgabe von 128 Bit erzeugt, UNIX aber nur Passwörter mit einer Länge von 8 Zeichen (64 Bit) zulässt, muss p_i entsprechend gekürzt werden. Dies kann erfolgen, in dem beispielsweise nur die ersten/letzten (oder eine andere Auswahl von) 64 Bit verwendet werden, oder aber das 128-Bit-Wort in zwei Hälften zerlegt wird, die miteinander XOR verknüpft werden.

  Natürlich muss man nach jeder erfolgreichen Authentifizierung des Benutzers das BSpwd zur Aktualisierung des Referenzwertes ändern (`passwd`-Befehl). Dazu benötigt der Server entweder root-Rechte, oder aber er muss sich selbst unter der Benutzerkennung bei dem System anmelden, um die Änderung vorzunehmen.

  Ein ganz gravierender Nachteil bei dieser Vorgehensweise ist allerdings, dass das aktuelle BSpwd immer der Hashwert des in der vorhergehenden Authentifizierungsnachricht übermittelten Passwortes ist. Ein Angreifer, der das Netz abhört,[10] kann somit nach der Hashwertberechnung des abgefangenen Passwortes direkten Zugang zum Server erhalten.

Unter Aspekten der gebotenen Sicherheit, stellt die erste Variante noch die bessere Alternative dar, da ein Angreifer, um in den Besitz des in der Datenbank gespeicherten Passwortes zu gelangen, erst einmal Zugang zum Server benötigt; nicht so beim zweiten skizzierten Szenario. Die Tabelle mit den Benutzerdaten kann durch das Datenbanksystem geschützt werden (via `grant`-Befehl) und die *tablespaces* der Datenbank, in denen das Passwort mit einem Hex-Editor aufspürbar wäre, wird durch das Betriebssystem geschützt (nur der Benutzer, der die Datenbank angelegt hat, hat Zugriff darauf).

5.5 Zusammenfassung

In diesem Kapitel wurden konkrete Vorgehensweisen für den Anmeldevorgang (Identifizierung und Authentifizierung) vorgestellt und auf deren Pro und Contra eingegangen. Besonders intensiv wurde der Passwortmechanismus behandelt, der bei der Regelung des Benutzerzugangs in MEntAs zum Einsatz kommt. Kernstück bildet dabei ein Einmal-Passwortsystem, das für den Benutzer transparent wie eine gewöhnliche Passwort-Authentifzierung abläuft.

[10] Genau davon geht man ja aus, wenn man Einmal-Passwörter einsetzt.

Kapitel 6

Übertragungssicherheit

Dieses Kapitel beschäftigt sich ihm Rahmen der Übertragungssicherheit mit der Authentifizierung sowie der Geheimhaltung von Nachrichten. Bekannte Verschlüsselungsalgorithmen als auch für den Web-Bereich relevante Sicherungsprotokolle werden neben Themen wie die Schlüsselerzeugung durch Pseudozufallsfolgengeneratoren sowie der Verwendung und Verwaltung von Zertifikaten besprochen. Abschließend wird auf die Übertragungssicherheit im konkreten Fall MEntAs eingegangen.

6.1 Einleitung

Durch die Offenheit des Internets (und im eingeschränkten Maße auch des Intranets) unterliegen die Daten während einer Übertragung in diesem nicht mehr dem Schutz der Rechner-Hardware oder einem Betriebssystem. Während der Übermittlung durchlaufen die Datenpakete eine Vielzahl von *store-and-forward* Knoten, bis sie bei ihrem Bestimmungsort angekommen sind. Auf dieser Strecke ist es möglich, den Inhalt der Pakete anzuschauen, zu verändern oder sogar die Pakete ganz verschwinden zu lassen – und alles, ohne etwas dagegen unternehmen zu können. Das ist auch gar nicht nötig, da es in erster Linie nicht um das Paket selbst, sondern um die darin enthaltenen Informationen geht. Diese können jedoch durch Verschlüsselung und Authentifizierung gesichert werden. Dadurch verhindert man zum Einen, eine ungewollte Einsichtnahme in die Daten, zum Anderen eine unbemerkte Veränderung der Daten. Darüberhinaus kann der Ursprung bzw. Absender der Daten zweifelsfrei festgestellt werden.

6.2 Verschlüsselungsalgorithmen

Wenn man die vielen selbstgestrickten Ansätze zur Verschlüsselung von Daten einmal unbeachtet lässt, gibt es immer noch eine fast unüberschaubare Anzahl von ernstzunehmenden, teilweise wissenschaftlich abgesicherten Verfahren, aus denen die Auswahl häufig schwer fällt. Dies zeigt sich auch durch die in den einschlägigen News-Gruppen `de.comp.security` und `sci.crypt` immer wieder aufkommenden Fragen, nach der besten Chiffre, über deren Beantwortung sich selbst die Experten nicht einige sind. Nach welchen Kriterien soll die Auswahl erfolgen? Wie kann man die Algorithmen bewerten?

Sicherlich ist die Geschwindigkeit des Algorithmus ein Kriterium. Auf sie richtet sich zwar nicht das Hauptaugenmerk, aber wie man am Beispiel von RSA sehen kann, trägt es doch zu der Entscheidung bei (vgl. S. 74).
Weitaus wichtiger hingegen ist die Sicherheit des Verfahrens, wo aber auch schon die Probleme einer Beurteilung beginnen. Nicht für alle Algorithmen kann man pauschal eine Aussage über ihre Sicherheit machen. Zu oft hängt sie von Faktoren wie der Schlüssellänge oder der Art des verwendeten Angriffs ab.

6.2.1 One-time Pads

Bis heute gibt es nur einen einzigen Algorithmus, der bewiesenermaßen absolute Sicherheit bietet. Es handelt sich dabei um die Vernam-Chiffre mit einem sogenannten *one-time pad* (OTP). Ihre Funktionsweise ist recht simpel: ein Kryptotextzeichen ergibt sich durch die modulare Addition von Klartextzeichen und Schlüsselzeichen. Folgendes Beispiel verdeutlicht dies (für die Addition gilt $\texttt{A} = 0, \ldots, \texttt{Z} = 25$; Modulofaktor 26)

	`mustereins`	(Klartext)
+	*yxgwpxpwkb*	(Schlüssel: One-time Pad)
=	`KRYPTOTEXT`	(Kryptotext)

Die absolute Sicherheit ergibt sich dabei aus der Verwendung eines One-time Pad als Schlüssel. Dabei handelt es sich um eine Zeichenfolge, die die gleiche Länge wie der Klartext hat und deren Zeichen völlig zufällig ausgewählt wurden.[1] Für einen Kryptoanalytiker, der nur den Kryptotext kennt, ergibt sich daraus das Problem, dass der Kryptotext `KRYPTOTEXT` aus obigem Beispiel auch unter Verwendung des Schlüssels *lvuhhubltc* aus dem Klartext `zweimuster` entstanden sein könnte. Selbst wenn ihm bestimmte Teile des Klartextes bekannt wären, wie zum Beispiel der Anfang der Nachricht in Form eines immer wiederkehrenden „Headers“, könnte der Analytiker daraus keinerlei Schlüsse auf die unbekannten Teile des Klartextes ziehen, da die Rekonstruktion der dazugehörigen Schlüsselteile keine Informationen über andere Stellen im One-time Pad verraten. Der Angreifer sieht sich also bei der Analyse des Kryptotextes dem Problem einer Gleichung mit zwei Unbekannten gegenüber – dem Klartext und dem One-time Pad –, für die es bekanntermaßen keine eindeutige Lösung gibt.
Hier wird auch ersichtlich, warum für die Erzeugung des One-time Pad kein (Pseudo-) Zufallszahlengenerator verwendet werden sollte. Bei ihm ist nicht gewährleistet, dass aus einer bereits bekannten Schlüsselbitfolge[2] keine weiteren Teile des Schlüssels berechnet werden können; es sind unter Umständen Vorhersagen für nachfolgende Bits mit einer Wahrscheinlichkeit von ungleich $1/2$ möglich.

Der Grund, warum die absolut sichere Vernam-Chiffre mit One-time Pad keine große Verbreitung gefunden hat, liegt in der aufwendigen Schlüsselverwaltung. Zum Einen müssen die Schlüssel wirklich zufällig sein, zum Anderen dürfen sie nur einmal verwendet werden, was bei einer regen Kommunikation eine sehr große Datenhaltung an

[1] In obigem Beispiel wurde der Schlüssel (zum besseren Verständnis) offensichtlich nicht zufällig ausgewählt, was aber keinen Einfluss auf gemachte Aussagen hat, da der dort verwendete Schlüssel genauso wahrscheinlich ist, wie jeder andere auch.
[2] Zur Durchführung der Verschlüsselung auf binären Daten verwendet man einfach das Alphabet $\mathcal{A} = \{0, 1\}$ und die Verknüpfung XOR, wodurch die Chiffre sogar involutorisch wird.

Schlüsseln mit sich bringen würde. Darüber hinaus muss sowohl der Sender, als auch der Empfänger im Besitz der One-time Pads sein; es ist also eine sichere Übertragung der Schlüssel erforderlich, deren Volumen dem aller später einmal abzusetzenden Nachrichten entspricht.

An dem Beispiel des One-time Pad ist gut ersichtlich, dass die Sicherheit eines Verfahrens nicht ausschließlich von der Länge des verwendeten Schlüssels abhängt. In diesem Fall, wo der Schlüssel seine maximale Länge annimmt, nämlich die des Klartextes, könnte man auch maximale Sicherheit vermuten. Dies trifft aber wie schon erwähnt nur dann zu, wenn der Schlüssel wirklich zufällig gewählt und vor allem nur einmal verwendet wurde. Nachrichten, die mit überlappenden oder mit komplett gleichen Schlüsseln chiffriert wurden, sind sehr leicht zu brechen.[3] Eine interessante Tatsache, wenn man bedenkt, dass der einzige bewiesene sichere Verschlüsselungsalgorithmus durch einen Chiffrierfehler nur noch eine kaum vorhandene Schutzwirkung hat.

6.2.2 Einmalschlüssel (*one-time keys*)

Aber nicht nur beim One-time Pad können Probleme entstehen, falls ein Schlüssel mehrfach verwendet wird. Selbst wenn ein Algorithmus dahingehend sicher ist, dass es bei Kenntnis mehrerer mit dem gleichen Schlüssel chiffrierter Kryptotexte nicht möglich ist, den Schlüssel „herauszurechnen“, besteht immer noch die Gefahr von Replay-Angriffen (vgl. S. 14).
Einzige Möglichkeit, um dies zu vermeiden, ist die Verwendung eines neuen, zufälligen Schlüssels für jede neue Datenübertragung. Dieser kann mit Hilfe eines Public-key Verfahrens dem Empfänger vor der Datenübertragung bekannt gegeben werden (vgl. Hybridverfahren im nächsten Abschnitt).

6.2.3 Symmetrische vs. asymmetrische Verschlüsselung

Die Vorteile asymmetrischer Verschlüsselung liegen auf der Hand: es kann eine Kommunikation zwischen Partnern stattfinden, die zuvor nie einen geheimen Schlüssel untereinander ausgetauscht haben. Demzufolge wird der Schlüsselaufwand bei einer exklusiven Kommunikation zwischen mehreren Parteien erheblich reduziert. Und schließlich kann man mit Public-Key Verfahren sogar Nachrichten signieren (siehe Abschnitt 6.4). Trotzdem verdrängen sie die symmetrischen Chiffren nicht. Die Ursache liegt hauptsächlich darin, dass aufgrund des großen Rechenaufwands von asymmetrischen Konzelationssystemen diese erheblich langsamer sind als ihre symmetrischen Konkurrenten. Beispielsweise ist RSA, der populärste Vertreter der Public-Key Verfahren, um den Faktor hundert langsamer als DES (bei einer Hardware-Implementierung sogar um tausend), im Vergleich zu IDEA um zweihundert (Schneier, 1997).
Aus diesem Grund wird häufig in Verschlüsselungs-Software eine Hybridform verwendet, die die Vorteile der beiden Verfahrensklassen vereinigt: Bei der Chiffrierung der (Nutz-) Daten wird ein (schnelles) symmetrisches Verfahren verwendet, dessen erheblich kürzerer Schlüssel mit Hilfe eines asymmetrischen Systems zwischen beiden Kommunikationspartner ausgetauscht wird. Mit ihm lässt sich ggf. auch eine Authentifizierung

[3] Schneier (1997) gibt in seinem Buch (S. 19) einen kurzen Hinweis, wie dies zu bewerkstelligen ist; vgl. auch Menezes, van Oorschot und Vanstone (1996, S. 21).

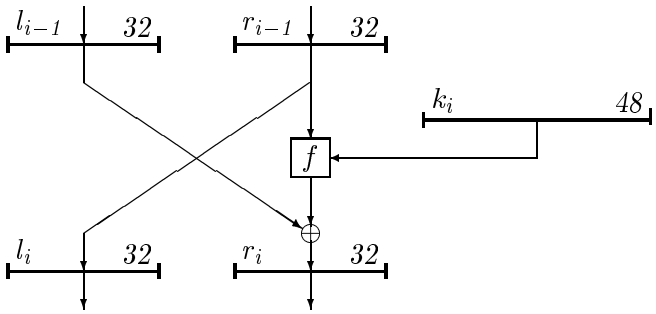
durchführen oder ein System mit Einmalschlüsseln (vgl. vorheriger Abschnitt) realisieren.

6.2.4 Data Encryption Standard (DES)

Der *data encryption algorithm* (DEA) wie der DES auch genannt wird, ist wohl die bekannteste und am meisten untersuchte (aber auch die älteste) der modernen Chiffren – und wohl auch die Chiffre, über die es die meisten Gerüchte gibt. Sie ist das Resultat einer 1973 erfolgten Ausschreibung des *National Bureau of Standards* (NBS)[4] für einen Verschlüsselungsstandard zur Verwendung bei nichtgeheimen U. S.-Behördenvorgängen. Der Vorschlag wurde von IBM unter dem Namen Lucifer eingereicht und (wegen mangelndem Know-How von Seiten des NBS) durch die *National Security Agency* (NSA) geprüft. 1993 wurde der Algorithmus in FIPS PUB 46-2 standardisiert.

Der Hauptkritikpunkt an dem Algorithmus durch die wissenschaftliche Öffentlichkeit waren die beiden Änderungen, die die NSA an dem Entwurf von IBM vorgenommen hatte: zum Einen wurde die Schlüssellänge von 128 Bit auf 56 verkürzt, zum Anderen wurden die S-Boxen – der Kern des Algorithmus – komplett ausgetauscht. Man mutmaßte, dass die NSA dadurch eine Hintertür eingebaut hatte.

Der Algorithmus verwendet wie erwähnt einen 56 Bit langen Schlüssel, der durch 8 Paritätsbits auf 64 Bit ergänzt wird, und operiert auf Klartextblöcken der Länge 64. Neben einer Eingangs- und Ausgangspermutation besteht die Chiffre aus 16 Iterationen eines Feistel-Netzwerkes (siehe Darstellung 6.1). Die Funktion f enthält neben einer Expansion und Permutation 8 Stück von den besagten S-Boxen, die jede für sich aus einer 6 Bit Eingabe eine 4 Bit Ausgabe erzeugt.



Darstellung 6.1: Ein **Feistel-Netzwerk**.

Wie in der Darstellung ersichtlich, ist die Funktion f für alle Iterationen die gleiche, nur die Schlüssel variieren. Sie werden unter Zuhilfenahme einer Eingangspermutation und der iterierten Anwendung einer weiteren Permutation sowie diverser Links-Shifts aus dem ursprünglichen Schlüssel erzeugt. Für eine detailierte Beschreibungen des DES sei auf Schneier (1997), Wobst (1998), Menezes, van Oorschot und Vanstone (1996) oder Köbler (1997) verwiesen.

Der DES wurde alle fünf Jahre durch das NIST als nationaler Standard bestätigt, letztmalig 1993. Seit November 1998 darf er nicht mehr in U. S.-Behörden eingesetzt werden.

[4] Heute heisst die Behörde *National Institute of Standards and Technology* (NIST).

Diese Entscheidung fiel nicht aus dem Grund, dass DES gebrochen wurde, sondern trug vielmehr dem Umstand Rechnung, dass wegen der immer schneller und billiger werdenden Hardware und der geringen Schlüssellänge eine exhaustive Schlüsselsuche in immer kürzerer Zeit bewerkstelligt werden kann.[5]
Bis der *advanced encryption standard* (AES)[6] – der Nachfolger von DES – bestimmt und als Standard verabschiedet ist, soll Triple-DES verwendet werden. Dabei handelt es sich um keine neue Chiffre, sondern um eine dreifache Hintereinanderausführung des DES-Algorithmus. Man unterscheidet folgende Betriebsmodi:

- DES-EEE3: Der Klartext wird dreimal mit drei verschiedenen Schlüsseln chiffriert.
- DES-EDE3: Der Klartext wird zuerst verschlüsselt, dann entschlüsselt und als letztes wieder verschlüsselt, wobei drei unterschiedliche Schlüssel verwendet werden.
- DES-EEE2 und DES-EDE2: Wie die beiden vorhergehenden Formate, nur dass die erste und die dritte Operation den gleichen Schlüssel verwendet.

Gegen die Zweischlüsselvarianten EEE2 und EDE2 haben Merkle und Hellman sowie Van Oorshot und Wiener bereits Attacken entwickelt, die allerdings wegen der großen benötigten Datenmenge unpraktikabel sind (RSA-FAQs v4.0, Question 3.2.6).

6.2.5 International Data Encryption Algorithm (IDEA)

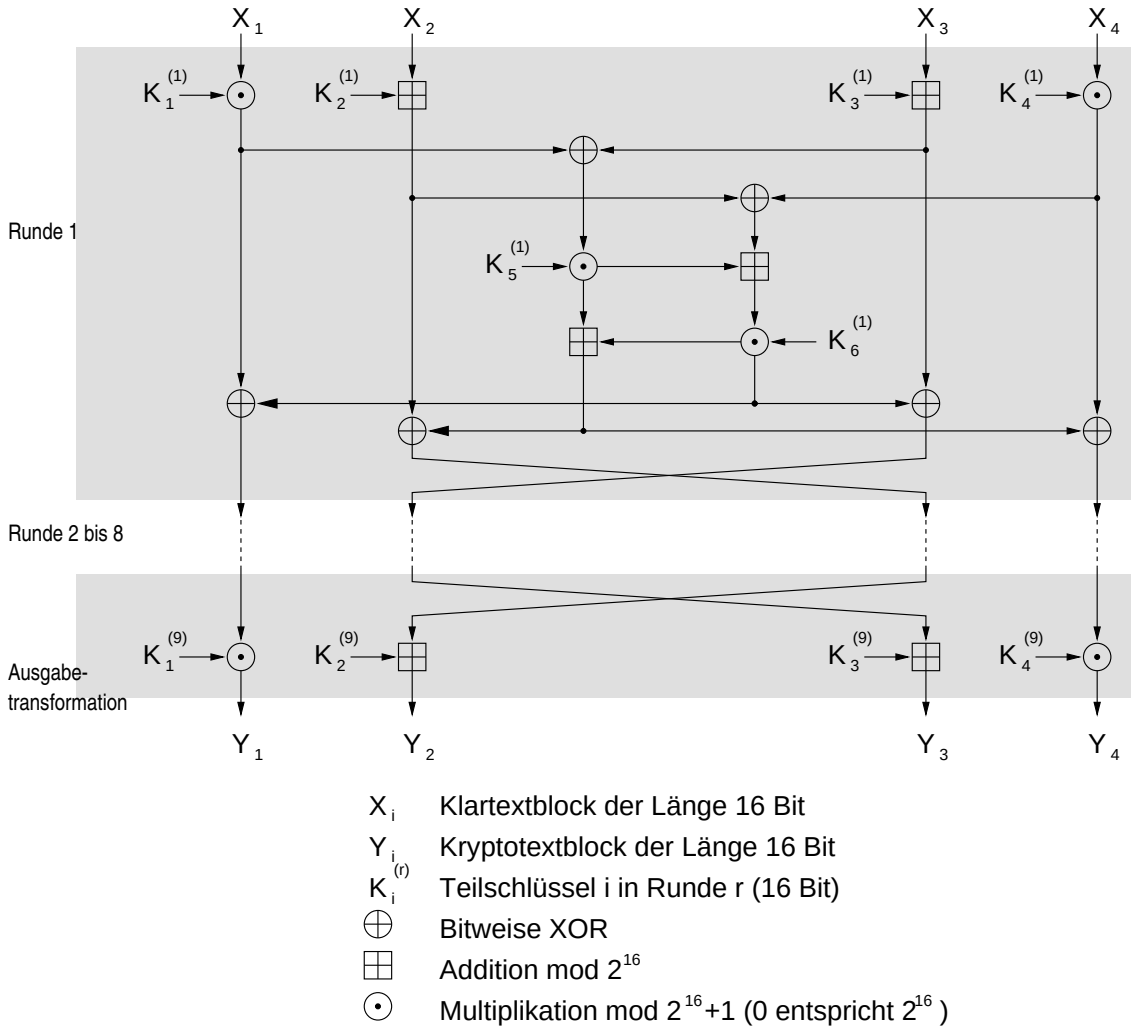
Eine andere symmetrische Blockchiffre ist der *international data encryption algorithm*, kurz IDEA, der hauptsächlich durch seine Verwendung in dem Verschlüsselungswerkzeug *Pretty Good Privacy* (PGP)bekannt wurde.
Nach Aussagen von Schneier stellt IDEA den derzeit sichersten öffentlich verfügbaren Blockalgorithmus dar.

IDEA operiert auf Blöcken der Länge 64 Bit mit einem Schlüssel von 128 Bit. Im Gegensatz zum DES verwendet die Chiffre keine Vertauschungen auf Bitebene sondern nur grundlegende arithmetische Operationen (XOR, Addition, Multiplikation) auf Teilblöcken der Größe 16, wodurch der Algorithmus auch noch auf 16-Bit Prozessoren recht effizient implementiert werden kann.

Darstellung 6.2 auf der nächsten Seite zeigt den schematischen Ablauf der Chiffre. Die Runden 2 bis 8 werden analog zur ersten durchgeführt.
Die 52 benötigten 16-Bit-Schlüssel ($8 \cdot 6 = 48$ für die acht Runden zuzüglich der 4 für die Ausgabetransformation) erhält man, indem man den 128-Bit-Schlüssel in Teilblöcke zu jeweils 16 Bit zerlegt (Teilschlüssel 1–8). Danach wird der Schlüssel um 25 Bits nach links rotiert und wieder in 8 Teilblöcke zerlegt (Teilschlüssel 9–16) usf.[7]

Es sei noch erwähnt, dass IDEA innerhalb Europas bis zum 16. Mai 2011 dem Patentschutz unterliegt. Eine unentgeltliche (kommerzielle) Nutzung kann also erst ab diesem Zeitpunkt erfolgen. Die Patentrechte liegen bei der Ascom-Tech AG (Schweiz).

[5] Einstiegspunkte für weitere Informationen zu den DES-Attacken sind `http://www.rsa.com/des/` (Kompromittierung des DES) und `http://www.rsa.com/rsalabs/des2/` (DES-Challenge).
[6] Weitere Informationen zu den AES-Bemühungen und den bisherigen Kandidaten sind unter `http://csrc.nist.gov/encryption/aes/aes_home.htm` abrufbar.
[7] Weitere Details zu dem Algorithmus können in Schneier (1997) oder Menezes, van Oorschot und Vanstone (1996) nachgelesen werden; letztgenannter Beitrag enthält auch einige Testvektoren.



Darstellung 6.2: Schematischer Ablauf des **IDEA** (nach Schneier, 1997).

6.2.6 RSA

Da Name dieses Public-Key Algorithmus ist von seinen Erfindern Rivest, Shamir und Adleman (1978) abgeleitet und wurde standardisiert in PKCS #1 (1993). Das Verfahren ist recht einfach (nach Wobst, 1998):

- Wähle eine Schlüssellänge N (laut Wobst sind derzeit 1024 Bit als sicher anzusehen).
- Wähle zwei Primzahlen p und q mit einer Bitlänge von $N/2$.
- Bilde $n = p \cdot q$.
- Wähle zufällig ein $e > 1$, das zu $\varphi(n) = (p-1) \cdot (q-1)$ teilerfremd ist.
- Berechne mittels des erweiterten Euklidschen Algorithmus[8] ein d mit $d \cdot e = 1 \bmod \varphi(n)$.

[8] Der Algorithmus ist teilweise auch unter dem Namen Berlekamp-Algorithmus bekannt und berechnet die Lineardarstellung des größten gemeinsamen Teilers (ggT) zweier nichtnegativer Ganzzahlen a und b: $\lambda \cdot a + \mu \cdot b = \mathrm{ggT}(a,b)$; der Algorithmus selbst ist beispielsweise in Menezes, van Oorschot und Vanstone (1996, S. 67) abgedruckt. Da $\mathrm{ggT}(e, \varphi(n)) = 1 = \lambda \cdot e + \mu \cdot \varphi(n)$ ergibt sich $d = \mu \bmod \varphi(n)$.

- e und n bilden den öffentlichen Schlüssel, d (und n) den privaten Schlüssel.
- *Verschlüsselung:* Nachdem der Klartext in Blöcke zu je $N - 1$ Bit zerlegt wurde, wird zu jedem Block mit dem Wert $m < n$ der Rest c von m^e bei Teilung durch n berechnet; c ist der Geheimtextblock und ist N Bit lang.
- *Entschlüsselung:* Zu jedem Block mit dem Wert $c < n$ ist der Rest von c^d bei Teilung durch n der zugehörige Klartext.

Zur Verdeutlichung ein Beispiel: Sei $p = 47$ und $q = 59$ sowie ein dazu teilerfremdes (zufälliges) $e = 17$. Daraus ergibt sich

$$n = p \cdot q = 2773$$

sowie

$$\varphi(n) = (p - 1) \cdot (q - 1) = 2668 \quad \text{und} \quad d = 157$$

Sei der Klartext die Zahl 2001, so ergibt sich der zugehörige Kryptotext zu

$$c = m^e \bmod n = 2001^{17} \bmod 2773 = 2029.$$

Die Entschlüsselung erfolgt analog:

$$m = c^d \bmod n = 2029^{157} \bmod 2773 = 2001.$$

Da das Zwischenergebnis der Potenzierung in der Regel recht groß wird (im Fall obiger Entschlüsselung war es 520 Dezimalstellen lang), durch die Modulo-Operation allerdings wieder erheblich „verkleinert“ wird, bedient man sich zur effizienteren Berechnung des Resultats von modularen Potenzen eines Verfahrens namens *wiederholtes Quadrieren und Multiplizieren*. Es sei $e = \sum_{i=0}^{r} e_i \cdot 2^i$ (mit $r = \lfloor \mathrm{ld}\, e \rfloor$) – die e_i bilden die Dualdarstellung des Exponenten e. Dann kann $c = m^e \bmod n$ recht einfach durch folgenden Algorithmus berechnet werden:

```
ModPot(m, e, n)
1  c ← 1
2  for i ← r downto 0
3      do c ← c² · m^(e_i) mod n
4  return c
```

Man vermutet, dass die Ermittlung des Klartextes aus dem öffentlichen Schlüssel äquivalent zum Problem der Faktorisierung von n ist (Wobst, 1998; bisher konnte dies aber noch nicht bewiesen werden). Könnte man n faktorisieren und somit in seine Bestandteile p und q aufspalten, wäre es ein Leichtes, das geheime d und damit auch den Klartext zu berechnen.
Da diese Faktorisierung für kleine n einfach ist, mit zunehmender Länge aber immer schwerer wird, sollte man n möglichst groß wählen. Ein Schlüssel mit 1024 Bit – was immerhin einer Zahl von über 300 Dezimalstellen entspricht – gilt bei den derzeit bekannten Faktorisierungsalgorithmen als sicher.

Jetzt ist auch ersichtlich, warum RSA im Vergleich zu einer symmetrischen Blockchiffre wie beispielsweise IDEA so langsam ist: die komplexen arithmetischen Operationen (Potenzieren) des Public-Key Systems mit so großen Zahlen sind wesentlich zeitintensiver als die einfachen, auf die Wortbreite des Prozessors zugeschnittenen Operationen der Secret-Key Chiffre.

6.3 Schlüsselerzeugung

Für die Generierung von Public-Key Schlüsselpaaren oder auch zur Erzeugung (symmetrischer) Sitzungsschlüssel sind häufig Zufallswerte erforderlich. Problematisch dabei ist, auf einem deterministischen System – das eine Rechenanlage ja sein sollte – eine unvorhersagbare Folge von Bits zu erzeugen (weswegen die generierten Werte auch als Pseudozufallsfolgen bezeichnet werden).

Zuerst sollte die Frage geklärt werden, was bedeutet überhaupt zufällig? Schneier hat in seinem Buch eine schöne Klassifikation von Zufallsfolgengeneratoren vorgenommen: Er unterscheidet zwischen Pseudozufallsfolgen, kryptographisch sicheren Pseudozufallsfolgen und echten Zufallsfolgen. Dabei stellt jede Klasse durch eine hinzugenommene Einschränkung eine Teilmenge der vorhergehenden dar.

Pseudozufallsfolgen
: Ein Generator ist pseudozufällig, wenn die von ihm erzeugten Folgen alle statistischen Zufallstests bestehen (wie die Anzahl der Nullen und Einsen ist ungefähr gleich oder etwa die Hälfte aller Folgen aus identischen Bits ist ein Bit lang, rund ein Viertel 2 Bit lang usw.).

Kryptographisch sichere Pseudozufallsfolgen
: Für kryptographisch sichere Pseudozufallsfolgen wird zusätzlich die Forderung gestellt, dass die Folge nicht vorhersagbar sein darf, insbesondere wenn alle zuvor erzeugten Werte bekannt sind. Dies kommt insbesondere dann zum Tragen, wenn man eine Pseudozufallsfolge als One-time Pad einsetzt und Teile des Schlüssels dem Angreifer bekannt sind.

Echte Zufallsfolgen
: Echte Zufallsfolgen sind nicht zuverlässig reproduzierbar. Wenn man den Generator zweimal mit exakt derselben Eingabe startet, erhält man zwei unterschiedliche Zufallsfolgen.

Wichtiger Kennwert eines Pseudozufallszahlengenerators ist seine Periode, das heißt die Anzahl Bits, nach deren Erzeugung sich die Folge wiederholt. Beispielsweise ist bei linearen Kongruenzgeneratoren – diese haben die Form $X_n = (a \cdot X_{n-1} + b) \bmod m$ – die maximale Periode m.
Der Wert X_0 wird Initialisierungs- oder Startwert (*seed*) genannt. Dieser sollte aus einer echten Zufallsquelle stammen. Da aber wie bereits erwähnt der Zufall in einem deterministschen System nicht gerade heimisch zu sein scheint, bedient man sich diverser Mechanismen, von denen man ausgeht, dass sie doch dem Zufallsprinzip unterliegen. Beispielsweise bezieht man Benutzerinteraktion mit ein, indem man Mausbewegungen in Zahlen transformiert oder aber (wie es bei der Erzeugung des Public-Key-Paars bei PGP der Fall ist) die Zeitintervalle zwischen dem Drücken zweier Tasten misst, während der Benutzer einen Text eingibt. Auch Verfahren, bei denen der *seed* aus der aktuellen Prozess-ID und der Systemzeit berechnet wird, werden eingesetzt (obwohl davon abgeraten wird; vgl. Garfinkel und Spafford, 1996).
Ein interessanter Ansatz ist, den nicht belegten Mikrophoneingang (Audio-Port) auszulesen, der in diesem Zustand ein gewisses „Rauschen“ produziert, was durchaus mit einer echten Zufallsquelle (wie der aus radioaktivem Zerfall) gleichzusetzen ist. Um

bessere Werte zu erhalten, kann die Folge zusätzlich noch komprimiert werden. (Eine Zufallsfolge kann auch so definiert werden, dass sie durch keine kürzere Folge als sie selbst beschrieben werden kann, sprich: sie kann nicht komprimiert werden. Und gerade eine solche Ausgabe sollte ein Komprimierungsalgorithmus liefern.)

6.4 Authentifizierung von Nachrichten

Wenn zwei Personen mit einem symmetrischen System chiffrierte Nachrichten austauschen, kann der eine Partner nach Erhalt einer solchen Nachricht mit hinreichender Sicherheit davon ausgehen, dass sie von seinem Kommunikationspartner stammt, sofern der verwendete Schlüssel alleinig den beiden Personen bekannt ist. Gegenüber einer dritten Person könnte aber nicht mit absoluter Sicherheit nachgewiesen werden, von wem eine Nachricht stammt, da beide Parteien im Besitz des gleichen Schlüssels sind, und damit jeder von ihnen die Nachricht verfasst haben könnte.
Noch schwieriger gestaltet sich der Herkunftsnachweis bei Verwendung eines Public-Key Systems, weil dort prinzipiell jeder in den Besitz des zur Verschlüsselung notwendigen öffentlichen Schlüssels gelangen kann (hier ist im Gegensatz zum symmetrischen System nur der Empfänger eindeutig identifiziert).

Betrachtet man einen aktiven Angreifer, der eine Nachricht abfängt und verändert, nicht als einen eigenständigen Absender, sondern eine manipulierende Zwischeninstanz, so kann man die Problematik der Integrität übertragener Daten in den Bereich der Nachrichtenauthentifizierung einordnen. Der Angreifer braucht bei seinem Vorgehen nicht einmal den Inhalt einer Nachricht zu kennen. Es reicht unter Umständen aus, wenn ihm die Position von bestimmten Informationen (beispielsweise von Geld- oder Zeitangaben) innerhalb der Nachricht bekannt sind. Da er die Ursprungsdaten der Nachricht und den Schlüssel der Chiffre nicht kennt, weiß er zwar nicht welchen alten und neuen Wert sein verändertes Datum darstellt, aber das ist bei Sabotageabsichten auch gar nicht notwendig. Unter Umständen bleibt somit trotz einer Plausibilitätsprüfung der Daten auf Seiten des Empfängers die Manipulation unentdeckt.

Anfällig für nachträgliche Änderungen sind vor allem Blockchiffren, bei denen die einzelnen Blöcke nicht miteinander verknüpft worden sind, und sich somit der Austausch von einzelnen Blöcken nicht auf die übrigen Teile der Nachricht auswirkt. Dies ist insbesondere bei One-time Pad-Chiffren der Fall.[9]

Wünschenswert wäre also zum Einen ein Mechanismus, mit dem man die Herkunft einer Nachricht eindeutig nachweisen kann, ähnlich einer handgetätigten Unterschrift, zum Anderen eine Möglichkeit die Unterschrift fest an das Dokument und ihren Inhalt zu binden, um nachträgliche Änderungen zu erkennen.
Durch die letzte Forderung scheiden naive Vorgehensweisen, wie das Scannen der Unterschrift und Anhängen an die Nachricht in Form eines Bildes, aus. Dieses Bild kann nur zu leicht kopiert und dann unter beliebig viele Nachrichten falscher Herkunft bzw.

[9] Man mag etwas verwundert sein, dass gerade eine informationstheoretisch absolut sichere Chiffre anfällig gegen nachträgliche Manipulationen ist. Dabei sollte man aber bedenken, dass diese Sicherheit sich nur auf die Geheimhaltung der in der Nachricht enthaltenen Informationen bezieht. Geheimhaltung und Schutz vor Veränderungen sind zwei unterschiedliche Aufgabenbereiche und müssen damit auch durch unterschiedliche Mechanismen realisiert werden.

falschen Inhalts gesetzt werden; ein generelles Problem von elektronischen Dokumenten, die letztendlich nur eine Folge von Bits sind, welche (ohne entdeckt zu werden) verändert oder kopiert werden können. Im folgenden Abschnitt wird eine Möglichkeit vorgestellt, eine digitale Signatur dennoch zu realisieren.

Digitale Signaturen

Public-Key Verfahren haben die Eigenschaft, dass eine mit dem geheimen Schlüssel chiffrierte Nachricht nur mit dem zugehörigen öffentlichen Schlüssel wieder entschlüsselt werden kann. Diese Funktionen sind häufig kommutativ, was man sich zur Realisierung einer Form von elektronischer Unterschrift zunutze machen kann (vgl. auch Rivest, Shamir und Adleman, 1978). Dazu verschlüsselt man die Nachricht mit dem geheimen Schlüssel, den nur der rechtmäßige Verfasser der Nachricht kennt. Die Echtheitsüberprüfung kann dann von jedem vorgenommen werden, indem er den Kryptotext mit dem zugehörigen öffentlichen Schlüssel dechiffriert (und eine gültige Nachricht erhält).
Man beachte, dass es sich bei der „Verschlüsselung“ nicht um eine Chiffrierung mit Geheimhaltungsfunktionalität handelt, da jeder mit dem allgemein verfügbaren öffentlichen Schlüssel die Nachricht wieder entschlüsseln kann. Durch den Vorgang wird nur sichergestellt, dass der Absender den geheimen Schlüssel des angegebenen Absenders kennt (und damit authentisch ist). Aus diesem Grund, und da wie schon erwähnt asymmetrische Kryptosysteme recht langsam sind, wird nicht die ganze Nachricht, sondern nur ihr Hashwert (*message digest*) mit dem geheimen, asymmetrischen Schlüssel signiert. Das Resultat wird an das Ende der übertragenen Nachricht angehängt und bildet die Unterschrift für das Datenpaket.
Im Gegensatz zu einer gewöhnlichen, handschriftlichen Signatur, hat das elektronische Verfahren damit die Vorteile, dass

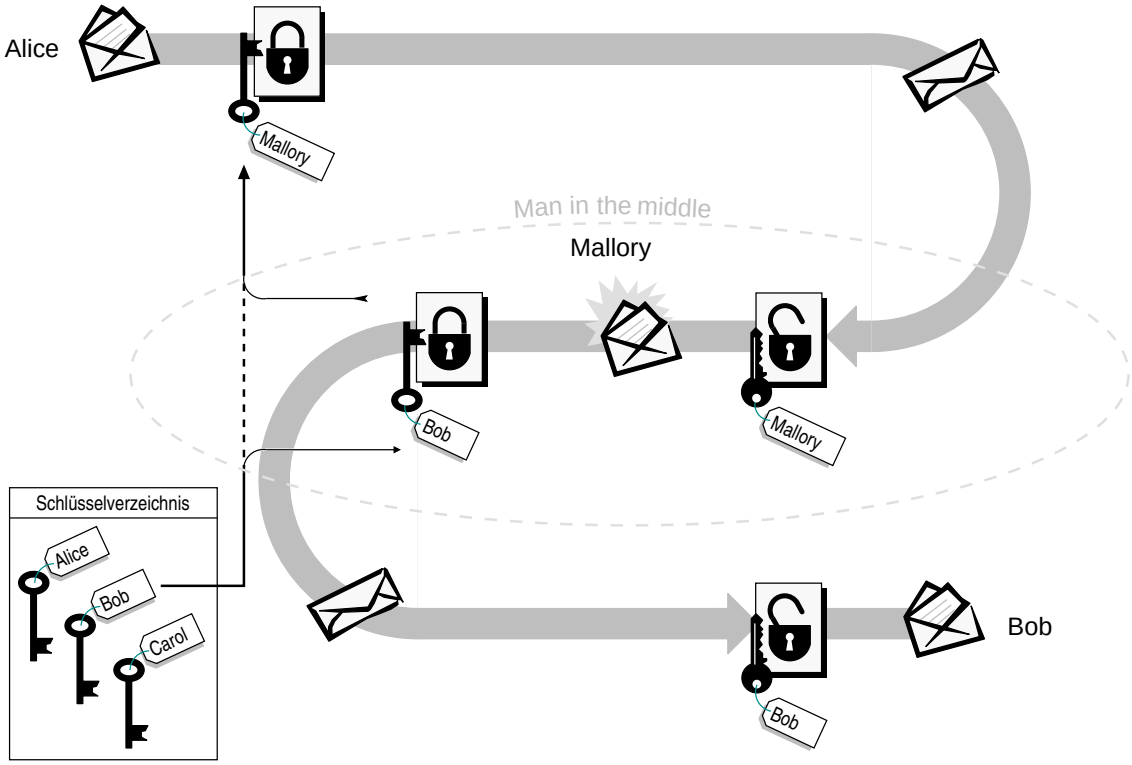
- die Unterschrift nur von dem rechtmäßigen Besitzer erzeugt werden kann.
- von jedem die Echtheit der Unterschrift problemlos nachgeprüft werden kann.
- die Unterschrift fest an die Nachricht gebunden ist: Weder können nachträglich (unentdeckt) Veränderungen an der Nachricht vorgenommen werden, noch kann die Signatur kopiert und unter eine andere Nachricht gesetzt werden.

Ausserdem ist es nicht zwingend erforderlich, die digitale Signatur unmittelbar bei der Nachricht selbst zu speichern. Da sie auf eine logische Art und Weise mit der Nachricht verknüpft ist, könnte sie auch an einem anderen Ort aufbewahrt werden.

Zertifikate

In vorigem Abschnitt wurde beschrieben, wie man mit Hilfe von digitalen Unterschriften die Authentizität von Nachrichten sicherstellen kann. Das funktioniert allerdings nur, wenn der zur Überprüfung verwendete öffentliche Schlüssel auch wirklich von der Person stammt, die behauptet, der Absender der Nachricht zu sein. Diese Aussage gilt nicht nur für den Einsatz digitaler Signaturen, sondern trifft generell auf alle Verfahren zu, bei denen asymmetrische Kryptosysteme eingesetzt werden und wo öffentliche Schlüssel publik gemacht werden müssen. Da häufig die Schlüssel aus einem allgemein

zugänglichen Schlüsselverzeichnis geholt werden, besteht die Gefahr eines *Man-in-the-middle*-Angriffs.[10] Dieser würde am Beispiel einer Nachrichtenverschlüsselung wie folgt aussehen (siehe Darstellung 6.3).



Darstellung 6.3: **Man-in-the-middle Angriff**. Mallory vertauscht auf eine Anfrage von Alice den öffentlichen Schlüssel von Bob mit einem eigenen, von ihr (Mallory) erzeugten Public-Key. Sie fängt die mit ihm verschlüsselte Nachricht ab, entschlüsselt sie mit dem zugehörigen geheimen Schlüssel und schickt sie verschlüsselt weiter, diesmal mit dem regulären öffentlichen Schlüssel von Bob, um dadurch ihren Angriff zu vertuschen.

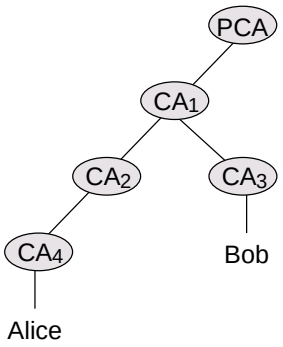
Man kann erkennen, dass der öffentliche Schlüssel zwar nicht geheimgehalten werden braucht, eine eindeutige Zuordnung zu der Person allerdings gewährleistet sein muss. Diese Zuordnung wird von einer Zertifizierungsinstanz (*certificate authorities*, CA) vorgenommen. Dazu stellt sie jedem Benutzer ein Zertifikat aus, das neben einer Gültigkeitsdauer, dem verwendeten Signatur-Algorithmus, Informationen über den Aussteller und einigen weiteren Feldern auch die eindeutige Kennung des Inhabers sowie dessen öffentlichen Schlüssel enthält. Ein häufig eingesetzter Standard für Zertifikate ist X.509 (ITU-T, 1997).

Die CA wird als eine vertrauenswürdige Instanz betrachtet, d. h. wenn mit derem öffentlichen Schlüssel ein von ihr ausgestelltes Zertifikat authentifiziert wurde, geht man von der Echtheit des in dem Zertifikat enthaltenen öffentlichen Schlüssels aus.

In der Praxis werden nicht alle Zertifikate von einer CA vergeben. Dies liegt zum Einen an der großen Anzahl Benutzer, die alle unmöglich von einer Zertifizierungsstelle

[10] Man kann dies auch als einen *Impersonation*-Angriff betrachten, bei dem der Angreifer vorgibt, die Schlüsselverzeichnisinstanz zu sein. Dadurch wird es ihm erst ermöglicht, in die Kommunikation der beiden Partner einzugreifen.

verwaltet werden können (man denke nur an die Anzahl der Internet-Benutzer). Zum Anderen liegt es aber auch an dem Grad der Vertrauenswürdigkeit, die eine CA offerieren möchte.
Eine Zertifizierungstelle kann die Überprüfung einer Identität für die Zuordnung von Zertifikat und Benutzer mit unterschiedlicher Intensität vollführen; wie streng diese Prüfung ausfällt, wird in der öffentlich verfügbaren Zertifizierungs-Policy festgelegt und bestimmt damit den Grad des Vertrauens, das in ein von dieser Instanz ausgestelltes Zertifikat gelegt werden kann. Ein naheliegender Vergleich wäre der mit einem herkömmlichen Ausweis: Ein Personalausweis ist sicherlich vertrauenswürdiger einzustufen als ein Schülerausweis (vgl. Platzer, 1996).



Darstellung 6.4

Damit einem Benutzer zur Authentifizierung eines Zertifikates nicht der öffentliche Schlüssel jeder benutzerausstellenden CA auf sicherem Wege übermittelt werden muss, sind diese meistens hierarchisch angeordnet (siehe nebenstehende Darstellung). Die oberste Instanz bildet die *policy certification authority* (PCA), welche gewissermaßen die Mindestanforderungen der Zertifizierungs-Policy des gesamten Baumes vorgibt und ihrerseits nur Zertifikate für andere CAs ausstellt. Sie ist die einzige Instanz, die sich ihr Zertifikat selbst ausstellt (signiert); die übrigen Knoten stellen jeweils für ihre Söhne Zertifikate aus.

Durch diese Hierarchie bedingt, benötigt der Benutzer nur den sicher übermittelten öffentlichen Schlüssel (das Zertifikat) *einer* CA, welcher abhängig von der Strategie zur Zertifikatsüberprüfung entweder der der eigenen CA (*bottom-up* Strategie) oder von der PCA (*top-down* Strategie) ist.

***Top-down* Strategie**

Der einfachere Fall von beiden besteht darin, dass jeder Benutzer nur den öffentlichen Schlüssel der PCA besitzt. Will Alice beispielsweise das Zertifikat von Bob überprüfen, muss sie sich nur die Zertifikate aller oberhalb von Bob liegenden Zertifizierungsinstanzen besorgen (CA_1 und CA_3) und diese von oben nach unten authentifizieren.

***Bottom-up* Strategie**

Bei diesem Ansatz wird vorausgesetzt, dass jede Zertifizierungstelle zusätzliche zu der darunterliegenden auch ihre darüberliegende CA signiert hat. Damit kann Alice, die nur im Besitz des öffentlichen Schlüssels ihrer eigenen Zertifizierungsinstanz (CA_4) ist, erst aufsteigend die Zertifikate von CA_2, CA_1 und dann absteigend von CA_2 anfordern und authentifizieren.

Wichtiger Punkt einer Zertifizierungsinstanz ist, dass sie auch die Möglichkeit haben muss, Zertifikate zu wiederrufen, sei dies aus dem Grund, dass ein privater Schlüssel kompromittiert wurde oder ein Benutzer einfach aus dem Arbeitsverhältnis seiner Firma ausgeschieden ist und somit keinen Zugriff mehr auf bisher ihm obliegende Daten erhalten darf. Daher führen die CAs neben dem Zertifikatverzeichnis auch immer noch eine Liste ungültiger Zertifikate (*certificate revokation list*; CRL), die bei der Überprüfung eines neuen Zertifikats konsultiert werden sollte.

Einen nicht zentralistischen Ansatz zur Authentifizierung von öffentlichen Schlüsseln wählte Phil Zimmermann bei dem von ihm entwickelten Verschlüsselungswerkzeug PGP – er nannte ihn *web of trust*.
Dabei wird die Zuordnung von Identität und öffentlichem Schlüssel nicht durch die CA geregelt, sondern jeder kann sich seinen Public-Key selber erzeugen. Das Vertrauen, das in einen solch erhaltenen Schlüssel gelegt wird, bleibt jedem selbst überlassen. Der Benutzer wird nur durch gewisse Rahmenbedingungen bei seiner Entscheidung unterstützt.

Jeder öffentliche Schlüssel, oder genauer jedes Zertifikat wird mit zwei zusätzlichen Parametern im Schlüsselring[11] des Benutzer gespeichert: Gültigkeit (*validity*) und Vertrauen (*trust*).
Die Gültigkeit ist ein Indikator dafür, ob man glaubt, dass der vorliegende Schlüssel wirklich der angegebenen Person gehört. Dies wird hauptsächlich durch die Umstände bestimmt, durch die man in den Besitz des Schlüssel gekommen ist. Wurde er beispielsweise persönlich auf einer Diskette überreicht, so kann man mit hoher Wahrscheinlichkeit von dessen Authentizität ausgehen. Aber auch für auf einem unsicheren Kanal erhaltene Schlüssel (zum Beispiel aus dem Web) besteht die Möglichkeit, dessen Gültigkeit hinreichend sicher zu bestimmen, indem er zum Beispiel mit dem Hashwert auf der persönlich überreichten Visitenkarte des Benutzers verglichen wird.
Dagegen ist der Parameter Vertrauen ein Maß dafür, wie sehr man an die Rechtschaffenheit und das Urteilsvermögen der Person glaubt, die einen Schlüssel erzeugt hat – je mehr man diesem Schlüssel (also der Person) vertraut, desto größeres Vertrauen bringt man der Person bei der Zertifizierung anderer Leute Schlüssel entgegen (Garfinkel, 1995). Hat beispielsweise Bob den Schlüssel von Carol signiert – er bestätigt damit die Echtheit des Schlüssels – und gibt Bob diesen Schlüssel an Alice weiter, wobei Alice uneingeschränktes Vertrauen in Bob hat, da sie mit diesem schon seit langem befreundet ist und beide sich gut kennen, kann Alice ebenfalls von der Echtheit des Schlüssels ausgehen, da Bob bereits dessen Authentizität für sie überprüft hat.

6.5 Übertragungsprotokolle

Im Bereich der verteilten Systeme, insbesondere des Internets, gibt es bereits einige Protokolle, die Methoden zur Unterstützung einer gesicherten Übertragung anbieten. Drei der bekanntesten seien hier herausgegriffen.

6.5.1 IPsec und IPv6

IPsec[12] ist ein kryptographisches Protokoll, das von der Internet Engineering Task Force (IETF) entwickelt wurde, um end-zu-end Vertraulichkeit von durch das Internet übertragenen Paketen zu realisieren (Garfinkel und Spafford, 1997). Es stellt gewissermaßen eine Erweiterung zu IPv4, der Standardversion des heutigen Internets, dar. In dem

[11] Bei der Nutzung von PGP werden im Arbeitsverzeichnis des Benutzers zwei sogenannte Schlüsselringe angelegt; der eine enthält alle öffentlichen Schlüssel (also alle erhaltenen Zertifikate und die öffentliche Hälfte des eigenen Public-Keys), der andere alle privaten Schlüssel (meist die geheime Hälfte der selbst erzeugten Public-Keys).
[12] Die Literatur zu IPsec und IPv6 besteht hauptsächlich aus RFC-Beiträgen und Internet Drafts: RFC 1752, RFC 1825, RFC 1883, Pereira und Bhattacharya (1998).

kommenden IPv6, die „next generation“ von IP (daher auch häufig mit IPnG bezeichnet), ist neben einigen anderen Neuerungen (wie zum Beispiel 128 anstatt 32 Bit langen Adressen) IPsec bereits enthalten.

IPv6 sieht keine speziellen kryptographischen Algorithmen vor, es bietet vielmehr *Header* zur Spezifikation des Dienstes und der Algorithmen für das zu übertragende Datagram an. Dabei werden prinzipiell zwei Dienste/*Header* unterschieden: *IPnG Authentication Header* (RFC 1826, 1995) und *IPnG Encapsulation Security Header* (RFC 1827, 1995). Mit ersterem kann die Authentizität und die Integrität der Daten gewährleistet werden, während mit dem zweiten die Vertraulichkeit der Daten garantiert wird. Trotz der Algorithmen-Unabhängigkeit werden im Zuge einer internet-weiten Interoperabilität MD5 und DES CBC als Standardalgorithmen empfohlen.

Durch die Ansiedlung der Sicherheitsfunktionen in einer der untersten Schichten des OSI-Basisreferenzmodells (OSI-RM)[13] – genauer: in der Netzwerkschicht – können fehlende Datenschutzmechanismen in den höheren Schichten, insbesondere der Anwendungsschicht, ausgeglichen werden.

6.5.2 Secure Socket Layer (SSL)

SSL wurde von Netscape Communications Corp. für die Verwendung in deren Web-Browser (Netscape Navigator) entwickelt.[14] Das Protokoll stellt eine Schicht zwischen dem reinen TCP/IP Protokoll und der Anwendungsschicht dar. Während das gewöhnliche TCP/IP Protokoll nur eine fehlerfreie Übertragung zwischen zwei Systemkomponenten erreicht, garantiert SSL durch Authentizität, Vertraulichkeit und Integrität der Paketdaten einen sicheren Kanal (vgl. Garfinkel und Spafford, 1997).

Da SSL auf die unter Windows und UNIX verwendete Socket-Schnittstelle des TCP-Protokolls aufsetzt (indem es diese durch eine neue Version ersetzt; vgl. Platzer, 1996) kommen auch alle anderen auf TCP/IP basierenden Protokolle wie beispielsweise FTP, *telnet* oder HTTP[15] in den Genuss der Sicherheitsfunktionen.

Der Ablauf einer Übertragung erfolgt in zwei Schritten. Während des ersten wird das sogenannte *Handshake*-Protokoll durchgeführt (siehe unten). Mit diesem Protokoll einigen sich die beiden Kommunikationspartner über die bei der Verbindung zu verwendenden Parameter. Diese Parameter werden in dem *Status* gespeichert und für den zweiten Schritt – die eigentliche Übertragung – verwendet.
Diese sieht so aus, dass der Sender seine Daten in Blöcke zerlegt. Jeder Block erhält eine Nummer und eine mittels einer Einweg-Hashfunktion berechnete Prüfsumme. Das Resultat wird mit einem symmetrischen Kryptosystem verschlüsselt. Welche Hashfunktion und welcher Schlüssel bzw. welche Chiffre verwendet wird, ist dem Status zu entnehmen. Zum Lesen der Nachricht geht der Empfänger in entsprechend umgekehrter Reihenfolge vor.

Das Handshake zwischen Client und Server beginnt mit einem Kommunikationswunsch des Client, der eine Liste unterstützter Kompressions- und Verschlüsselungsalgorithmen

[13] siehe dazu Informatiktaschenbuch, 1995, S. 239 ff.
[14] Die Spezifikation wurde in einem Internet-Draft (Freier, Karlton und C., 1996) veröffentlicht.
[15] Im Gegensatz zu einer normalen HTTP Verbindung, die auf Port 80 geführt wird, wird beim Einsatz von SSL der Port 443 verwendet; der zugehörige URL-Typ ist `https`.

an den Server schickt. Der Server wählt aus der Liste für ihn genehme Algorithmen aus (Kompressionsmethode, symmetrisches und asymmetrische Kryptosystem sowie ein symmetrisches Hashverfahren) und teilt sie dem Client zusammen mit seinem Zertifikat mit (eine Client-Authentifizierung kann optional erfolgen). Der Client erzeugt nun einen Sitzungsschlüssel und lässt diesen dem Server mit dessem öffentlichen Schlüssel chiffriert (ist im Zertifikat enthalten) zukommen. Ausserdem teilt er dem Server mit, dass ab sofort nur noch verschlüsselt kommuniziert wird, da beide Seiten im Besitz aller dafür notwendigen Informationen sind.

Es gibt zwei voneinander unabhängige Referenzimplementierungen des Protokolls: *SSLRef* von Netscape und *SSLeay* von dem Australier Eric Young (letzteres frei erhältlich per FTP).

6.5.3 Secure Hypertext Transfer Protcol (S-HTTP)

S-HTTP (Rescorla und Schiffman, 1996b) wurde entworfen, bevor SSL öffentlich bekannt gegeben wurde und arbeitet im Gegensatz zu diesem (und dem vorgestellten IPnG) in der obersten Schicht des OSI-RM, der Anwendungsschicht.

S-HTTP stellt eine Ergänzung des HTTP-Protokolls um kryptographische Funktionalitäten dar. Genaugenommen wird die HTTP Nachricht gekapselt in einer S-HTTP Nachricht verschickt. Dazu mussten neue Elemente für HTML definiert werden (Rescorla und Schiffman, 1996a).
Mit diesen werden unter anderem die Parameter für die Kryptoalgorithmen festgelegt, beispielsweise die Art der Schlüsselfestlegung, der verwendete Verschlüsselungsalgorithmus oder eine Auswahl unterstützter Chiffren (für die Rückantwort). Dies kommt gewissermaßen einer Aushandlung der (kryptographischen) Rahmenbedingungen für die Übertragung gleich.

Für jede S-HTTP Nachricht besteht die Möglichkeit Verschlüsselung, digitale Unterschrift oder Authentifizierung zu aktivieren.
Der Schlüsselaustausch zwischen Sender und Empfänger kann auf vier verschiedene Arten erfolgen: *RSA*, *outband*, *inband* und *Kerberos*. –
RSA entspricht dem weiter oben in diesem Kapitel beschriebenen Hybrid-Verfahren, bei dem ein Public-Key Kryptosystem, namentlich RSA, zum Austausch des (symmetrischen) Schlüssels verwendet wird.
Outband weist darauf hin, dass der Schlüssel auf eine wie auch immer geartete Weise bereits (extern) ausgetauscht wurde.
Inband: der Schlüssel wird in Form einer Hexadezimaldarstellung – wohlgemerkt innerhalb einer verschlüsselten Nachricht – direkt einem symbolischen Bezeichner zugewiesen.
Kerberos: analog zu *inband*, nur dass das gemeinsame Geheimnis ein Kerberos[16] *ticket/ authenticator* Paar ist.

Auf die S-HTTP-Fähigkeit eines Web-Servers weist der für diesen Zweck eingeführte URL-Typ `s-http` hin, der nicht zu verwechseln ist mit der Kennzeichnung `https` für SSL-Verbindungen.

[16] Kerberos ist ein Authentifizierungsprotokoll mit einer vertrauenswürdigen dritten Partei, das für TCP/IP-Netze entwickelt wurde (vgl. Schneier, 1997, S. 642 ff).

Trotz einer umfangreicheren Funktionalität gegenüber SSL und einer damit einhergehenden größeren Flexibilität gibt es wegen der entsprechend aufwendigeren Implementierung nur einige wenige Produkte, die dieses Protokoll unterstützen (Platzer, 1996).

6.6 Übertragungssicherheit in MEntAs

In der MEntAs-Architektur kommen verschiedene Protokolle zum Einsatz. Für die Übertragung zwischen dem Java-Client und dem Java-/MEntAs-Server wird RMI eingesetzt. Der Server verbindet sich mit der Middleware über die Datenbankschnittstelle JDBC und die Verbindung zu den integrierten Datenbanken wird über herstellerspezifische Protokolle abgewickelt.

All diesen Protokollen ist nur gemeinsam, dass sie auf dem Transport- bzw. Netzwerkprotokoll TCP/IP aufbauen. Eine homogene Lösung zur Gewährleistung der Übertragungssicherheit kann daher nur auf dieser oder einer darunterliegenden Schicht stattfinden.

Idealerweise sollten Vertraulichkeit, Integrität und Authentizität der Nachricht von der Netzwerkschicht angeboten werden. Dies wird im Internet/Intranet aber erst mit dem *Internet Protocol next Generation*(IPnG) realisiert werden. Bis dahin muss also mit anderen Lösungen Vorlieb genommen werden.

6.6.1 Übertragung zwischen Middleware und Datenbanken

In dem vorliegenden Prototyp *mini*-MEntAs wird derzeit nur ein Abzug der später zu integrierenden Datenbanken verwendet. Diese Abzüge befinden sich in extra angelegten lokalen DBSen ebenfalls auf der Server-Maschine. Dadurch kommt bei der Übermittlung zwischen Middleware und den eigentlichen Datenbanken der *loopback*-Mechanismus (zur Erläuterung siehe Abschnitt 6.6.2) zum Tragen, und eine Absicherung der übertragenen Daten ist nicht notwendig.
Dies entspricht allerdings nicht der realen Umgebung einer produktiv eingesetzten Version von MEntAs. Wegen der Erhaltung der lokalen Autonomie der integrierten Datenbanken werden diese auf Systemen in den entsprechenden Abteilungen verbleiben. Ein Zugriff durch die Middleware würde auf jedenfall entfernt erfolgen, unter Umständen sogar über die Grenzen der Standort-Firewall hinaus. Eine Verschlüsselung wird notwendig.

Die Integration der Datenbanken in DataJoiner erfolgt über deren eigene Kommunikationsprotokolle (bisher SQL*Net und DB2RA). Durch die Unterschiedlichkeit der Protokolle gestaltet sich eine homogene Lösung für die Verschlüsselung auf der Teilstrecke zwischen Abteilungsdatenbanken und Middleware als schwierig. Einziger Ansatzpunkt ist das beiden Protokollen zugrundeliegende Transport-/Netzwerkprotokoll TCP/IP. Aber gerade darauf baut das auf Seite 32 vorgestellte Produkt cryptSSL auf. Mit ihm lassen sich Daten auf beliebigen TCP/IP-Verbindungen verschlüsseln, womit sich cryptSSL gut zur Sicherung dieses Teilabschnittes von MEntAs eignet.

Für den Betrieb von cryptSSL muss auf der Client- als auch der Server-Seite eine Software-Komponente installiert werden. Server wären in dem Fall die integrierten

Abteilungsdatenbanken, Client der MEntAs-Server. Derzeit werden von cryptSSL allerdings nur WindowsNT-Clients und UNIX-Server unterstützt, im vorliegendend Fall wird jedoch zumindest ein UNIX-Client benötigt. Eine Portierung auf andere Plattformen stellt nach Aussagen der Verantwortlichen bei dem Hersteller Giesecke & Devrient kein Problem dar, bedarf jedoch einer Beauftragung von seiten des DaimlerChrysler-Konzerns.

6.6.2 Übertragung zwischen Server und Datenbank-Server

Der MEntAs-Server und die Middleware laufen auf derselben Maschine[17]. Die TCP/IP-Verbindung von JDBC (siehe Darstellung 3.1 auf Seite 26) wird daher über die sogenannte *loopback-Adresse* abgewickelt. Bei dieser Adresse handelt es sich um eine spezielle IP-Adresse (127.0.0.1), dem sogenannten *localhost*, über den der lokale Rechner genau so adressiert werden kann wie ein fremder Rechner. Dadurch wird die Programmierung von Anwendungen dahingehend vereinfacht, dass sowohl für die Kommunikation mit lokalen als auch mit fremden Prozessen derselbe Code verwendet werden kann. Darüber hinaus reduziert die Adressierung des *localhost* die Belastung des Netzwerkes, weil Daten an den lokalen Rechner zurückgeschickt werden, bevor sie über das Netz wandern. Dies wird über ein sogenanntes pseudo-device realisiert, welches ein Gerätetreiber ist, der nicht direkt auf eine Hardware-Schnittstelle zugreift. (Hunt, 1995)

Da im vorliegenden Fall die Kommunikation nur intern auf dem Rechner stattfindet, ist eine Verschlüsselung der Daten auf dieser Teilstrecke nicht notwendig.

6.6.3 Übertragung zwischen Client und Server

Auf der Client-Seite wird nach Aufruf des MEntAs-Applets eine RMI-Verbindung zu dem Server aufgebaut, über die der anschließende Datenaustausch abgewickelt wird (weshalb auch nicht S-HTTP verwendet wird). Da RMI ebenfalls auf TCP/IP basiert, könnte auch hier die Kommunikation mittels der cryptSSL-Lösung aus der Sicherungsinfrastruktur gesichert werden. Zur Verwendung dieses Produkts müssen allerdings sowohl auf Client-, als auch auf Server-Seite Software-Komponenten installiert werden (siehe S. 32). Das bedeutet, dass auf jedem Rechner, von dem aus auf MEntAs zugegriffen werden soll, ein cryptSSL-Client installiert werden muss, wodurch die Realisierung der angestrebten plattformunabhängigen Schnittstelle über Java wieder zunichte gemacht wird.
Die noch fehlende generelle IP-Sicherung durch IPnG sowie der Tatbestand, dass außer dem Applet keine zusätzliche Software auf Client-Seite notwendig sein sollte,[18] machen eine selbst vorgenommene Implementierung der Verschlüsselungskomponenten fast unumgänglich. Dies erweist sich allerdings als recht unproblematisch, da es sich bei den Client- und Server-Komponenten von MEntAs um Eigenentwicklungen handelt, die wegen der Verfügbarkeit des Quellcodes bequem um Kryptoerweiterung zur Transportsicherung ergänzt werden können.

Für die Chiffrierung wird ein Hybrid-System aus einem Public-Key (beispielsweise das RSA-Verfahren) und einem Secret-Key Verfahren (zum Beispiel IDEA) vorgeschlagen.

[17] Eine IBM RS/6000 aus der F50 Serie mit dem UNIX-Derivat AIX 4.3.
[18] Dabei wird das Vorhandensein des Netscape-Browser stillschweigend vorausgesetzt.

- Für den Server gibt es ein Public-Key Schlüsselpaar bestehend aus dem öffentlichen Teil $\overline{k}_S$ und dem geheimen Teil k_S.

- Nachdem der Client sich bei dem Server erfolgreich authentifiziert hat (vgl. Abschnitt 5.4), erzeugt er einen zufälligen Sitzungsschlüssel k_r für ein symmetrisches Kryptosystem.

- Dieser Sitzungschlüssel wird mit dem öffentlichen Schlüssel des Servers $\overline{k}_S$, welchen der Client über das Applet erhalten hat, verschlüsselt und an den Server geschickt.

  Da das Applet signiert ist (siehe S. 33), kann der Client sicher sein, dass das Applet wirklich von dem Server stammt und damit auch der darin enthaltene öffentliche Schlüsselteil authentisch ist.

- Der Server (und nur er) kann die erhaltene Nachricht mit dem geheimen Schlüsselteil k_S entschlüsseln. Beiden Partnern liegt nun der Sitzungsschlüssel k_r vor, mit dem unter Verwendung eines symmetrischen Kryptoverfahrens die zu übertragenden Daten chiffriert werden.

Bei der eben skizzierten Vorgehensweise kann der Client sicher sein, dass sein erzeugter Sitzungsschlüssel nur von dem richtigen Server verwendet werden kann. Allerdings ist für den Server trotz der zuvor erfolgten Benutzerauthentifizierung ungewiss, ob der erhaltene Sitzungsschlüssel auch von dem rechtmäßigen Client stammt (ein Angreifer könnte sich erst nachträglich zwischengeschaltet haben). Daher ist es anzuraten, den Sitzungsschlüssel schon während der Authentifizierungsphase des Client an den Server zu übermitteln. Dazu wird das aktuelle Passwort p_i zusammen mit dem Sitzungschlüssel in eine Nachricht gepackt, die dann mit dem öffentlichen Schlüssel $\overline{k}_S$ des Servers verschlüsselt wird. Der Server hat nun (bei erfolgreicher Entschlüsselung) die Gewissheit, dass der Sitzungsschlüssel zu der authentifizierten Instanz gehört und dass diese Informationen während der Übertragung nicht verändert wurden.

Eine direkte Übermittlung des öffentlichen Server-Schlüssels zusammen mit dem Applet hat den Vorteil, dass im Falle einer Kompromittierung das Schlüsselpaar auf einfache Weise ausgetauscht werden kann, ohne dass der Benutzer davon etwas mitbekommt; beim nächsten Laden des Applets erhält er auch automatisch den neuen Server-Schlüssel.

Trotz der Einbindung von Chiffrierroutinen in MEntAs sollte die Möglichkeit gegeben sein, die Daten zwischen Client und Server unverschlüsselt zu übertragen. Zum Einen besteht dafür keine Notwendigkeit, falls Client und Server in einer gesicherten Netzwerkumgebung ablaufen (innerhalb einer Firewall), zum Anderen bringt die Verschlüsselung (da Mehraufwand) immer Verluste in der Performanz mit sich, was sich gerade zu Zeiten großer Netzlast als äusserst störend auswirkt.
Es sei aber auch erwähnt, dass die Möglichkeit einer Deaktivierung der Verschlüsselung Gefahren mit sich bringt. Manch einem ist die Geschwindigkeit oft wichtiger als die Sicherheit der Daten. Benutzer, die sich der Notwendigkeit gesicherter Übertragungswege nicht bewusst sind, tendieren in diesem Fall dazu, die Verschlüsselung generell abzuschalten. Sollte der Schutz dann einmal auf Grund sensitiver Daten wirklich relevant sein, wird häufig vergessen, die Verschlüsselung wieder einzuschalten.

6.7 Zusammenfassung

In diesem Kapitel wurden bezüglich der Authentifizierung Zertifikate und deren Verwaltung betrachtet; zum Thema Isolierung/Geheimhaltung wurden einige bekannte Verschlüsselungsalgorithmen erläutert sowie Betrachtungen zur Erzeugung von Schlüsseln durch Pseudozufallsfolgengeneratoren vorgenommen. Bevor abschließend auf die Übertragungssicherheit im konkreten Fall MEntAs eingegangen wurde, wurden noch drei für den Web-Bereich in Frage kommende Sicherungsprotokolle vorgestellt.

Kapitel 7

Auditing

In diesem Kapitel werden nach einigen generellen Betrachtungen zu Überwachungsmechanismen die Auditing-Möglichkeiten – entsprechend den Komponenten von MEntAs – durch das Betriebssystem (UNIX), die Datenbanksysteme (ORACLE und DB2) und den Web-Server (Apache) beschrieben.

7.1 Einleitung

Nach der Einführung von Schutzmechanismen möchte man auch sicher sein, dass diese ihre Aufgabe erwartungsgemäß erfüllen. Da es aber keine absolute Sicherheit gibt, sollte man – getreu dem Spruch „Vertrauen ist gut, Kontrolle ist besser“ – ihre Funktionsweise überwachen. Diese Überwachung, durch Aufzeichnung von ausgewählten Aktivitäten in einem Computersystem realisiert, bezeichnet man als *Auditing*.
Bevor man sich allerdings an das blinde Aufzeichnen aller im System stattfindenden Aktivitäten macht, sollte man sich Gedanken über einige generelle Anforderungen an einen Auditing-Mechanismus machen.[1]

Zuerst sollte man sich die Frage stellen, ob ein permanentes Auditing überhaupt notwendig ist, oder ob es nur dann aktiviert werden soll, wenn bestimmte Verdachtsmomente auftreten (ähnlich der momentan aktuellen Diskussion über das Mitschneiden von Telefongesprächen).

Der Auditing-Mechanismus selbst sollte so in das System eingebettet sein, dass er automatisch ablaufen kann. Dabei darf der normale Betrieb nur so wenig wie nötig beeinträchtigt werden. Ideal wäre, wenn er sich gar nicht auf die Leistungsfähigkeit des Systems auswirken würde. Dazu ist es unumgänglich, sich Klarheit darüber zu verschaffen, welche Aktionen aufgezeichnet werden sollen. Es macht wenig Sinn, einfach alles mitzuprotokollieren, da dadurch zum Einen die oben erwähnte Leistung des Systems mit Sicherheit in Mitleidenschaft gezogen, und zum Anderen die Auswertung der Daten, sprich die Unterscheidung von wichtigen und unwichtigen Aktionen erschwert würde. Über welche Vorgänge letztendlich Buch geführt werden soll, hängt von dem zugrundeliegenden System ab.

Die Speicherung der Auditing-Datensätze muss in einer gesicherten Umgebung erfolgen. Dadurch soll es einem Angreifer erschwert werden, seine bei dem Einbruch hinterlassenen

[1] Teilweise in Anlehnung an Amoroso (1994).

Spuren zu verwischen. Redundante Aufzeichnungen in mehreren Dateien erschweren die Vertuschung zusätzlich. Für die Protokollierung von Vorgängen ist es nicht unbedingt erforderlich, eine naheliegende elektronische Speicherung vorzunehmen. Denkbar wäre auch das Ausdrucken der Datensätze auf einem von der Öffentlichkeit abgeschirmten Drucker. Es müsste allerdings seine ständige Bereitschaft gewährleistet werden. Ein Papierstau oder das Ausgehen der Druckfarbe würden den Auditing-Vorgang behindern.

Die gespeicherten Datensätze sollten (ähnlich den Bedrohungen aus Kapitel 2) Aufschluss darüber geben, wer wann auf welche Daten und mit welchem Typ von Operation zugegriffen hat. Dabei ist es durchaus sinnvoll, nicht nur die erfolgreich durchgeführten Aktionen festzuhalten, sondern auch misslungene Versuche, an Hand derer zum Beispiel festgestellt werden kann, ob ein Benutzer massive Versuche unternimmt, sich durch Passwort-Raten auf einem System als `root` anzumelden.

Das Aufzeichnen von Systemaktivitäten macht wenig Sinn, wenn die Daten nicht von Zeit zu Zeit auf Unregelmäßigkeiten hin untersucht werden. Garfinkel und Spafford empfehlen mindestens eine tägliche Revision der Daten. Eine so häufig wiederkehrende Aufgabe, kann leicht zur Routine werden, besonders wenn man bedenkt, dass im Normalfall eine große Menge unbedeutender Daten durchgesehen werden muss. Dabei können leicht wichtige Einträge übersehen werden. Es ist daher ratsam, einen Filter auf die Dateien anzuwenden, der die unwichtigen Daten unberücksichtigt lässt.
In jedem Fall benötigt die Durchsicht Zeit und Personal. Ein Umstand, der gerne dazu führt, die Zeitintervalle für die Überprüfung der Protokolldateien recht großzügig zu gestalten (wenn überhaupt eine Durchsicht erfolgt). Dabei darf man aber nicht vergessen, dass wenn die Abstände zu groß gewählt werden, unter Umständen ein automatisches Speichermanagement der Protokolle bereits wichtige Einträge gelöscht hat, oder aber der Angreifer die Möglichkeit erhalten hatte, die Protokolldateien wieder „zu richten“.

Letztendlich sollte man sich noch Gedanken über die Verwaltung von alten Auditing-Datensätzen machen. Da diese Datensätze bekanntlich Platz verbrauchen, wird ohne eine vorhandene Strategie das System unweigerlich um wertvolle Speicher-Ressourcen beschnitten. Knackpunkt scheint hier der Begriff „alt“ zu sein, der darüber entscheidet, ob bestimmte Datensätze gelöscht werden können oder beibehalten bzw. archiviert werden müssen. Eine automatische Lösung dieses Problems kann den Nachteil eines falsch dimensionierten Schwellenwertes mit sich bringen (beispielsweise kann bei einem Zeitintervall gesteuerten Ansatz an verkehrsreichen Tagen der Speicherplatz knapp werden). Bei einer manuellen Verwaltung wird hingegen ein Administrator benötigt, der sich regelmäßig um die Protokolldateien kümmern muss. Unabhängig davon stellt sich die generelle Frage, wie festgelegt wird, ob bestimmte Einträge noch benötigt werden und welche nicht mehr aktuell sind.[2]

Auditing bietet neben seinem eigentlichen Zweck – der Rekonstruktion von Details eines Angriffes – den zusätzlichen Aspekt der Abschreckung. Potenzielle Angreifer werden unter Umständen von ihrem Vorhaben abgehalten, weil sie wissen, dass ihre Schritte nachverfolgt werden können. Sie müssen in dieser Situation genau abwägen, ob der durch den Angriff erlangte Nutzen den Nachteil einer durch den Auditing-Mechanismus

[2] Unter Linux (S.U.S.E. Distribution) existiert zum Beispiel unter `/etc` eine Datei namens `logfiles`, in der alle Log-Dateien des Systems zusammen mit ihrer maximalen Größe aufgeführt sind. Es wird von dem Shell-Skript `cron.daily` benützt, welches – wie der Name schon sagt – täglich gestartet werden sollte und neben vielen anderen administrativen Aufgaben auch die Log-Dateien entsprechend kürzt.

wahrscheinlichen Entdeckung aufwiegt. Darüber hinaus stellt das Auditing einen weiteren Sicherheitsmechanismus dar, der, falls der Angreifer nicht entdeckt werden will, überwunden werden muss und somit einen Mehraufwand beim Einbruch in das System bedeutet.

Wenden wir uns nun den einzelnen Teilsystemen zu, die Auditing unterstützen.

7.2 Betriebssystem

Es sei vorweggenommen, dass sich keine pauschalen Aussagen über den Einsatz von Auditing bei Betriebssystemen machen lassen. Vielmehr hängt es von dem einzelnen System ab, welche Möglichkeiten vorhanden sind. Man kann allerdings einige Punkte feststellen, die für eine Protokollierung von Aktivitäten auf Basis des Betriebssystems erfüllt werden sollten. Zum Einen sollte Buch über die An- und Abmeldeprozesse geführt werden, an Hand derer nachvollzogen werden kann, wer wie (lokal oder per Fernzugang) und zu welchem Zeitpunkt Zugang zu den Systemressourcen hatte. Zum Anderen sollte festgehalten werden, auf welche Ressourcen zugegriffen wurde. Dabei spielt die CPU-Nutzung eine große Rolle, zu deren Überwachung die Namen der auf ihr ausgeführten Programme protokolliert werden.
Die aufgezeichneten Daten werden oft von den Systemprogrammen in Textdateien abgespeichert (entweder in einer lesbaren Form oder als typisierte Datensätze). Da diese Protokolldateien großteils auch in dem System gespeichert werden, welches es zu überwachen gilt, sind sie häufig selbst Ziel eines Angriffes. Eine Lösung dieses Problems wäre, die log-Einträge auf einem anderen, gesichert aufgestellten Rechner zu speichern (beispielsweise einem PC, auf dem man sich nicht von außerhalb anmelden kann und der durch eine direkte Leitung mit dem Hostsystem verbunden ist; vgl. Garfinkel und Spafford, 1996).

Im Folgenden werden nun die verschiedenen Protokolle des UNIX-Betriebssystems AIX Version 4.3 (IBM) – der Plattform für den MEntAs-Server – beschrieben.

7.2.1 Anmeldungsprotokolle

In UNIX gibt es fünf Dateien, die alle – mehr oder weniger redundant – Informationen über die An- und Abmeldung von Benutzern enthalten. Wichtige Daten dabei sind der Benutzername und der Zeitpunkt der Anmeldung, sowie ggf. der Name des Hostrechners, von dem aus die Anmeldung erfolgte.

Protokolldatei wtmp

`wtmp` ist eine typisierte (Binär-) Datei, die mit dem Programm `last` ausgelesen werden kann. Sie zeigt einen zeitlichen Abriss aller An- und Abmeldezeiten von Benutzern bei dem System.

```
root      pts/3        hobbes       27 Jan 09:49   still logged in.
ruetschl  ftp          frevo        07 Jan 11:07 - 11:07  (00:00)
mentas    lft0                      04 Jan 09:40 - 16:44 (11+07:04)
```

Die drei Zeilen sind eine exemplarische Ausgabe von `last`. Die Ausgabe erfolgt chronologisch absteigend. Nach der Spalte mit dem Benutzernamen folgt ein Eintrag für das verwendete Terminal und dem Rechner, von dem der Zugriff aus erfolgte.

Die Zugriffsrechte der beiden Dateien sehen wie folgt aus:

```
-r-xr-xr-x   1 bin      bin         8244 30 Sep 1997  /usr/bin/last*
-rw-rw-r--   1 adm      adm       605824 22 Dez 09:47 /var/adm/wtmp
```

`last` kann also von jedem Benutzer ausgeführt werden, um sich einen Überblick über die letzten Anmeldungen bei dem System ausgeben zu lassen. Wichtig ist, dass nur Mitglieder aus der `adm`-Gruppe Änderungen an der Log-Datei vornehmen können.

Protokolldatei utmp

Im Gegensatz zu der vorhergehenden Protokolldatei stellt diese keine Historie der Ab- und Anmeldeprozesse dar, sondern gibt Aufschluss darüber, wer momentan bei dem System angemeldet ist. Neben den Informationen wie Datum, Zeit, Terminal und Hostname des Benutzers werden auch Prozessdaten festgehalten (Prozess-ID und Typ). Im Folgenden ist eine Auswahl von Einträgen zu sehen.

```
User      ID       Line         PID     Type          Term.  Exit  Hostname     Time
-------------------------------------------------------------------------------------------
                   system boot  0       boot_time     0      0                  Thu Nov 12 21:44:36 1998
                   run-level 2  0       run_lvl       50     83                 Thu Nov 12 21:44:36 1998
cron      cron                  11880   init_process  0      0                  Thu Nov 12 21:45:26 1998
qdaemon   qdaemon               12132   init_process  0      0                  Thu Nov 12 21:45:26 1998
orapw     orapw                 14706   init_process  0      0                  Thu Nov 12 21:45:36 1998
root      pts/7     pts/7       26262   user_process  0      0     hobbes       Tue Dec 22 09:40:29 1998
mentas    pts/8     pts/8       22496   user_process  0      0     calvin       Tue Dec 22 09:47:40 1998
mentas    pts/9     pts/9       49150   user_process  0      2     :0.0         Fri Dec 11 12:25:37 1998
hermsen   /8        pts/8       33956   dead_process  12274  2     gundel       Fri Dec 18 17:49:46 1998
          pts/2     pts/2       35352   dead_process  208    15                 Wed Dec  2 13:19:23 1998
mentas    /4        pts/4       16574   dead_process  12274  2     frevo        Thu Nov 26 10:50:27 1998
hermsen   12        pts/12      52294   dead_process  12274  2     hobbes       Thu Dec 17 18:44:53 1998
pereira   14        pts/14      21864   dead_process  12274  2     53.16.10.54  Thu Dec 17 15:50:01 1998
ruetschl  pts/14    pts/14      21974   user_process  0      0     hobbes       Tue Dec 22 11:59:28 1998
          pts/15    pts/15      54920   dead_process  64     19                 Tue Dec 22 12:04:55 1998
```

Prinzipiell kann jeder Benutzer diese Log-Datei auslesen. Da es sich allerdings um eine typisierte Binärdatei handelt, muss er sich dazu ein entsprechendes Programm schreiben (das Datenformat ist in der Header-Datei `/usr/include/utmp.h` definiert).

```
-rw-r--r--   1 root     system      3456 27 Jan 11:32 /etc/utmp
```

Protokolldatei sulog

Diese Protokolldatei führt Buch über alle Aufrufe des Kommandos `su`, mit dem sich die Benutzeridentifikation ändern lässt.

```
SU 01/20 09:41 - pts/2 hermsen-mentas
SU 01/20 09:41 + pts/2 hermsen-mentas
SU 04/06 16:31 + pts/0 root-mentas
SU 04/08 17:39 + pts/3 root-ruetschl
SU 04/09 14:19 - pts/2 mentas-ruetschl
```

Neben Datum, Uhrzeit und verwendetem Terminal, zeigt ein Eintrag ausserdem an, von welcher Kennung auf welche gewechselt wurde und ob der Vorgang erfolgreich verlaufen ist (+) oder nicht (-).
Aus den obigen Zeilen, lässt sich zum Beispiel ablesen, dass der Benutzer `hermsen` sich bei seinem ersten Benutzerwechsel offenbar bei der Eingabe des Passwortes vertippt hat, da er sich bei dem noch in der gleichen Minute stattfindenden zweiten Versuch erfolgreich anmelden konnte. Eine sehr lange Serie gleichlautender, negativer Benutzerwechsel kann auf einen massiven Angriff gegen das Passwort eines Benutzers hindeuten.
Rechte an der Datei hat nur der Superuser `root`.

```
-rw-------   1 root     system     58447 27 Jan 12:12 /var/adm/sulog
```

Protokolldatei lastlog

Diese Datei enthält für jeden Benutzer einen Eintrag darüber, wann dessen letzte (nicht) erfolgreiche Anmeldung stattgefunden hat. Neben Terminal (tty) und Hostrechner wird auch die Anzahl erfolgloser Versuche festgehalten. Bei den angegebenen Zeiten handelt es sich um die Anzahl von Sekunden, die seit der „UNIX-Geburtsstunde“ (1. Januar 1970, 00:00:00 GMT) vergangen sind.

```
ruetschl:
        time_last_login = 913715033
        time_last_unsuccessful_login = 908963755
        tty_last_login = /dev/pts/6
        tty_last_unsuccessful_login = /dev/pts/11
        host_last_login = hobbes
        host_last_unsuccessful_login = gustav
        unsuccessful_login_count = 0
```

Zugriff auf die Datei hat nur der Superuser und Mitglieder aus der `security` Gruppe.

```
-rw-r-----   1 root     security    7730 28 Jan 10:46 /etc/security/lastlog
-r-sr-xr-x   3 root     security   58474 03 Okt 1997  /usr/sbin/login*
```

Die in dieser Protokolldatei gepflegten Informationen werden beispielsweise bei der Anmeldung eines Benutzers ausgegeben. Um dies zu ermöglichen, wurde das Kommando `login` mit dem s-Flag versehen, welches das Betriebssystem veranlässt, den Befehl mit den Rechten des Eigentümers und nicht wie normalerweise mit denen des aufrufenden Benutzers auszuführen.

Protokolldatei failedlogin

Eine chronologische Auflistung aller fehlgeschlagenen Anmeldeversuche wird in der Datei `failedlogin` geführt.

```
User      Line        PID    Type          Term.  Exit  Hostname  Time
-------------------------------------------------------------------------------
UNKNOWN_  dtlogin/_0  5178   user_process  0      0               Tue Jan 26 09:52:25 1999
ruetschl  pts/12      28278  user_process  0      0     maracatu  Thu Jan 28 11:55:16 1999
mentas    pts/12      28498  user_process  0      0     calvin    Thu Jan 28 15:30:47 1999
hermsen   pts/11      35374  user_process  0      0     gundel    Fri Jan 29 10:15:39 1999
ruetschl  dtlogin/_0  41710  user_process  0      0               Mon Feb  1 09:12:45 1999
root      lft0        13418  user_process  0      0               Mon Feb  1 09:35:39 1999
```

Die Informationen werden in demselben Format gespeichert wie bei `utmp` (siehe S. 92) und benötigen daher auch ein eigenes Programm, um die Daten auszulesen. Der einzige Unterschied zu `utmp` liegt darin, dass bei nicht erkanntem Benutzernamen `UNKNOWN_USER` eingetragen wird. Dies soll verhindern, dass ein fälschlicherweise als Benutzername eingegebenes Passwort unverschlüsselt im System erscheint.

```
-rw-r--r--  1 root   system    26176 28 Jan 10:46 /etc/security/failedlogin
```

7.2.2 Prozessprotokolle

Neben der Überwachung von An- und Abmeldeprozessen kann in UNIX jedes Kommando jedes Benutzers protokolliert werden; in der englischsprachigen Literatur bezeichnet man diese Überwachung oft als *process accounting* (im Folgenden mit PA abgekürzt).

Protokolldatei pacct

Als einfache Schnittstelle zum Starten des PA dient das Skript `startup` (`shutacct` schaltet die Protokollierung wieder ab). Es steht neben den anderen PA-spezifischen Befehlen in dem Verzeichnis `/usr/sbin/acct`.
Zur Protokollierung schreibt der Betriebssystemkern nach Beendigung jedes Prozesses einen entsprechenden Datensatz in die Datei `pacct`.

```
-rw-rw-r--   1 adm      adm         5800 22 Dez 13:14 /var/adm/pacct
-r-xr-xr-x   1 bin      bin         7810 30 Sep 1997  /usr/bin/lastcomm*
-r-xr-xr-x   1 root     adm        23184 27 Sep 1997  /usr/sbin/acct/acctcom*
```

Gelesen können diese Einträge mit den beiden Kommandos `acctcom` und `lastcomm`. Während `lastcomm` nur sehr wenige Optionen zulässt – es können Kommandos nach Name, Benutzer und Terminal gefiltert ausgegeben werden – bietet `acctcom` mit seinen knapp zwei Dutzend Parametern ein reichhaltiges Angebot an Auswertungsmöglichkeiten für die Protokolldaten. Im nächsten Absatz sind einige ausgewählte Zeilen der `lastcomm`-Ausgabe zu sehen.

```
sh             hermsen  pts/11         0,05 secs Do Jan 28 10:34
java           hermsen  pts/11        21,52 secs Do Jan 28 10:34
cp       S     root     pts/4          0,01 secs Do Jan 28 10:41
sh       SF    root     __             0,01 secs Do Jan 28 10:36
less     S     root     pts/4          0,01 secs Do Jan 28 10:35
ls             ruetschl pts/14         0,01 secs Di Dez 22 12:30
```

Nach dem Befehlsnamen, einer Statusspalte, dem Benutzer und Terminal wird die für die Ausführung benötigt CPU-Zeit ausgegeben, beendet durch den Zeitpunkt des Befehlsstarts.
In der Statuszeile können vier verschiedene Flags auftreten: Befehle mit einem `S` wurden durch den Superuser ausgeführt; `F` bezeichnet Prozesse nach einem *fork*; mit `D` markierte erzeugten einen *core dump*, und mit `X` gekennzeichte Befehle wurden durch ein Signal beendet.

Die angegebenen CPU-Zeiten werden auch häufig zu Abrechnungszwecken in Rechenzentren verwendet (daher der Begriff *process accouting*). Eine etwas differenzierte Aufstellung für diesen Zweck liefert der Befehl `acctcom`, der zusätzlich noch die zur Ausführung benötigte Realzeit und den Hauptspeicherplatz anzeigt.

```
ACCOUNTING RECORDS FROM:  Di 22 Dez 12:28:57 1998
COMMAND                        START    END          REAL       CPU     MEAN
NAME       USER     TTYNAME    TIME     TIME       (SECS)    (SECS)  SIZE(K)
#bsh       root     pts/1    12:28:57 12:28:57      0,14      0,02   113,00
#rm        root     pts/1    12:28:57 12:28:57      0,00      0,00   348,00
#bsh       root     pts/1    12:28:57 12:28:57      0,28      0,02     0,00
#ls        root     pts/1    12:29:00 12:29:00      0,00      0,00   112,00
stty       ruetschl pts/14   12:30:12 12:30:12      0,02      0,00     0,00
ls         ruetschl pts/14   12:30:14 12:30:14      0,00      0,00     0,00
#lastcomm  root     pts/1    12:30:24 12:30:24      0,03      0,03    54,00
dtexec     mentas   ?        12:27:35 12:30:35    180,12      0,03    78,0
```

Während auf die Befehle zum Starten und Stoppen der Protokollierung nur der Superuser und Mitglieder der `adm`-Gruppe Zugriff haben, kann `pacct` mit Hilfe der Programme `acctcom` oder `lastcomm` von jedem eingesehen werden. Dadurch wird es einem Angreifer prinzipiell ermöglicht, festzustellen, ob er bei seinen Aktivitäten überwacht wird, und kann versuchen, entsprechende Gegenmaßnahmen zu treffen.

syslog-Mechanismus

Im Gegensatz zu der im vorherigen Abschnitt beschriebenen Protokollierung bei der der Betriebssystemkern die Erfassung und Speicherung der Audit-Einträge regelt, wird bei *syslog* ein eigener Protokollprozess gestartet (`/etc/syslogd`). Wenn ein Programm bestimmte Informationen protokolliert haben möchte, schickt es sie an diesen Prozess. Dabei kann über die Konfigurationsdatei `/etc/syslog.conf` festgelegt werden, welche Arten von Meldungen in welchen Dateien abgelegt werden sollen. Ein Eintrag hat die Form

$$\langle Bereich \rangle.\langle Priorität \rangle \qquad \langle Datei \rangle$$

Mit dem Bereich[3] können bestimmte Systemteile bezeichnet werden. Beispiele sind *kern* für den Kern, *user* für reguläre Benutzerprozesse oder *mail* für das Mail-System. Die Priorität kann Werte wie *alert*, *warning*, *info* oder *notice* annehmen, wodurch die Meldungen ihrer Dringlichkeit nach eingestuft werden können. Schließlich muss noch über die Dateibezeichnung (mit Pfad) angegeben werden, wohin die Informationen gespeichert werden sollen.
Es besteht weiter die Möglichkeit, einzelne Bestandteile eines Eintrages mit einem Jokerzeichen zu maskieren: `auth.*` erfasst alle Mitteilungen der Kategorie Authorisierung, während mit `*.alert` alle Meldungen mit Alarmpriorität in der angegebenen Datei festgehalten werden. Eine Maskierung der Zieldatei, sendet die Nachricht an alle Benutzer.
Ein Beispiel für die Datei `syslog.conf` könnte wie folgt aussehen.

[3] In der englischen Dokumentation zu *syslog* wird dieser Teil des Eintrags mit *facility* bezeichnet.

```
mail.debug              /usr/spool/mqueue/syslog
lpr.*                   /var/adm/lpr-errs
*.debug                 /dev/console
*.crit                          *
```

Programme, die standardmäßig Meldungen an *syslog* geben, sind beispielsweise `halt`, fehlgeschlagene `login`-Versuche, `su`, `reboot` oder auch `shutdown`.
Aus eigenen Anwendungen heraus können syslog-Einträge entweder über den Befehl `logger` oder unter Verwendung der C-Routine *syslog* (aus der libc-Bibliothek) erfolgen.

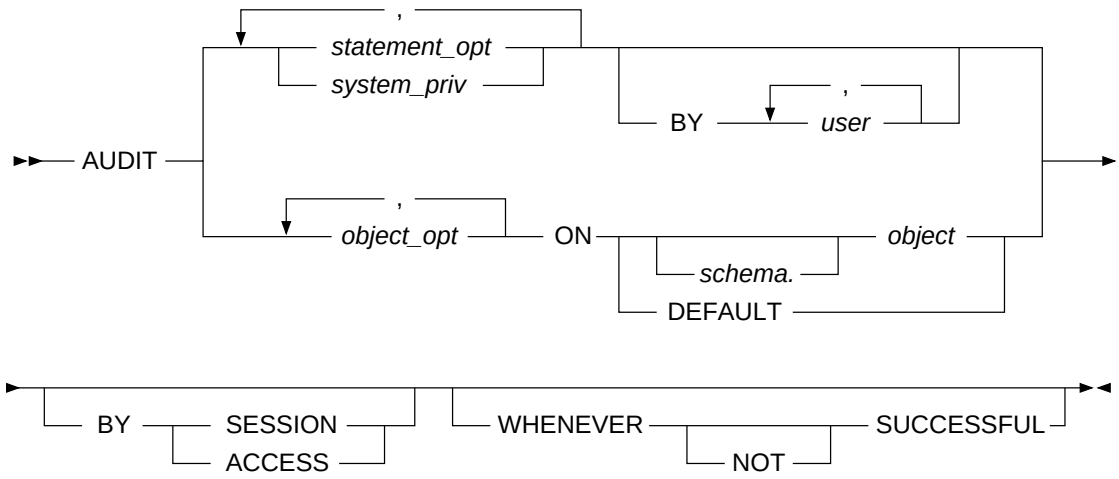
Weitergehende Informationen zu dem syslog-Mechanismus finden sich in Garfinkel und Spafford (1996, S. 310) oder den Manual-Seiten zu *syslog*.

7.3 Datenbanksystem

Wie auch bei den Betriebssystemen hängen die Auditing-Möglichkeiten in Datenbanken maßgeblich von dem verwendeten DBMS ab. Während ORACLE7 explizit einen dafür vorgesehenen Mechanismus anbietet, kann bei dem Konkurrenzprodukt DB2 von IBM nur über sogenannte *Trigger* (siehe Abschnitt 7.3.2) eine Mitprotokollierung von Datenbankaktivitäten erreicht werden.

7.3.1 Auditing bei ORACLE

Zur Definition von Auditing-Objekten gibt es bei ORACLE den SQL-Befehl `audit`, dessen Syntaxdiagramm wie folgt aussieht



Der Fokus richtet sich dabei entweder auf die Benutzer (`BY` *user*) oder auf Datenbankobjekte (`ON` *object*). Beim Benutzer-Auditing handelt es sich hauptsächlich um die Überwachung von administrativen Funktionen wie das Anlegen, Löschen und Ändern von Tabellen, Sichten, Rollen, Speicherbereichen o. ä. (*statement_opt* und *system_priv*). Dagegen wird bei Datenbankobjekten (Tabellen, Sichten oder Sequenzen) Buch über deren Inhalt betreffende Auslese-, Einfüge-, Änderungs- oder Löschoperationen (*object_opt*) geführt. Schließlich kann mit der `WHENEVER`-Klausel festgelegt werden, ob ein

Protokollierungseintrag entweder nur für erfolgreich durchgeführte oder nur für misslungene Befehle erzeugt werden soll. (Weitere Einzelheiten zu dem `audit`-Befehl können in ORA-SQL, 1992, S. 4-100 ff nachgelesen werden.)
Die Auditing-Datensätze werden in der Systemtabelle `sys.aud$` abgelegt und bilden die sogenannte *Auditfolge* (ORA-ADM, 1993). Auf dieser Tabelle sind einige Sichten definiert, um die aufgezeichneten Informationen strukturiert auszulesen. Ist die Auditfolge voll, so werden Aktionen, die zu einem Eintrag in dieser führen würden, zurückgewiesen; der Sicherheitsadministrator muss erst wieder Platz schaffen (durch Löschen oder Umkopieren der Auditfolge), bevor weitere Aktionen durchgeführt werden können. Die Systemtabelle `sys.aud$` unterliegt wie alle anderen Tabellen auch der Datenbankzugriffssicherung durch den `grant`-Befehl. Ihrem Namen entsprechend gehört sie in das Schema `sys` und kann somit nur von dem Datenbankadministrator (`internal`) manipuliert werden, sofern dieser keine Rechte an der Tabelle weitergegeben hat.

Falls die durch den Befehl `audit` angebotene Funktionalität den Bedürfnissen des Sicherheitsbeauftragten nicht ausreicht, besteht zusätzlich die Möglichkeit, auf das im nächsten Abschnitt beschriebene Trigger-Konzept zurückzugreifen

7.3.2 Auditing mit Triggern

Trigger sind in der Datenbank gespeicherte Prozeduren, die in dem Moment ausgeführt bzw. ausgelöst werden, in dem auf eine Tabelle zugegriffen wird (ORACLE, 1993). Sowohl die auslösende Zugriffsart als auch die ausgeführte Aktion wird bei der Definition eines Triggers festgelegt. Sie bieten damit ein breit gefächertes Anwendungsspektrum und können aufgrund ihrer Struktur unter anderem für das automatische Erzeugen von abgeleiteten Spaltenwerten, die Einhaltung referenzieller Integrität oder aber auch die Einrichtung eines ausgefeilten Auditings eingesetzt werden.
Chamberlin (1998, S. 426 f) beschreibt ein Beispiel, in dem mit Hilfe eben dieses Mechanismus eine Protokollierung aller Einfüge-, Änderungs- und Löschoperationen auf der Benutzertabelle `accounts` realisiert ist. Für die Speicherung der Audit-Datensätze wird folgende Tabelle angelegt:

```
CREATE TABLE account_changes
(
  type      CHAR(1),
  when      TIMESTAMP,
  bywhom    CHAR(8),
  nrows     INTEGER
);
```

Sie besteht aus einem Feld für die Beschreibung der Operation (`type`), einem Zeitstempel (`when`) für den Zeitpunkt der Aktion, der Benutzerkennung (`bywhom`) und der Anzahl durch den Befehl „angefassten“ Tupel (`nrows`). Mit folgenden drei Befehlen werden Trigger für Einfügungen (`account_trig1`), Änderungen (`account_trig2`) und Löschungen (`account_trig3`) auf der `account`-Tabelle definiert.

```
CREATE TRIGGER account_trig1 AFTER INSERT ON accounts
  REFERENCING NEW_TABLE AS newtable
```

```
  FOR EACH STATEMENT MODE DB2SQL
  INSERT INTO account_changes (type, when, bywhom, nrows) VALUES
    ('I', CURRENT TIMESTAMP, USER, (SELECT COUNT(*) FROM newtable) );

CREATE TRIGGER account_trig2 AFTER UPDATE ON accounts
  REFERENCING NEW_TABLE AS newtable
  FOR EACH STATEMENT MODE DB2SQL
  INSERT INTO account_changes (type, when, bywhom, nrows) VALUES
    ('U', CURRENT TIMESTAMP, USER, (SELECT COUNT(*) FROM newtable) );

CREATE TRIGGER account_trig3 AFTER DELETE ON accounts
  REFERENCING OLD_TABLE AS oldtable
  FOR EACH STATEMENT MODE DB2SQL
  INSERT INTO account_changes (type, when, bywhom, nrows) VALUES
    ('D', CURRENT TIMESTAMP, USER, (SELECT COUNT(*) FROM oldtable) );
```

Das Schlüsselwort `after` sorgt dafür, dass der Trigger erst *nach* erfolgter Operation auf der Tabelle `accounts` angestoßen wird. `new_table` und `old_table` stellen gewissermaßen das *before-* und *after-image* dar. Sie enthalten alle von der Aktion betroffenen Tupel vor bzw. nach deren Ausführung. Bei `account_trig3` sind der Natur der Löschoperation entsprechend nach Ausführung des Befehls die gelöschten Tupel nicht mehr vorhanden, so dass hier auf das *before-image* `old_table` zurückgegriffen werden muss, um die Anzahl der von der Aktion betroffenen Tupel zu berechnen (`select count(*)`).

7.4 Web-Aktivitäten

Bei der Überwachung von Bewegungen innerhalb des World Wide Web muss man die Client- und die Server-Seite unterscheiden. Während der Betreiber eines Web-Servers primär daran interessiert ist, wer mit welchen Operationen auf seine Daten zugreift, steht auf der Client-Seite die Information über die angeforderten Daten/Seiten im Vordergrund.

Neben der Möglichkeit diese Informationen bidirektional durch den Internet-*Proxy* (engl. Stellvertreter) aufzuzeichnen (wobei allerdings nur der Betreiber des Proxys Zugang zu den Aufzeichnungsdaten hätte), führen die gängigen Browsern und Web-Server selbst Log-Dateien über die angeforderten Informationen.
Beispielsweise werden die URLs der über den Netscape-Navigator aufgerufenen Seiten in der datei `~/.netscape/history.db` abgelegt (in `history.list` stehen die von Hand eingegebenen Adressen). Diese Datei sowie das Unterverzeichnis selbst, sind nur von derem Besitzer zugreifbar; ein Systemadminstrator (`root`) kann allerdings wegen seiner Sonderstellung trotzdem darauf zugreifen. Die Inhalte der letzten angeforderten Seiten können direkt im `cache`-Verzeichnis eingesehen werden, sofern sie nicht von Hand oder aus der Anwendung heraus gelöscht wurden. Genau da liegt auch der Knackpunkt, warum man die durch den Browser angebotenen Log-Dateien streng genommen nicht als Auditing-Mechanismus ansehen kann: der Benutzer hat die Möglichkeit die aufgezeichneten Daten zu löschen oder zu verändern.

Anders sieht es auf der Server-Seite aus. Beispielsweise werden bei dem Web-Server Apache die Log-Dateien mit den Rechten versehen, die der installierende bzw. den

Server startende Benutzer hat. Gepflegt werden die Dateien `access_log`, `error_log`, `agent_log` und `referer_log` (vgl. Garfinkel und Spafford, 1996).[4]

Protokolldatei access_log

Diese Datei enthält eine Liste mit den Server-Zugriffen.

```
53.16.10.26 - - [27/Jul/1998:10:57:06 +0100] "GET
  /manual/index.html HTTP/1.0" 200 2537
53.16.10.26 - - [27/Jul/1998:10:57:06 +0100] "GET
  /manual/images/sub.gif HTTP/1.0" 200 6083
53.16.10.26 - - [27/Jul/1998:10:57:06 +0100] "GET
  /manual/images/index.gif HTTP/1.0" 200 1540
```

Die erste Spalte ist die Adresse des Rechners, von dem der Zugriff aus erfolgte. Dann folgen der *remote login* und der *remote username*, sofern sie übertragen wurden (ansonsten wie im Beispiel ein `-`). Nach dem Zeitstempel folgt der HTTP-Befehl, mit dem auf den Server zugegriffen wurde; das Beispiel zeigt die Anforderung (`GET`) eines HTML-Dokument und zweier Bilder. Falls vorhanden werden auch Parameter mit aufgeführt. Abschließend folgt der Rückgabewert und die Anzahl der übertragenen Bytes.

Selbst wenn der Benutzername nicht übertragen wird, ist es häufig möglich, Individuen zu identifizieren. Oft sitzt immer der gleiche Benutzer an einer Workstation, oder die Firmen vergeben jedem Benutzer zur Einbindung seines PCs in das Firmennetz eine eigene IP-Adresse. Gegenbenenfalls können weitere Nachforschungen über beispielsweise die Anmeldungs-Protokolle der angeführten Maschine und dem Zeitstempel zu dem Benutzernamen führen (dies ist allerdings nur möglich, wenn der Superuser sowohl Zugang zum Server, als auch zur Client-Maschine – dem Rechner, von dem aus der Web-Browser seine Anfrage machte – hat).

Protokolldatei error_log

Hier werden alle Meldungen des Servers protokolliert. Zwischen dem Zeitstempel und der eigentlichen Meldung befindet sich ein Feld für einen Statusbezeichner.

```
[Thu Jan 28 15:49:45 1999] [notice]
    Apache/1.3b3 configured -- resuming normal operations
[Thu Jan 28 16:03:48 1999] [notice]
    httpd: caught SIGTERM, shutting down
```

Protokolldatei agent_log

Hier wird eine Liste von Programmen gepflegt, über die auf den Web-Server zugegriffen wurde. Neben dem Namen des verwendeten Browsers werden auch Informationen über die Struktur der Firewall einer Firma oder der verwendeten Beta-Software aufgeführt.

[4] Damit die beiden letztgenannten Protokolldateien geführt werden, müssen die entsprechenden Flags bei der Übersetzung von Apache gesetzt werden.

Protokolldatei referer_log

Eine sehr aufschlussreiche Datei für das Benutzerverhalten ist `referer_log`. In ihr wird festgehalten, von welchem Server und Dokument auf eine Ressource dieses Servers zugegriffen wurde. Dadurch lässt sich verststellen, von welchen Seiten aus man auf ein bestimmtes Dokument dieses Servers gelangt ist.[5]

```
http://www2.infoseek.com/NS/Titles?qt=unix-hater -> /uinx-haters.html
http://www.intersex.com/main/ezienes.html -> /awa/
http://www.jaxnet.com/~jdcarr/places.html -> /vineyard/ferry.tiny.gif
```

7.5 Auditing in MEntAs

Das Betriebssystem sowie der Web-Server bieten für MEntAs ausreichende Auditing-Möglichkeiten an.
Bei den von der Datenbankseite her angebotenen Funkitionalitäten muss auf den recht mächtigen `audit`-Befehl (ORACLE7) verzichtet werden, da die integrierende Middleware des MEntAs-Datenbank-Servers eine DB2-Datenbank ist.
Ausserdem ergibt sich das Problem, dass Trigger nur auf Basistabellen definiert werden können. Die in DataJoiner zur Integration eingerichteten *nicknames* haben aber den Status einer Sicht. Die Trigger müssten also direkt in den Abteilungsdatenbanken angelegt werden, was aber wegen der geforderten Autonomie dieser Datenbanken (insbesondere nur lesender Zugriff) nicht realisierbar ist bzw. nicht im Aufgabenbereich von MEntAs liegt.
Einzige Möglichkeit eines Auditings über die integrierten Datenquellen besteht darin, direkt aus dem Anwendungsprogramm heraus (über den MEntAs-Server) die von den Benutzern abgeschickten SQL-Befehle in einer spezielle für diesen Zweck angelegten Relation zu protokollieren.
Diese Auditing-Daten können auch zu Optimierungszwecken herangezogen werden. Es lässt sich feststellen, welche Tabellen über welche Attribute besonders oft angefragt wurden. Auf diese können dann zusätzliche Indexe für einen beschleinigten Zugriff definiert werden.
Für die von den Ingenieuren mit MEntAs erstellten und in DataJoiner gespeicherten Motorkonzepte lässt sich Auditing über Trigger wie beschrieben einsetzen.

7.6 Zusammenfassung

Nach einigen grundsätzlichen Bemerkungen zum Thema Auditing wurde in diesem Kapitel ansatzweise die Mächtigkeit des Trigger-Konzepts von Datenbanken vorgestellt, mit dem nahzu beliebig gartete Protokollierungsansprüche realisiert werden können. Das DBS von Oracle bietet ergänzend dazu einen eigenen Auditing-Mechanismus an. Leider können beide Möglichkeiten für die integrierten Datenquellen in MEntAs nicht genutzt werden, so dass dort ein proprietäre Lösung verfolgt werden muss.
Unter dem Betriebssystem UNIX und dem Web-Server Apache läuft bereits von Haus aus eine ausreichende Protokollierung grundlegender Operationen im Hintergrund mit.

[5] Das folgende Beispiel wurde aus dem Buch von Garfinkel und Spafford (1996) entnommen.

Kapitel 8

Bewertung und Ausblick

Während den Untersuchungen zu dieser Arbeit hat sich herausgestellt, dass der Bedarf an Sicherheitskonzepten erkannt wurde, bisher aber größtenteils nur heterogene Teillösungen realisiert wurden. Für jede Komponente – sei es das Betriebssystem oder ein Datenbanksystem – wurde eine zweckdienliche Lösung implementiert. Möchte man jedoch ein alle Systeme überspannendes Sicherheitsnetz aufbauen, so ist man momentan gezwungen, diese Teillösungen mehr oder weniger in einer proprietären Art und Weise zu verknüpfen.

Die Realisierung eines ganzheitlichen und vor allem homogenen Sicherheitskonzept ist zur Zeit nicht möglich. Ein großer Schritt in die richtige Richtung wurde zum Einen mit dem *Internet Protocol next Generation* (IPnG) gemacht, welches auf einer der untersten Ebenen des OSI-Basisreferenzmodells ansetzt. Dadurch werden Sicherungsmechanismen auch standardmäßig für alle darüberliegenden Schichten angeboten. Da heutzutage sehr viele Protokolle auf TCP/IP basieren, kann mit IPnG ein breites Maß an Vertraulichkeit und Integrität der Daten während der Übertragung erreicht werden.

Der andere Aspekt betrifft die Konzeptionierung der Programmiersprache Java, bei der nicht erst im Nachhinein, sondern schon von Anfang an Sicherheitsbelange bezüglich der Erstellung und Ausführung von sicherem Programmcode berücksichtigt wurden.

Die Sicherheit wird auch in Zukunft weiterhin an Bedeutung gewinnen. Dies liegt zum Einen an der Beliebtheit und rasanten Entwicklung des Web. Unter dem Schlagwort e-commerce werden immer mehr Prozesse über das Internet abgewickelt, die besondere Sicherheitsvorkehrungen benötigen. Hierzu gehören vor allem Online-Banking sowie generell alle Verkaufsprozesse, bei denen Zahlungsdaten des Kunden ausgetauscht werden müssen.

Der zweite wichtige Grund für die zukünftige Relevanz liegt in der Tatsache, dass Geschäftsprozesse immer globaler angelegt werden. Dazu gehört, dass Unternehmensdaten nicht nur über das Intranet erreichbar sein müssen, sondern auch über das Internet, um Zulieferern beispielsweise den Zugang zu ermöglichen. Diese Verzahnung von weltweit verteilten Systemen kann im heute üblichen Konkurrenzkampf nur dann verwirklicht werden, wenn die Sicherheit der Daten gewährleistet ist.

Anhang

Anhang A

Quellcode

A.1 *dcag.security.provider.Cipher*

```
1   package dcag.security.provider;

    /**
     * Defines basic constants for cipher algorithms.
5    *
     * @version 1.00, 99/03/04
     * @author Jochen R&uuml;tschlin
     */
    interface Cipher {
10    /**
       * Constant for selecting encryption mode.
       */
      static final int ENCRYPT_MODE = 0;

15
      /**
       * Constant for selecting decryption mode.
       */
      static final int DECRYPT_MODE = 1;
20  }
```

A.2 *dcag.security.provider.IDEASpec*

```
1   package dcag.security.provider;

    /**
     * Crypto class for the International Data Encryption Algorithm (IDEA).
5    *
     * Defines IDEA specific constants.
     *
     * @version 1.00, 99/03/04
     * @author Jochen R&uuml;tschlin
10   */
    interface IDEASpec extends Cipher {
      /**
       * Number of rounds.
       */
15    static final int NUM_OF_ROUNDS = 8;


      /**
       * Length of the input block in bits.
20     */
      static final int BLOCKLEN = 64;
```

```
    /**
     * Key length in bits.
     */
    static final int KEYLEN = 128;


    /**
     * Block length for basic IDEA operations.
     */
    static final int BLOCKSIZE = 16;

    /**
     * Number of basic blocks to represent one input block.
     */
    static final int INP_BLOCKS = BLOCKLEN/BLOCKSIZE;

    /**
     * Number of basic blocks to represent one key block.
     */
    static final int KEY_BLOCKS = KEYLEN/BLOCKSIZE;
}
```

A.3 *dcag.security.provider.IDEAKey*

```
package dcag.security.provider;

import dcag.security.util.*;

/**
 * Crypto class for the International Data Encryption Algorithm (IDEA).
 *
 * Specifies an IDEA key and calculates the necessary round-keys for
 * encryption and decryption.
 *
 * @version 1.00, 99/03/04
 * @author Jochen R&uuml;tschlin
 */
public class IDEAKey implements IDEASpec, java.security.Key {
  /**
   * Holds the 52 encryption keys.
   */
  private BitSet16[] encryptionKey = new BitSet16[52];

  /**
   * Holds the 52 decryption keys.
   */
  private BitSet16[] decryptionKey = new BitSet16[52];

  /**
   * Constructor for initializing the key with bits from a hex-string. The
   * conversion to a bit set is done by the
   * <a href="dcag.security.util.BitSet128.html#BitSet128(dcag.security.util.String)">
   * <code>BitSet128</code></a> constructor. Afterwards the result is given
   * to the <a href="#initialize"><code>initialize</code></a> method.
   *
   * @param keyString the hexadecimal representation of the key.  Its
   * string length is supposed to be 32 characters long (two characters for
   * one byte).
   */
  public IDEAKey(String keyString) {
    BitSet128 key = new BitSet128(keyString);
    this.initialize(key);
```

```
  }


  /**
   * Constructor for initializing the key with values from a 128 bit
   * block. The key is passed directly to the <code>initialize()</code>
   * methode.
   *
   * @param key the key in a bit set representation.
   */
  public IDEAKey(BitSet128 key) {
    this.initialize(key);
  }


  /**
   * Calculates the 52 encryption and 52 decryption keys. The result is
   * stored into the internal (private) array variables
   * <a href="#encryptionKey"><code>encryptionKey</code></a> and
   * <a href="#decryptionKey"><code>decryptionKey</code></a>.
   *
   * @param key the key in a bit set representation of 128 bit.
   */
  public void initialize(BitSet128 key) {
    int i, j;
    int b = 0;

    // Generate encryption keys

    for (i = 0; i < 6; i++) {
      for (j = 0; j < IDEASpec.KEY_BLOCKS; j++, b++) {
        encryptionKey[b] = new BitSet16(key.shortValue(j));
      }
      key.rotateLeft(25);
    }

    for (j = 0; j < 4; j++, b++) {
      encryptionKey[b] = new BitSet16(key.shortValue(j));
    }


    // Generate decryption keys

    b = 0;
    decryptionKey[b] = new BitSet16(encryptionKey[48]);
    decryptionKey[b++].mulInverse();
    decryptionKey[b] = new BitSet16(encryptionKey[49]);
    decryptionKey[b++].addInverse();
    decryptionKey[b] = new BitSet16(encryptionKey[50]);
    decryptionKey[b++].addInverse();
    decryptionKey[b] = new BitSet16(encryptionKey[51]);
    decryptionKey[b++].mulInverse();
    decryptionKey[b++] = new BitSet16(encryptionKey[46]);
    decryptionKey[b++] = new BitSet16(encryptionKey[47]);

    for (i = IDEASpec.NUM_OF_ROUNDS-1; i > 0; i--) {
      decryptionKey[b] = new BitSet16(encryptionKey[i*6+0]);
      decryptionKey[b++].mulInverse();
      decryptionKey[b] = new BitSet16(encryptionKey[i*6+2]);
      decryptionKey[b++].addInverse();
      decryptionKey[b] = new BitSet16(encryptionKey[i*6+1]);
      decryptionKey[b++].addInverse();
      decryptionKey[b] = new BitSet16(encryptionKey[i*6+3]);
      decryptionKey[b++].mulInverse();
      decryptionKey[b++] = new BitSet16(encryptionKey[i*6-2]);
      decryptionKey[b++] = new BitSet16(encryptionKey[i*6-1]);
    }

    decryptionKey[b] = new BitSet16(encryptionKey[0]);
    decryptionKey[b++].mulInverse();
    decryptionKey[b] = new BitSet16(encryptionKey[1]);
```

```
      decryptionKey[b++].addInverse();
      decryptionKey[b] = new BitSet16(encryptionKey[2]);
      decryptionKey[b++].addInverse();
      decryptionKey[b] = new BitSet16(encryptionKey[3]);
      decryptionKey[b++].mulInverse();
    }


    /**
     * Returns one of the 104 encryption/decryption keys.
     *
     * @param opmode either <code>ENCRYPT_MODE</code> or
     * <code>DECRYPT_MODE</code> (both are defined in the <a
     * href="dcag.security.provider.Cipher.html">Cipher</a> class).  Specifies
     * which key array should be used:
     * <a href="#encryptionKey"><code>encryptionKey</code></a> or
     * <a href="#decryptionKey"><code>decryptionKey</code></a>.
     *
     * @param index the index position of the key inside the key array
     * designated thru <code>opmode</code>.
     *
     * @return The requested key of 16 bit block size.
     *
     * @exception java.lang.IllegalArgumentException If <code>opmode</code>
     * is not <code>ENCRYPT_MODE</code> or <code>DECRYPT_MODE</code>.
     */
    protected BitSet16 getKey(int opmode, int index) throws IllegalArgumentException {
      if (opmode == IDEASpec.ENCRYPT_MODE)
        return encryptionKey[index];
      else if (opmode == IDEASpec.DECRYPT_MODE)
        return decryptionKey[index];
      else
        throw new IllegalArgumentException("Bad mode: " + opmode);
    }


    /**
     * Returns the standard algorithm name this key is for.
     *
     * @return The name of the algorithm this key is for, or null if the
     * algorithm this key is for is unknown.
     */
    public String getAlgorithm() {
      return "IDEA";
    }


    /**
     * Returns the format used to encode the key or null if the key does not
     * support encoding.
     *
     * @return The format used to encode the key.
     */
    public String getFormat() {
      return "NONE";
    }


    /**
     * Returns the encoded key.
     *
     * @return The encoded key, or null if the key does not support encoding.
     */
    public byte[] getEncoded() {
      return null;
    }
  }
```

A.4 *dcag.security.provider.IDEACipher*

```
1   package dcag.security.provider;

    import dcag.security.util.*;
    import java.security.*;
5
    /**
     * Crypto class for the International Data Encryption Algorithm (IDEA).
     *
     * @version 1.01, 99/04/05
10   * @author Jochen R&uuml;tschlin
     */
    public class IDEACipher implements IDEASpec {
      /**
       * Holds the 52 encryption and decryption keys. Gets its values from
15     * the constructor.
       *
       * @see IDEAKey
       * @see IDEACipher
       */
20     private IDEAKey mKey;


       /**
        * Constructs an IDEA cipher with the specified key. The key will be
25      * stored in <a href="#mKey"><code>mKey</code></a>.
        *
        * @param key  contains 52 keys for encryption and 52 keys for decryption.
        *
        * @exception java.security.InvalidKeyException If parameter
30      * <code>key</code> is not of type <a
        * href="dcag.security.provider.IDEAKey.html">IDEAKey</a>.
        *
        * @see #mKey
        */
35     public IDEACipher(IDEAKey key) throws InvalidKeyException {
         if (!(key instanceof IDEAKey))
           throw new InvalidKeyException("I didn't get an IDEAKey.");

         mKey = key;
40     }


       /**
        * Encryption of a block of 64 bit length.
45      *
        * @param input a 64 bit block of data to encrypt.
        *
        * @return The encrypted 64 bit block.
        */
50     public BitSet64 encrypt(BitSet64 input) {
         BitSet64 output;

         output = this.cryptBlock(input, ENCRYPT_MODE);
         return output;
55     }



       /**
60      * Encryption of a byte array. As IDEA uses 64 bit blocks for coding, the
        * length of the input array must be a multiple of 8. This is not
        * processed automatically in this class because simply adding bytes (for
        * example 0x00 bytes) could lead to errors during the interpretation of
        * the data which will be decrypted at a later point in time. The
65      * decrypted message could then be longer as the original input data. In
        * order to avoid such a behavior the user has to implement a kind of
        * wrapper around this method which guarantees that the length of the
        * decrypted data is as long as the byte array of the original input
```

```
     * data. This could be done by adding 4 bytes (an int value) as a length
70   * descriptor in front of the input data specifying the length of the
     * data to be interpreted. If the last 64 bit block is not completed the
     * missing bytes have to be appended to fill the block.  By means of the
     * described wrapper it is possible to ignore the previously added bytes
     * at the end.
75   *
     * @param input a byte array of arbitrary length. The only requirement is
     * that the array length must be a multiple of 8 to fill the last 64 bit
     * block.
     *
80   * @return Encrypted byte array.
     */
    public byte[] encrypt(byte[] input) {
      byte[] output;

85    output = this.cryptBlock(input, ENCRYPT_MODE);
      return output;
    }


90

    /**
     * Decryption of a block of 64 bit length.
     *
     * @param input a 64 bit block of data to decrypt.
95   *
     * @return The decrypted 64 bit block.
     */
    public BitSet64 decrypt(BitSet64 input) {
      BitSet64 output;
100
      output = this.cryptBlock(input, DECRYPT_MODE);
      return output;
    }

105

    /**
     * Decryption of a byte array.
     *
110  * @param input a byte array of arbitrary length. The only requirement is
     * that the array length must be a multiple of 8 to fill the last 64 bit
     * block.
     *
     * @return Encrypted byte array.
115  *
     * @see #encrypt(byte[])
     */
    public byte[] decrypt(byte[] input) {
      byte[] output;
120
      output = this.cryptBlock(input, DECRYPT_MODE);
      return output;
    }

125

    /**
     * IDEA Cipher on a 64 bit block. Encryption/decryption is enabled
     * thru the <code>opmode</code> parameter. Depending on this the 52
     * <a href="dcag.security.provider.IDEAKey.html#encryptionKey"><code>encryptionKey</code></a>s or
130  * <a href="dcag.security.provider.IDEAKey.html#decryptionKey"><code>decryptionKey</code></a>s of
     * <a href="#mKey"><code>mKey</code></a> are used.
     *
     * @param input a BitSet to be en-/decrypted.
     *
135  * @param opmode specifies the mode of cipher: either
     * <code>ENCRYPT_MODE</code> or <code>DECRYPT_MODE</code> (which are
     * defined in the <a href="dcag.security.provider.Cipher.html">Cipher</a>
     * class).
     *
```

```
140     * @return The en-/decrypted 64 bit block; which operation is done
        * depends on <code>opmode</code>.
        *
        * @see IDEAKey
        */
145    private BitSet64 cryptBlock(BitSet64 input, int opmode) {
         BitSet16 x[] = new BitSet16[4];           // subblocks of 'input'
         BitSet16 tmp1 = new BitSet16((short) 0);  // interim result
         BitSet16 tmp2 = new BitSet16((short) 0);  // interim result
         int r;                                    // round counter
150      int k;                                    // key counter


         // break down input block into blocks of 16 bit length

155      for (int i = 0; i < INP_BLOCKS; i++) {
           x[i] = new BitSet16(input.shortValue(i));
         }

         // 8 rounds of IDEA cipher
160
         for (r = 0, k = 0; r < NUM_OF_ROUNDS; r++) {
           x[0].multiply(mKey.getKey(opmode, k++));
           x[1].add(mKey.getKey(opmode, k++));
           x[2].add(mKey.getKey(opmode, k++));
165        x[3].multiply(mKey.getKey(opmode, k++));

           tmp1.xor(x[0], x[2]);
           tmp2.xor(x[1], x[3]);
           tmp1.multiply(mKey.getKey(opmode, k++));
170        tmp2.add(tmp1);
           tmp2.multiply(mKey.getKey(opmode, k++));
           tmp1.add(tmp2);
           x[0].xor(tmp2);
           x[2].xor(tmp2);
175        x[1].xor(tmp1);
           x[3].xor(tmp1);

           // exchange x[1] with x[2]

180        tmp1.assign(x[1]);
           x[1].assign(x[2]);
           x[2].assign(tmp1);
         }

185      // exchange x[1] with x[2]

         tmp1.assign(x[1]);
         x[1].assign(x[2]);
         x[2].assign(tmp1);
190
         // output transformation

         x[0].multiply(mKey.getKey(opmode, k++));
         x[1].add(mKey.getKey(opmode, k++));
195      x[2].add(mKey.getKey(opmode, k++));
         x[3].multiply(mKey.getKey(opmode, k));

         // return result

200      BitSet64 output = new BitSet64(x);
         return output;
       }


205
       /**
        * IDEA Cipher on a byte array. Encryption/decryption is enabled
        * thru the <code>opmode</code> parameter. Depending on this the 52
        * <a href="dcag.security.provider.IDEAKey.html#encryptionKey"><code>encryptionKey</code></a>s or
210     * <a href="dcag.security.provider.IDEAKey.html#decryptionKey"><code>decryptionKey</code></a>s of
```

```
     * <a href="#mKey"><code>mKey</code></a> are used. The length of
     * <code>message</code> is supposed to be a multiple of 8 to fill whole
     * 64 bit blocks.
     *
     * @param message a byte array to be en-/decrypted.
     * Its length is required to be a multiple of 8.
     *
     * @param opmode specifies the mode of cipher: either
     * <code>ENCRYPT_MODE</code> or <code>DECRYPT_MODE</code> (which are
     * defined in the <a href="dcag.security.provider.Cipher.html">Cipher</a>
     * class).
     *
     * @return The en-/decrypted byte array, depending on <code>opmode</code>
     *
     * @see IDEAKey
     */
    private byte[] cryptBlock(byte[] message, int opmode) {
      BitSet64 output;                              // interim result
      byte[] outputVector = new byte[message.length];  // return value
      byte[] inputVector = new byte[8];             // 64 bit block as byte array
      int i, j;                                     // byte counter

      // takes 8 bytes together for a 64 bit input block and crypt it

      for (i = 0; i < message.length/8; i++) {

        // Copy 8 bytes into the inputVector to get a full 64 bit block

        for (j = 0; j < 8; j++) {
          inputVector[j] = message[i*4+j];
        }

        // Do the IDEA cipher

        BitSet64 input = new BitSet64(inputVector);
        output = this.cryptBlock(input, opmode);

        // Copy back 8 bytes form the 64 bit cipher result into the output
        // stream.

        for (j = 0; j < 8; j++) {
          outputVector[i*4+j] = output.byteValue(j);
        }
      }

      return outputVector;
    }
  }
```

A.5 *dcag.security.util.BitSet16*

```
package dcag.security.util;

/**
 * This class implements a 64 bit block for the use in cryptographic
 * functions.
 *
 * @version 1.01, 99/04/05
 * @author Jochen R&uuml;tschlin
 */
public class BitSet16 {
  /**
   * Internal representation of the BitSet.
   */
  private short blockData;

  /**
```

```
     * Size of the BitSet (number of bits).
     */
20   private static final byte blockSize = 16;


     /**
     * Constructs a BitSet from the bits of a short value.
25   */
     public BitSet16(short value) {
       this.blockData = value;
     }

30

     /**
     * Constructs a BitSet from the bits of another 16 bit block.
     */
     public BitSet16(BitSet16 block) {
35     this.blockData = block.shortValue();
     }


     /**
40   * Assigns the 16 bits from a short value to this.
     */
     public void assign(short value) {
       this.blockData = value;
     }
45

     /**
     * Assigns the bits of another 16 bit block to this.
     */
50   public void assign(BitSet16 block) {
       this.blockData = block.shortValue();
     }


55   /**
     * Returns the 16 bits as short value.
     */
     public short shortValue() {
       return this.blockData;
60   }


     /**
     * Left shift of this (this <<= nbits); bits value 0 will be pulled
65   * behind on the right side.
     *
     * @param nbits number of bits to shift.
     */
     public void shiftLeft(int nbits) {
70     this.blockData <<= nbits;
     }


     /**
75   * Right shift of this (this >>>= nbits); bits with value 0 will be
     * pulled behind on the left side.
     *
     * @param nbits number of bits to shift.
     */
80   public void shiftRight(int nbits) {
       // Use of >>> is necessary, because >> pulls behind the signum bit.
       this.blockData >>>= nbits;
     }

85

     /**
     * Left rotation of this; bits vanishing on the left will be put back on
     * the right.
```

```
     *
90   * @param nbits number of bits to rotate.
     */
    public void rotateLeft(int nbits) {
      nbits %= blockSize;
      this.blockData = (short) ((this.blockData << nbits) |
95      (this.blockData >>> (blockSize-nbits)));
    }


    /**
100  * Right rotation of this; bits vanishing on the right will be put back
     * on the left.
     *
     * @param nbits number of bits to rotate.
     */
105 public void rotateRight(int nbits) {
      nbits %= blockSize;
      this.blockData = (short) ((this.blockData >>> nbits) |
        (this.blockData << (blockSize-nbits)));
    }
110

    /**
     * Calculates (this &= block).
     */
115 public void and(BitSet16 block) {
      this.blockData &= block.shortValue();
    }


120 /**
     * Calculates (this |= block).
     */
    public void or(BitSet16 block) {
      this.blockData |= block.shortValue();
125 }


    /**
     * Calculates (this ^= block).
130  */
    public void xor(BitSet16 block) {
      this.blockData ^= block.shortValue();
    }

135

    /**
     * Calculates (this = block1 ^ block2).
     */
    public void xor(BitSet16 block1, BitSet16 block2) {
140   this.blockData = (short) (block1.shortValue() ^ block2.shortValue());
    }


    /**
145  * Calculates (this = ~this).
     */
    public void not() {
      this.blockData = (short) ~this.blockData;
    }
150

    /**
     * Calculates (this += block). Both values are interpreted as positive
     * values (no signum bit). Possible overflow bit is simply cut off.
155  */
    public void add(BitSet16 block) {
      // For unsigned addition cast to next bigger number type and erase MSBs
      // with &-operation.
      int a = ((int) this.blockData) & 0xFFFF;
```

```
160     int b = ((int) block.shortValue()) & 0xFFFF;

        this.blockData = (short) (a + b);
      }

165
      /**
       * Calculates the additive inverse (mod 2**16). Its denoted thru (2**16 -
       * this) & 0xFFFF.
       */
170   public void addInverse() {
        // For unsigned operations cast to next bigger number type and erase
        // MSBs.
        int a = ((int) this.blockData) & 0xFFFF;

175     blockData = (short) ( (0x10000 - a) & 0xFFFF);
      }


      /**
180    * Calculates a slightly modified multiplication (this *= block). If
       * (this == 0) or (block == 0) then (this = 2**16). If the result is 2**16
       * then replace this by 0.
       */
      public void multiply(BitSet16 block) {
185     long a, b, r;

        // For unsigned operations cast to next bigger number type and erase
        // MSBs with &-operation.
        a = ((long) blockData) & 0xFFFF;
190     b = ((long) block.shortValue()) & 0xFFFF;

        if (a == 0)
          r = 0x10001 - b;
        else if (b == 0)
195       r = 0x10001 - a;
        else {
          r = a * b;
          r = (r & 0xFFFF) - (r >>> 16);
          if (r < 0)
200         r = 0x10001 + r;
        }

        this.blockData = (short) (r & 0xFFFF);
      }
205

      /**
       * Calculates the multiplicative inverse (mod 2**16 + 1). Calculation is
       * done directly on the underlying data type of <a
210    * href="#blockData"><code>blockData</code></a> by the Extended Euclidean
       * algorithm.
       */
      public void mulInverse() {
        int r, q, t;
215
        // d = gcd(n1, n2) and values b1, b2 satisfying b1*n1 + b2*n2 = d
        // with n2 = this.blockData.

        if (this.blockData == 0) {
220       this.blockData = (short) 0;
        }
        else {
          int n1 = 0x10001;
          int n2 = ((int) this.blockData) & 0xFFFF; // casting without signum
225       int b1 = 0;
          int b2 = 1;

          do {
            r = (n1 % n2);
230         q = (n1 - r) / n2;
```

```
            if (r == 0) {
              if (b2 < 0)
                b2 = 0x10001 + b2;
235         }
            else {
              n1 = n2;
              n2 = r;
              t = b2;
240           b2 = b1 - q * b2;
              b1 = t;
            }
          } while (r != 0);

245       this.blockData = (short) b2;
        }
      }


250   /**
       * Returns a base 2 string representation of this.
       */
      public String toString() {
        char[] c = new char[blockSize];
255     short v = this.blockData;
        byte i;

        for (i = blockSize-1; i >= 0; i--) {
          c[i] = (char) ((v & 1) + '0');
260       v >>= 1;
        }

        String s = new String(c);
        return s;
265   }
    }
```

A.6 *dcag.security.util.BitSet32*

```
1   package dcag.security.util;

    /**
     * This class implements a 64 bit block for the use in cryptographic
5    * functions.
     *
     * @version 1.01, 99/04/05
     * @author Jochen R&uuml;tschlin
     */
10  public class BitSet32 {
      /**
       * Internal representation of the BitSet.
       */
      private int blockData;
15

      /**
       * Size of the BitSet (number of bits).
       */
20    private static final byte blockSize = 32;


      /**
       * Constructs a BitSet from the bits of a short value.
25     */
      public BitSet32(int value) {
        this.blockData = value;
      }
```

```
30
       /**
        * Constructs a BitSet from the bits of another 32 bit block.
        */
       public BitSet32(BitSet32 block) {
35       this.blockData = block.intValue();
       }


       /**
40      * Assigns the 32 bits from an int value to this.
        */
       public void assign(int value) {
         this.blockData = value;
       }
45


       /**
        * Assigns the bits from another 32 bit block to this.
        */
50     public void assign(BitSet32 block) {
         this.blockData = block.intValue();
       }


55     /**
        * Returns the 32 bits as an int value.
        */
       public int intValue() {
         return this.blockData;
60     }


       /**
        * Left shift of this (this <<= nbits); bits with the value 0 will be
65      * pulled behind on the right side.
        *
        * @param nbits number of bits to shift.  */
       public void shiftLeft(int nbits) {
         this.blockData <<= nbits;
70     }


       /**
        * Right shift of this (this >>>= nbits); bits with the value 0 will be
75      * pulled behind on the left side.
        *
        * @param nbits number of bits to shift.
        */
       public void shiftRight(int nbits) {
80       // Use of >>> is necessary, because >> pulls behind the signum bit.
         this.blockData >>>= nbits;
       }


85     /**
        * Left rotation of this; bits vanishing on the left will be put back on
        * the right.
        *
        * @param nbits number of bits to rotate.
90      */
       public void rotateLeft(int nbits) {
         nbits %= blockSize;
         this.blockData = (short) ((this.blockData << nbits) |
           (this.blockData >>> (blockSize-nbits)));
95     }


       /**
        * Right rotation of this; bits vanishing on the right will be put back
100     * on the left.
```

```
     *
     * @param nbits number of bits to rotate.
    */
    public void rotateRight(int nbits) {
105   nbits %= blockSize;
      this.blockData = (short) ((this.blockData >>> nbits) |
        (this.blockData << (blockSize-nbits)));
    }

110

    /**
     * Calculates (this &= block).
     */
    public void and(BitSet32 block) {
115   this.blockData &= block.intValue();
    }


    /**
120  * Calculates (this |= block).
     */
    public void or(BitSet32 block) {
      this.blockData |= block.intValue();
    }
125

    /**
     * Calculates (this ^= block).
     */
130 public void xor(BitSet32 block) {
      this.blockData ^= block.intValue();
    }


135 /**
     * Calculates (this = block1 ^ block2).
     */
    public void xor(BitSet32 block1, BitSet32 block2) {
      this.blockData = (int) (block1.intValue() ^ block2.intValue());
140 }


    /**
     * Calculates (this = ~this).
145  */
    public void not() {
      this.blockData = (int) ~this.blockData;
    }

150

    /**
     * Calculates (this += block). Both values are interpreted as positive
     * values (no signum bit). Possible overflow bit is simply cut off.
     */
155 public void add(BitSet32 block) {
      // For unsigned addition cast to next bigger number type and erase MSBs
      // with &-operation.
      long a = ((long) this.blockData) & 0xFFFFFFFF;
      long b = ((long) block.intValue()) & 0xFFFFFFFF;
160
      this.blockData = (int) (a + b);
    }


165 /**
     * Calculates (this += value). Both values are interpreted as positive
     * values (no signum bit). Possible overflow bit is simply cut off.
     *
     * @param value Integer value to be added (32 bits).
170  */
    public void add(int value) {
```

```
        // For unsigned addition cast to next bigger number type and erase MSBs
        // with &-operation.
        long a = ((long) this.blockData) & 0xFFFFFFFF;
        long b = ((long) value) & 0xFFFFFFFF;

        this.blockData = (int) (a + b);
      }

      /**
       * Returns a base 2 string representation of this.
       */
      public String toString() {
        char[] c = new char[blockSize];
        int v = this.blockData;
        byte i;

        for (i = blockSize-1; i >= 0; i--) {
          c[i] = (char) ((v & 1) + '0');
          v >>= 1;
        }

        String s = new String(c);
        return s;
      }
    }
```

A.7 *dcag.security.util.BitSet64*

```
package dcag.security.util;

/**
 * This class implements a 64 bit block for the use in cryptographic
 * functions.
 *
 * @version 1.01, 99/04/05
 * @author Jochen R&uuml;tschlin
 */
public class BitSet64 {
  /**
   * Internal representation of the BitSet.
   */
  private long blockData;

  /**
   * Size of the BitSet (number of bits).
   */
  private static final byte blockSize = 64;


  /**
   * Constructs a BitSet from the bits of a long value.
   */
  public BitSet64(long value) {
    this.blockData = value;
  }

  /**
   * Constructs a BitSet from the bits of a byte array vector.
   * Representation in <code>vector</code> is in big endian format (most
   * significant byte at position 0).
   *
   * @param vector array of minimum 8 bytes (to get 64 bits); the rest is
   * ignored.
   */
  public BitSet64(byte[] vector) {
```

```
    this.blockData =  ( (long) vector[0] ) & 0xFF;
    for (int i = 1; i < 8; i++) {
      this.blockData = (( (long) vector[i] ) & 0xFF) | (this.blockData << 8);
    }
  }

  /**
   * Constructs a BitSet from the bits of a 16 bit block array
   * vector.  Representation in <code>vector</code> is in "big endian"
   * format (most significant 16 bit block at position 0).
   *
   * @param vector array of minimum four 16 bit blocks (to get 64 bits);
   * the rest is ignored.
   */
  public BitSet64(BitSet16[] vector) {
    this.blockData =  ((long) vector[0].shortValue()) & 0xFFFF;
    for (int i = 1; i < 4; i++) {
      this.blockData = (((long) vector[i].shortValue()) & 0xFFFF) |
        (this.blockData << 16);
    }
  }


  /**
   * Constructs a BitSet from a hexadecimal string representation. The
   * string is supposed to be in a correct hexadecimal format; no character
   * check is done.
   *
   * @param s string with minimum length of 16 characters (8 bytes with two
   * characters for each byte); the rest is ignored.
   */
  public BitSet64(String s) {
    int nBytes = s.length() / 2;
    int b;

    this.blockData = (long) getByteFromHex(s, 0);
    for (b = 1; b < 8; b++)
      this.blockData = (long) getByteFromHex(s, b) | (this.blockData << 8);
  }


  /**
   * Returns a specified byte value (8 bits) from this.
   *
   * @param i byte index of the big endian represantation of this.
   */
  public byte byteValue(int i) {
    byte r;

    r = (byte) (this.blockData >>> ((7-i) * 8) );
    return r;
  }


  /**
   * Returns a specified short value (16 bits) from this.
   *
   * @param i short index of the big endian represantation of this.
   */
  public short shortValue(int i) {
    short r;

    r = (short) (this.blockData >>> ((3-i) * 16) );
    return r;
  }


  /**
   * Returns a base 2 string representation of this.
   */
  public String toString() {
```

```
        char[] c = new char[blockSize];
        long v = this.blockData;
        byte i;

115     for (i = blockSize-1; i >= 0; i--) {
          c[i] = (char) ((v & 1) + '0');
          v >>= 1;
        }

120     String s = new String(c);
        return s;
      }


125   /**
       * Returns the numeric value of a hexadecimal character. One character
       * represents 4 bits of a byte value (sometimes called a nibble). For
       * example '5' returns 5 and 'f' returns 14. The characters are supposed
       * to be in lower case.
130    */
      private byte getNibble(char c) {
        char b;
        byte r;

135     if ( (c >= '0') && (c <= '9') )
          b = '0';
        else
          b = 'a';

140     r = (byte) (c - b);
        return r;
      }


145   /**
       * Returns a specified byte value out of a hexadecimal string
       * representation.
       *
       * @param pos specifies the big endian byte index inside the string.
150    */
      private byte getByteFromHex(String s, int pos) {
        byte r;

        s.toLowerCase();
155     r = (byte) ( (this.getNibble(s.charAt(2*pos)) << 4) |
                     this.getNibble(s.charAt(2*pos+1)) );
        return r;
      }
    }
```

A.8 *dcag.security.util.BitSet128*

```
1   package dcag.security.util;

    import java.math.BigInteger;

5   /**
     * This class implements a 128 bit block for the use in cryptographic
     * functions.
     *
     * @version 1.01, 99/04/05
10   * @author Jochen R&uuml;tschlin
     */
    public class BitSet128 {
      /**
       * Internal representation of the BitSet. Two long values (each with 64
15     * bits) are needed to get 128 bit data.
       */
```

```
  private long[] blockData = {0, 0};

  /**
   * Size of the underlying base type (long) in bits.
   */
  private static final byte baseSize = 64;

  /**
   * Constructs a BitSet from the bits of a long array vector.
   * Representation in <code>vector</code> is in big endian format (most
   * significant long at position 0).
   *
   * @param vector array of minimum 2 long values (to get 128 bits); the
   * rest is ignored.
   */
  public BitSet128(long[] vector) {
    this.blockData[0] = vector[0];
    this.blockData[1] = vector[1];
  }

  /**
   * Constructs a BitSet from the bits of a 32 bit block array
   * vector.  Representation in <code>vector</code> is in "big endian"
   * format (most significant 32 bit block at position 0).
   *
   * @param vector array of minimum four 32 bit blocks (to get 128 bits);
   * the rest is ignored.
   */
  public BitSet128(BitSet32[] vector) {
    // For unsigned type casting from int to long &-operation is needed.
    this.blockData[0] = (((long) vector[0].intValue()) << 32) |
      (((long) vector[1].intValue()) & 0xFFFFFFFF);
    this.blockData[1] = (((long) vector[2].intValue()) << 32) |
      (((long) vector[3].intValue()) & 0xFFFFFFFF);
  }

  /**
   * Constructs a BitSet from the bits of an integer array
   * vector.  Representation in <code>vector</code> is in "big endian"
   * format (most significant 32 bit block at position 0).
   *
   * @param vector array of minimum four integer (to get 128 bits);
   * the rest is ignored.
   */
  public BitSet128(int[] vector) {
    // For unsigned type casting from int to long &-operation is needed.
    this.blockData[0] = (((long) vector[0]) << 32) |
      (((long) vector[1]) & 0xFFFFFFFF);
    this.blockData[1] = (((long) vector[2]) << 32) |
      (((long) vector[3]) & 0xFFFFFFFF);
  }

  /**
   * Assigns the bits from another 128 bit block to this.
   */
  public BitSet128(BitSet128 block) {
    this.assign(block);
  }

    /**
     * Assigns the bits from a hexadecimal string representation. The string
     * is supposed to be in a correct hexadecimal format; no character check
     * is done.
     *
     * @param s string with minimum length of 32 characters (16 bytes with
```

```
     * two characters for each byte); the rest is ignored. (The will be
     * read from the right!).
     */
    public BitSet128(String s) {
        BigInteger bigInt = new BigInteger(s, 16);

        this.blockData[1] = bigInt.longValue();
        this.blockData[0] = bigInt.shiftRight(64).longValue();
    }


    /**
     * Assigns the bits from another 128 bit block to this.
     */
    public void assign(BitSet128 block) {
      this.blockData[0] = block.highValue();
      this.blockData[1] = block.lowValue();
    }

    /**
     * Assigns the bits from an int value (32 bit) to the specified
     * <code>index</code> position.
     *
     * @param index big endian position where the bits from integer
     * <code>value</code> should copied to. No boundary check will be done on
     * this value.
     *
     * @param value 32 bits to be copied.
     *
     * @see #intValue
     */
    public void assign(int index, int value) {
          switch (index) {
          case 0:
          case 2:
              blockData[index%2] = (blockData[index%2] & 0xFFFFFFFF) |
                  (((long) value) << 32);
              break;
          case 1:
          case 3:
              blockData[index%2] = (blockData[index%2] & (~0xFFFFFFFF)) |
                  ((long) value);
              break;
          }
    }


    /**
     * Returns 64 bit half with least significant bits into a long value.
     */
    public long lowValue() {
      return this.blockData[1];
    }


    /**
     * Returns 64 bit half with most significant bits into a long value.
     */
    public long highValue() {
      return this.blockData[0];
    }


    /**
     * Returns an int value (32 bit) from position <code>index</code> of this.
     *
     * @param index specifies the short position with big endian format of
     * this.
     *
     * @see #assign(int,int)
     */
```

```
  public int intValue(int index) {
    return (int) (this.blockData[index/2] >>> ((1-(index%2))*32));
  }


    /**
     * Returns a short value (16 bit) from position <code>index</code> of this.
     *
     * @param index specifies the short position with big endian format of
     * this.
     */
    public short shortValue(int index) {
        short rv;  // return value

        if (index < 4) {
            rv = (short) (blockData[0] >>> ((3-index) * 16) );
        }
        else {
            rv = (short) (blockData[1] >>> ((7-index) * 16) );
        }

        return rv;
    }


    /**
     * Returns a byte value (8 bit) from position <code>index</code> of
     * this.
     *
     * @param index specifies the byte position with big endian format of
     * this.
     */
    public byte byteValue(int index) {
        byte rv;  // return value

        if (index < 8) {
            rv = (byte) (blockData[0] >>> ((7-index) * 8) );
        }
        else {
            rv = (byte) (blockData[1] >>> ((15-index) * 8) );
        }

        return rv;
    }


  /**
   * Left rotation of this; bits vanishing on the left will be put back on
   * the right.
   *
   * @param nbits number of bits to rotate.
   */
  public void rotateLeft(int nbits) {
    long dummy = blockData[0];

    nbits %= (2*baseSize);

    this.blockData[0] = (long) ((this.blockData[0] << nbits) |
                                (this.blockData[1] >>> (baseSize-nbits)));

    this.blockData[1] = (long) ((this.blockData[1] << nbits) |
                                (dummy >>> (baseSize-nbits)));
  }


    /**
     * Returns a base 10 string representation of this.
     */
    public String toString() {
        return this.toString(10);
    }
```

```
230
        /**
         * Returns a string representation of this.
         *
235      * @param radix Radix of the output string. Possible values are 2, 8,
         * 10 and 16. Only radix returns a representation with a signum, the
         * other are unsigned integers.
         */
        public String toString(int radix) {
240         String s[] = {"", ""};
            Long l = new Long(this.blockData[0]);

            switch (radix) {
            case 2:
245             s[0] = l.toBinaryString(this.blockData[0]);
                s[1] = l.toBinaryString(this.blockData[1]);
                break;
            case 8:
                s[0] = l.toOctalString(this.blockData[0]);
250             s[1] = l.toOctalString(this.blockData[1]);
                break;
            case 10:
                s[0] = l.toString(this.blockData[0]);
                s[1] = l.toString(this.blockData[1]);
255             break;
            case 16:
                s[0] = l.toHexString(this.blockData[0]);
                s[1] = l.toHexString(this.blockData[1]);
                break;
260         }

            return (s[0] + s[1]);
        }
    }
```

Literatur- und Quellenverzeichnis

Wegen der Dynamik des Internets können die hier angegebenen URLs nur als Momentaufnahme (Stand 5. April 1999) betrachtet werden.
Im Folgenden aufgeführte RFC-Beiträge können beispielsweise unter der World Wide Web Adresse `http://www.leo.org/pub/comp/doc/standards/rfc/index.html` nachgeschlagen werden.

Amoroso, E. G. (1994). *Fundamentals of Computer Security Technology.* Englewood Cliffs, New Jersey: Prentice-Hall International. ISBN 0-13-305541-8.

Bauer, F. L. (1993). *Kryptologie: Methoden und Maximen.* Berlin · Heidelberg · New York [u. a.]: Springer-Verlag. ISBN 3-540-56356-3.

Bauer, F. L. (1995). *Entzifferte Geheimnisse: Methoden und Maximen der Kryptologie.* Berlin · Heidelberg · New York [u. a.]: Springer-Verlag. ISBN 3-540-58118-9. Die vorliegende Fassung beruht auf dem Lehrbuch „Kryptologie: Methoden und Maximen" des Autors, das in 2. Auflage 1994 im Springer-Verlag erschienen ist.

Biller, H. (1995). *Datenschutz und Datensicherheit.* Folien zur Vorlesung an der Universität Ulm WS 94/95. Siemens Nixdorf Informationssysteme AG.

BSI ▷ Bundesamt für Sicherheit in der Informationstechnik, BSI (Januar 1997). Elektronisches Bezahlen/Faltblatt Ö 18. Godesberger Allee 183, 53175 Bonn.

Carter, M. G., Karger, P. A. und Lipner, S. B. (Oktober 1981). *Protecting Data and Information: A Workshop in Computer Security.* Maynard, Massachusetts: Digital Equipment Corporation (DEC).

Chamberlin, D. (1998). *A Complete Guide to DB2 Universal Database.* The Morgan Kaufmann Series in Data Management Systems. San Francisco, California: Morgan Kaufmann Publishers, Inc. ISBN 1-55860-482-0.

Chen, P. P. (1976). The Entity-Relationship Model: Toward a Unified View of Data. ACM Transactions on Database Systems 1(1), 9–36.

DB2-SQL ▷ International Business Machines Corporation (17. November 1995). *DATABASE 2 – SQL Reference for common servers, Version 2.*

Denning, D. E. (Mai 1976). A Lattice Model of Secure Information. Communications of the ACM 19(5), 236–243.

Diffie, W. und Hellman, M. E. (November 1976). New Directions in Cryptography. IEEE Transactions on Information Theory 22(6), 644–654.

FIPS PUB 46-2 ▷ U.S. Department of Commerce/National Institute of Standards and Technology – NIST (30. Dezember 1993). *Federal Information Processing Standards Publication 46-2: Data Encryption Standard (DES).* Beispielsweise verfügbar

unter [Internet, WWW] http://www.dice.ucl.ac.be/crypto/standards/fips/html/fip46-2.htm.

FIPS PUB 74 ▷ U.S. Department of Commerce/National Institute of Standards and Technology – NIST (1. April 1981). *Federal Information Processing Standards Publication 76: Guidelines for Implementing and Using the NBS Data Encryption Standard*. Beispielsweise verfügbar unter [Internet, WWW] http://www.dice.ucl.ac.be/crypto/standards/fips/html/fip74.htm.

Flanagan, D. (1998). *Java in a Nutshell – Deutsche Ausgabe für Java 1.1* (2. erweiterte und aktualisierte Auflage). Cambridge · Köln · Paris u. a.: O'Reilly. ISBN 3-90721-100-9.

Freier, A. O., Karlton, P. und C., K. P. (18. November 1996). *The SSL Protocol Version 3.0*. Internet-Draft. Beispielsweise verfügbar unter [Internet, WWW] http://home.netscape.com/eng/ssl/draft302.txt.

Fritzinger, J. S. und Mueller, M. (1996). Java Security. Whitepaper, Sun Microsystems, Inc. http://java.sun.com/security/whitepaper.ps.

Garfinkel, S. (März 1995). *PGP: Pretty Good Privacy*. Sebastopol, California: O'Reilly & Associates, Inc. ISBN 1-56592-098-8.

Garfinkel, S. und Spafford, G. (April 1996). *Practical UNIX & Internet Security* (2. Auflage). Sebastopol, California: O'Reilly & Associates. ISBN 1-56592-148-8.

Garfinkel, S. und Spafford, G. (Juni 1997). *Web Security & Commerce*. Risks, Technologies, and Strategies. Cambridge · Köln · Paris [u. a.]: O'Reilly & Associates. ISBN 1-56592-269-7.

Gerhardt, W. (1993). *Zugriffskontrolle bei Datenbanken*, Band 3 aus *Schriftenreihe Sicherheit in der Informationstechnik*. München · Wien: R. Oldenbourg Verlag. ISBN 3-486-22186-8.

Hergula, K. (Mai 1998). *Datenbank-Middleware-Systeme: Eine vergleichende Untersuchung und Bewertung kommerzieller Werkzeuge*. Diplomarbeit. Universität Ulm, Fakultät für Informatik.

Hergula, K. und Rezende, F. d. F. (Juni 1998). Eine gründliche Untersuchung von Datenbank-Middleware-Technologien. Technischer Bericht FT3/E-1998-002, Daimler-Benz AG, Forschung und Technologie 3: Prozeßkette Produktentwicklung, Ulm.

Hobbes ▷ Zakon, R. H. (12. April 1998). *Hobbes' Internet Timeline v3.3*. [Internet, WWW] http://www.isoc.org/guest/zakon/Internet/History/HIT.html.

Horster, P. (1985). *Kryptologie*. Nummer 47 in Reihe Informatik. Mannheim · Wien · Zürich: B. I.-Wissenschaftsverlag. ISBN 3-411-03106-9.

Hunt, C. (1995). *TCP/IP Netzwerk Administration*. Bonn: O'Reilly/International Thomson Verlag. ISBN 3-930673-02-9.

Informatiktaschenbuch ▷ Werner, D., Hrsg. (1995). *Taschenbuch der Informatik* (2., völlig neu bearbeitete Auflage). Leipzig: Fachbuchverlag. ISBN 3-343-00892-3.

ITU-T ▷ International Telecommunication Union · Telecommunication Standardization Sector (Juni 1997). ITU-T Recommendation X.509: *Information Technology – Open Systems Interconnection – The Directory: Authentication Framework.*

Kahn, D. (1996). *The Codebreakers – The Story of Secret Writing; The Comprehensive History of Secret Communication from Ancient Times to the Internet* (durchgesehene und erweiterte Auflage). New York: Scribner (Simon & Schuster Inc.). ISBN 0-684-83130-9.

Keedy, J. L. (Weihnachten 1994). *Betriebssystemkonzepte (Teil 1).* Vorlesungsskript WS 94/95. Universität Ulm, Fakultät für Informatik (Abteilung Rechnerstrukturen).

Knudsen, J. (Mai 1998). *Java Cryptography.* The Java Series. Cambridge · Köln · Paris u. a.: O'Reilly. ISBN 1-56592-402-9.

Köbler, J. (Februar 1997). *Kryptologie.* Vorlesungsskript WS 95/96. Universität Ulm, Fakultät für Informatik (Abteilung Theoretische Informatik).

Köbler, J. (1999). *Kryptologie.* Heidelberg · Berlin · Oxford: Spektrum Akademischer Verlag. ISBN 3-82740-258-1. (Noch nicht erschienen).

Lamport, L. (1981). Password Authentication with Insecure Communication. Communications of the ACM 24, 770–772.

Lampson, B. W. (1971). Protection. In *Proceedings of the the Fifth Symposium of Information Sciences and Systems*, Princeton, Seiten 437 ff. Nachgedruckt in *ACM Operating Systems Review* 8(1), 18–24, Januar 1974.

Lawton, G. (August 1998). Biometrics: A new Era in Security. COMPUTER, Innovative Technology for Computer Professionals 31(8), 16–18.

Luckhardt, N. (1998). Nicht ganz dicht – Jugendliche Hacker knacken T-Online. c't, Magazin für Computer-Technik 7/1998, 62–65.

Menezes, A. J., van Oorschot, P. C. und Vanstone, S. A. (1996). *Handbook of Applied Cryptography.* The CRC Press Series on Discrete Mathematics and Its Applications. Boca Raton · New York · London · Tokyo: CRC Press. ISBN 0-8493-8523-7.

Oaks, S. (Mai 1998). *Java Security.* The Java Series. Cambridge · Köln · Paris u. a.: O'Reilly. ISBN 1-56592-403-7.

ORA-ADM ▷ Bobrowski, S. (Juli 1993). *ORACLE7 Server – Administration.* ORACLE Corporation.

ORA-SQL ▷ Linden, B. (Dezember 1992). *ORACLE7 Server – SQL Language Reference Manual.* ORACLE Corporation.

ORACLE ▷ Bobrowski, S. (Juli 1993). *ORACLE7 Server – Begriffe.* ORACLE Corporation.

Pereira, R. und Bhattacharya, P. (19. Februar 1998). *IPSec Policy Data Model.* Internet-Draft. Im Internet (WWW) zu suchen unter `draft-ietf-ipsec-policy-model-00.txt`.

PKCS #1 ▷ RSA Laboratories (1. November 1993). *PKCS #1: RSA Encryption Standard.* Version 1.5. RSA Data Security, Inc. Public-Key Cryptography Standards (PKCS). Verfügbar unter [Internet, WWW] `ftp://www.rsa.com/pub/pkcs/ps/pkcs-1.ps`.

Platzer, W. (1996). *World Wide Web Security*. Diplomarbeit. Technische Universität Graz · Institut für Angewandte Informationsverarbeitung und Kommunikationstechnologie. Beziehbar über [Internet, WWW] `http://www.iaik.tu-graz.ac.at/~wplatzer/Diplom/Title.html`.

Poe, E. A. (1989). Der Goldkäfer (*The Gold-Bug*). In *Erzählungen*, Nummer 8619 in der Universal-Bibliothek, Seiten 259–303. Stuttgart: Reclam. ISBN 3-15-008619-1.

Rescorla, E. und Schiffman, A. (Mai 1996a). *Security Extensions for HTML*. Internet-Draft. Terisa Systems, Inc. Beispielsweise verfügbar unter [Internet, WWW] `http://www.terisa.com/shttp/shtml2.txt`.

Rescorla, E. und Schiffman, A. (Mai 1996b). *The Secure HyperText Transfer Protocol*. Internet-Draft. Terisa Systems, Inc. Beispielsweise verfügbar unter [Internet, WWW] `http://www.terisa.com/shttp/1.2.1.txt`.

Rezende, F. d. F., Hermsen, U., Oliveira, G. d. S., Pereira, R. C. G., Rütschlin, J. und Schneider, P. (September 1998). The Database Access Interface in MEntAs: Architecture and Functionality. Technischer Bericht FT3/E-1998-003, Daimler-Benz AG, Forschung und Technologie 3: Prozeßkette Produktentwicklung, Ulm.

RFC 1752 ▷ Bradner, S. und Mankin, A. (Januar 1995). *The Recommendation for the IP Next Generation Protocol*. Request for Comments (RFC) 1752.

RFC 1825 ▷ Atkinson, R. (August 1995). *Security Architecture for the Internet Protocol*. Request for Comments (RFC) 1825.

RFC 1826 ▷ Atkinson, R. (August 1995). *IP Authentication Header*. Request for Comments (RFC) 1826.

RFC 1827 ▷ Atkinson, R. (August 1995). *IP Encapsulating Security Payload (ESP)*. Request for Comments (RFC) 1827.

RFC 1883 ▷ Deering, S. und Hinden, R. M. (Dezember 1995). *Internet Protocol, Version 6 (IPv6) Specification*. Request for Comments (RFC) 1883.

Rivest, R. L., Shamir, A. und Adleman, L. M. (Februar 1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21(2), 120–126.

RSA-FAQs v4.0 ▷ RSA Laboratories (1998). *Frequently Asked Questions About Today's Cryptography v4.0*. Verfügbar unter [Internet, WWW] `http://www.rsa.com/rsalabs/faq/index.html`.

Rueß, C., Ludwig, B. und Zapf, M. (Oktober 1998). Java und Sicherheit. Technischer Bericht FT3/K-1998-007, Daimler-Benz AG, Forschung und Technologie, Ulm.

Schneier, B. (1997). *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C* (1., korrigierter Nachdruck). Reihe Informationssicherheit. Bonn · Reading, Massachusetts [u. a.]: Addison-Wesley. ISBN 3-89319-854-7. Die amerikanische Originalausgabe ist erschienen unter dem Titel: Applied Cryptography, Bruce Schneier, ISBN 0-471-11709-9.

Tanenbaum, A. S. (1992). *Computer Netzwerke* (2. Auflage)., Abschnitt 8.3.2 Die Datenverschlüsselungsnorm DES, Seiten 610–618. Attenkirchen: Wolfram's Fachverlag. ISBN 3-925328-79-3.

Tanenbaum, A. S. (1994). *Verteilte Betriebssysteme*. München · New York · Toronto: Prentice Hall. ISBN 3-930436-23-X.

Weck, G. (1984). *Datensicherheit: Methoden, Maßnahmen und Auswirkungen des Schutzes von Informationen*. Leitfäden der angewandten Informatik. Stuttgart: B. G. Teubner. ISBN 3-519-02472-1.

Wobst, R. (1998). *Abenteuer Kryptologie: Methoden, Risiken und Nutzen der Datenverschlüsselung* (2., überarbeitete Auflage). Reihe Informationssicherheit. Bonn · Reading, Massachusetts [u. a.]: Addison Wesley Longman Verlag GmbH. ISBN 3-8273-1413-5.

Schlagwortverzeichnis

Name: Jochen Rütschlin Matr.Nr. 260 380

Erklärung

Ich erkläre, dass ich die Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 4. Mai 1999

(Unterschrift)