CARL HANSER VERLAG

Paul Levi, Ulrich Rembold

Einführung in die Informatik

für Naturwissenschaftler und Ingenieure

3-446-21932-3

www.hanser.de

9 Spezielle Netzwerkdienste und Middleware-Unterstützungen für verteilte Anwendungen

Im vorangegangenen Kapitel wurden die wichtigsten Dienste beschrieben, die von der Anwendungsschicht der OSI- oder TCP/IP-Referenzmodelle mit jeweils verschiedenen Protokollen angeboten werden. Bei diesen Diensten gibt es eine größere Menge von Übereinstimmungen in den einzelnen Dienstarten, wie bei E-Mail, Dateitransfer und Namendienst, aber auch Unterschiede, die sich vor allem durch die Bereitstellung von Middleware-Serviceelementen (z.B. ROSE von OSI) oder von HTTP-Übermittlungen klar angeben lassen. In diesem Kapitel geht es nicht darum, diese Dienste im größeren Detail einzeln zu besprechen und andere noch nicht erwähnte Dienste, wie Gopher [RFC 1436] oder das Finger Protocol [RFC 1288], die beide vom Internet angeboten werden, zusätzlich aufzuführen, da es hierzu genügend ausführliche Beschreibungen gibt. Unseren Augenmerk richten wir hingegen darauf, wie verteilte Anwendungen durch Basisdienste verteilter Systeme (typischerweise Internet) unterstützt und durch welche grundlegenden Ansätze der Middleware sie entwickelt und ausgeführt werden.

Die Basisdienste z.B. vom Internet für verteilte Anwendungen, die wir besprechen werden, sind: Namen-, Zeit- und Sicherheitsdienste. Eine wesentliche Grundlage für verteilte Anwendungen stellt die Interprozess-Kommunikation dar. Hier ist vor allem der ältere Client-Server-basierte Prozedur-Fernaufruf zu nennen. Das neuere Konzept der objektorientierten Programmierung hat in seiner Weiterentwicklung zur Kommunikation zwischen verteilten Objekten, die in einzelnen Prozessen definiert sind, geführt. Hier ist beispielsweise der Methoden-Fernaufruf (engl. Remote Method Invocation, RMI) von Java zu nennen. Beide Arten der Fernaufrufe werden wir in Kapitel 10.5.2.2 vorstellen. So gehört zwar der Methoden-Fernaufruf noch zum Aufgabenbereich heutiger Middleware und nicht zu dem von Betriebssystemen, doch wird diese Zugehörigkeit mehr durch die beschränkte Funktionalität vorhandener Netzwerkbetriebssysteme vorgegeben und weniger durch deren prinzipielle Aufgabenstellung, weshalb wir diesen Punkt im Kapitel über Betriebssysteme untergebracht haben.

Daneben existiert aber noch ein weiterer wesentlicher Aspekt der verteilten Programmiersprache Java, der für uns hier wichtig ist. Gemeint ist die Integration von typischen Middleware-Diensten wie Unabhängigkeit von Rechnerplattformen (Rechner und Betriebssystem), Erzeugung flacher Strukturen für die Datenübertragung (engl. marshalling), Bereitstellung entfernter Methodenaufrufe und Erzeugung mobiler Codes in einer Programmiersprache. Weitere Details hierzu finden Sie in Abschnitt 9.4.

Mit der Integration von verteilten Diensten in eine Sprache ist dem Programmierer aber noch kein umfassendes Entwicklungswerkzeug und Programmierwerkzeug für verteilte Anwendungen in die Hand gegeben. Solche werden durch CORBA konzipiert (Abschn. 9.5). CORBA ist eine exemplarische und weit verbreitete Architektur für eine Middleware, die das Programmiermodell der verteilten Objekte als Kernprinzip einsetzt und alle Interaktionen und Dienste auf Objekte und ihre Wechselwirkungen abbildet.

Wir beenden dieses Kapitel, indem wir uns mit den wesentlichen Anwendungsbereichen, den Perspektiven und den gesellschaftlichen Auswirkungen der Dienste verteilter Systeme und Anwendungen (Abschn. 9.6) beschäftigen.

9.1 Namendienste

Namen sind symbolische Bezeichner von Objekten, deren einzelne Zeichen aus einem vorgegebenen Alphabet entnommen werden. Solche Objekte in verteilten Systemen können Dateien, Kommunikationskanäle, Variablen, Benutzer, Netzwerke, Hosts, Web-Seiten usw. sein. Namen sind mnemonisch, d.h. sie sagen etwas über die Bedeutung des Objektes aus, sind einfacher zu merken und können unverändert beibehalten werden, wenn das Objekt an eine andere Lokation verschoben (relokiert) wird. Adressen sind meist numerische Bezeichner von Objekten, welche den Ort der Speicherung angeben, aber das Objekt nicht beschreiben. Eine Abweichung von dieser Definition bilden E-Mail-Adressen. Sie sind Namen, wie die einer Web-Seite, aber bei der postalischen Briefvorlage aus Papier hat es sich eingebürgert, alle Angaben, einschließlich des Namens, als Adresse zu bezeichnen. Dieser Brauch wurde vom Internet übernommen.

Die Hauptaufgabe eines *Namendienstes* besteht darin, Namen in Adressen umzuwandeln. Diese Transformation kann mehrstufig sein und wird als Namenresolution bezeichnet. Sie wird statt der unmittelbaren Adressierung verwendet, um die Lokationstransparenz des aufgerufenen Objektes sicherzustellen. Solche Namenauflösungen werden typischerweise bei der Übergabe einer URL an einen Web-Browser durchgeführt. Wird etwa *www.informatik.uni-stutt-gart.de/ipvr/bv/bv_home.html* eingegeben, was die Homepage der Abteilung Bildverstehen im Institut IPVR in der Fakultät Informatik der Universität Stuttgart identifiziert, startet der Web-Browser als ein Client eine Anfrage an einen Namendienst, der auf einem oder mehreren Namen-Servern installiert ist, und er erhält die IP-Adresse (129.69.211.1) zurück, die dem strukturierten Domänennamen *informatik.uni-stuttgart.de* zugeordnet wurde. Danach baut der Browser eine TCP/IP-Verbindung auf, um die Seite von dem Host zu laden, dessen IP-Adresse übergeben wurde. Der letzte Teil der URL, der den Pfadnamen auf dem Server angibt, wird von der Dateiverwaltung umgesetzt, um die relevante Datei lokal zu finden. Entsprechendes gilt für eine E-Mail-Adresse, wobei durch das Symbol @ (at-Zeichen) der Bestimmungstyp E-Mail festgelegt wird.

Ferner sollte jeder Dienst über einen lokationstransparenten Namen und nicht über eine Adresse angefordert werden, denn die Relokation eines Dienstes auf einen anderen Server zieht eine Adressänderung nach sich, die aber der Anwender nicht selbst umsetzen sollte. Die Aktualisierung der Dienstverlegung gehört mit zum Aufgabenbereich eines Namendienstes.

9.1.1 Namenraum

Die wesentliche Datenbasis, auf die ein Namendienst intern zugreift, ist ein *Namenraum*, der durch die Menge aller erlaubten Namen aufgebaut wird. Dabei können wir ganz allgemein einen Namen in einem solchen Raum als eine zweikomponentige

Namenstruktur beschreiben: *Servername/Objektname*, bei der jeder dieser beiden Hauptbezeichner in weitere Namenkomponenten untergliedert werden kann. In unserem Beispiel gibt *www.informatik.uni-stuttgart.de* den Rechnernamen an und *ipvr/bv/bv_home.html* ist der Name des Objektes.

Ein Namenraum kann durch verschiedene Eigenschaften (global, lokal; homogen, heterogen) und Strukturen (flach, hierarchisch) charakterisiert werden. Er ist *global*, wenn ein Objekt durch denselben Namen eindeutig identifiziert wird, unabhängig von welchem Rechner, in welchem Teilnetz, es aufgerufen wird. Ein globaler Namenraum stellt eine *Namentransparenz* her. Gleichgültig, von welchem Ort aus ein Objekt aufgerufen wird, der Name ist immer der gleiche.

Die Namentransparenz ist nicht mit der *Relokationstransparenz* oder *Ortsunabhängigkeit* (Ortsinvarianz) eines Namens zu verwechseln, denn Letztere fordert die unveränderte Beibehaltung eines Objektnamens, wenn das Objekt relokiert wird, d.h. nicht während seiner Ausführungen an einem anderen Ort gespeichert wird (Abschn. 10.7.5.1). Dies entspricht der Beibehaltung des persönlichen Namens bei einem Umzug, nur die Adresse ändert sich, nicht der Name.

Der übliche Namendienst ist namentransparent (Name ist unabhängig vom Aufrufort), lokationstransparent (Name verbirgt den Speicherungsort) und relokationstransparent (Name ist unabhängig vom Speicherungsort). Allerdings gilt das nur, solange man in einem System bleibt. Wenn man jedoch den Arbeitgeber oder den Internet-Anbieter wechselt, muss z.B. die E-Mail-Adresse gewöhnlich auch gewechselt werden.

Wenn die Verschiebung eines Dienstprozesses während der Laufzeit erfolgt, d.h. wenn er migriert, dann wird seine Migrationstransparenz von einem Namendienst üblicherweise nicht gewährleistet, da hierfür weiterreichende Maßnahmen notwendig sind [Coulouris, Dollimore, Kindberg 01].

Ein Namenraum heißt *lokal*, wenn der Name eines Objektes nur auf einem Rechner (in einem Teilnetz) gültig ist und von einem anderen Teilsystem nicht aufgerufen werden kann.

Ein Namenraum ist *homogen*, wenn alle darin verwendeten Namen nach demselben Schema aufgebaut werden. Dies gilt insbesondere auch dann, wenn viele Teil-Namenräume zu einem globalen Namenraum zusammengefügt werden. Er heißt heterogen, wenn einzelne Teilräume verschiedene Namenschemata benutzen. Der Namendienst von DCE (Distributed Computing Environment) arbeitet mit einem heterogenen Namenraum [OSF 92].

Ein einziger unstrukturierter Namenraum mit flachen Namen (z.B. # A12B16C13F) enthält zwar lokationstransparente Bezeichner, wäre aber zu groß und zu ineffizient, um beispielsweise alle Namen, die im Internet gebraucht werden, aufzunehmen. Daher sind hierarchisch strukturierte Namenräume für verteilte Systeme besser geeignet. Wir wollen im Folgenden am Beispiel von DNS (Domain Name System), das im Internet eingesetzt wird und den höchsten Verbreitungsgrad besitzt, einen hierarchischen Namenraum und die Funktionsweise eines Namensystems genauer beschreiben [RFC 1034] und [RFC 1035].

DNS verwendet einen globalen, homogenen und hierarchisch strukturierten Namenraum, der Domänen als Namen einsetzt. Die wichtigsten *Domänennamen*, kurz Domä-

nen, sind die generischen Domänen, welche nach Organisationsformen eingeteilt werden, wie *com* (commercial organisations), *edu* (educational institution), *org* (international organisation) und net (network support cenntre), und die geografische Klassifikation, wie *de* (Deutschland), *us* (United States) und *fr* (France). Jede dieser Hauptdomänen hat Unterdomänen, diese besitzen wiederum weitere Unterdomänen usw. Sämtliche Domänen werden in einem Baum dargestellt. Abbildung 9.1 zeigt die Strukturierung des Namenraumes nach Domänen. Ein Blatt kann ein einzelner Host sein, aber auch ein Institut mit einem eigenen Subnetz und weiteren Hosts. Die Vergabe von Domänen an Internet-Anbieter erfolgt in Deutschland (de) über die Firmenvereinigung DENIC [www.denic.de].

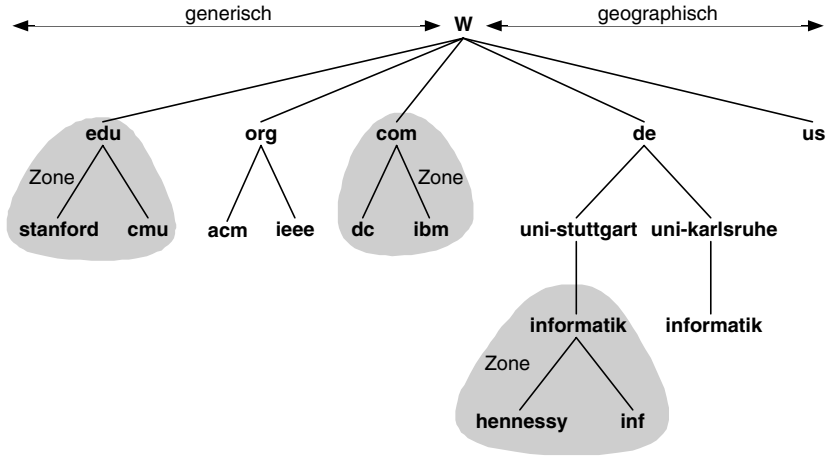

Abbildung 9.1: Hierarchischer DNS-Namenraum, der nach Domänennamen aufgeteilt ist. Die Zonen verdeutlichen die Zuordnung von Teilräumen an einen oder mehrere Namen-Server

Der Name eines Servers (Hosts), wie wir ihn in Abschn. 8.2.3 bei der Beschreibung einer URL gesehen haben, setzt sich zusammen aus dem gesamten Pfadnamen, der bis zur Wurzel jeden neuen Domänennamen, durch einen Punkt getrennt, neu anfügt. So ist der Domänenname für den Host mit dem Namen Hennessy: *hennessy.informatik.uni-stuttgart.de.* Namen, welche die Namen sämtlicher Vorgänger bis zur Wurzel (ohne diese selbst zu nennen) enthalten, heißen *absolute* Namen (vgl. Abschn. 10.7.4). Diese identifizieren ein Objekt global eindeutig. Ein relativer Name enthält nur einen Teil des Namenpfades und identifiziert ein Objekt nur im Kontext seiner Vorgänger eindeutig. So ist der Name hennessy.informatik nicht eindeutig und man benötigt die Vorgängernamen *uni-stuttgart.de.* DNS operiert mit absoluten Namen.

Der DNS-Namenraum wird als eine verteilte Datenbasis implementiert. Dies bedeutet, dass einzelne Domänen oder Gruppierungen von Domänen von einem oder mehreren Namen-Servern verwaltet werden. Der gesamte DNS-Namenraum wird dadurch in „Server-Zonen“ aufgeteilt, wobei der durch eine Zone abgedeckte Namenraum durch die zugeteilten Namen-Server bedient wird. Diese Zoneneinteilung führt dazu, dass einzelne Namenkomponenten auf unterschiedlichen Rechnern bearbeitet werden, aber

die interne verteilte Verwaltungsstruktur bleibt dem Anwender verborgen, sodass eine verteilte Namenverwaltung bereitgestellt wird, welche eine Transparenz der Namen und der Speicherungsorte gewährleistet.

Der Aufgabenbereich eines Name-Servers von DNS besteht zwar primär in der Zuordnung von Name zu IP-Adresse, doch muss er darüber hinaus noch eine Reihe weiterer Aufgaben lösen. Hierzu zählen die Interaktionen (Beauftragung, Kooperation) mit anderen Name-Servern, die Verwaltung lokaler Kontextinformation (Domänenname, Gültigkeitsdauer, Informationsklasse, Recordtyp, IP-Adresse), welche in so genannten *Resource Records*, wie hier die Namen-Server-Tabellen genannt werden, gespeichert wird, Replikation von Resource Records auf anderen Name-Servern und Pufferung von häufig gebrauchten Resource Records anderer Name-Server von anderen Zonen im lokalen Speicher. Eine Replikation steigert die Verfügbarkeit und Zuverlässigkeit eines Namendienstes durch die Techniken der Redundanz und Datei-Pufferung (engl. caching) und wird zur Effizienzsteigerung der Namenresolution in Zwischenspeichern eingesetzt. Beide Maßnahmen dienen der Gewährleistung der Dienstqualität einer Namenverwaltung und werden im folgenden Abschnitt näher behandelt.

9.1.2 Namenresolution

Jeder Namendienst wie DNS speichert Millionen von Einträgen bezüglich eines globalen Namenraums. Diese Information darf nicht auf einem Name-Server abgelegt werden, sondern muss auf viele miteinander verbundene Name-Server, nach dem Einteilungsschema in Zonen eines hierarchischen Namenraums, verteilt werden. Für jede Zone gibt es mindestens einen Name-Server, der für die Namenresolution in dieser Zone verantwortlich ist. Der Resolutionsvorgang geschieht nach folgendem Client-Server-Interaktionsmuster: Ein Client stellt an einen Name-Server eine Anfrage nach einer Namenresolution. Wenn dieser den Namen kennt, d.h. in seinem Resource Record eingetragen hat, liefert er die IP-Adresse sofort zurück. Wenn er ihn nicht kennt, was leicht möglich ist, weil der Namenraum partitioniert ist, werden verschiedene Varianten von Interaktionsmustern angewendet, um eine Namennavigation, wie der Resolutionsprozess genannt wird, durchzuführen. Bleibt die Kontrolle der Namenresolution beim Client, dann beauftragt er nach dem Muster einer Einfachbeauftragung einen Name-Server nach dem andern, bis der Name vollständig aufgelöst ist (iterative Namennavigation), oder er beschleunigt den Vorgang durch eine Fließbandbeauftragung. Wenn die Namenresolution in die Zuständigkeit des zuerst angesprochenen Name-Servers fällt, kann dieser entweder eine Fließbandbeauftragung (vgl. Abb. 8.5. b) anstoßen (rekursive Namennavigation), welche eine Server-Hierarchie durchläuft, die durch die Struktur des Namenraums vorgegeben ist, oder er kooperiert nach dem Interaktionsmuster „Team“ (Abb. 9.2) mit den anderen Name-Servern als Partnern (nichtrekursive Namennavigation). Das Resultat dieser Teamarbeit meldet der zuerst angesprochene Name-Server an den beauftragenden Client zurück.

In einem verteilten Dateisystem, was hier der Implementierung eines verteilten Namenraums entspricht, besteht ein gängiger Dienst in der Replikation von Dateien (hier Resource Records), wobei jede Kopie einer Datei auf einem anderen Rechner gespeichert wird. Man verfährt so, weil erstens dadurch die Verfügbarkeit einer Datei er-

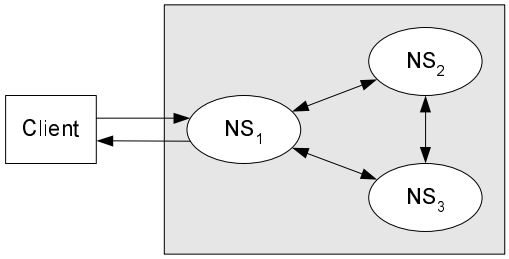

Abbildung 9.2: Namenresolution durch Kooperation zwischen drei Name-Servern (NS1-NS3)

höht wird, da sie selbst bei einem temporären oder totalen Ausfall eines Rechners noch vorhanden bleibt und deshalb nicht verloren geht. Dadurch wird das System insgesamt auch fehlertoleranter und zuverlässiger. Zweitens kann der Zugriff auf eine Datei beschleunigt werden, da durch die Aufteilung der Nutzlast auf mehrere Rechner derjenige Rechner eingesetzt wird, der am wenigsten belastet ist (Lastbalancierung).

Eine Effizienzsteigerung wird ferner durch die Technik der Datei-Pufferung erreicht. Ein Client hält in seinem lokalen Speicher (Cache-Speicher, Abschn. 7.6.2.1) die Namenresolutionen, welche er am meisten braucht, um möglichst schnell die gewünschte Adresse zu erhalten. Befindet sich eine angeforderte Namenauflösung nicht bereits im Cache, wird sie vom Namendienst geholt und in den Cache-Speicher eingefügt. Es liegt nahe, dass diese Technik auch bei Web-Seiten angewandt wird.

Replikation und Pufferung im Cache-Speicher sind Standardtechniken, die in verteilten Dateisystemen angewendet werden. Die Problematik der Konsistenzhaltung von gepufferten Dateien und der Aktualisierungsstrategie von Replikaten, die dabei entsteht, werden wir in Abschnitt 10.7.5 behandeln.

Der Dienst DNS (TCP/IP-Referenzmodell) ist vergleichbar mit dem, den weiße Telefonseiten liefern; vom Namen wird auf die Telefonnummer geschlossen. Die Dienstqualität, welche der globale und verteilte *Verzeichnisdienst* (engl. directory service) X.500 (OSI-Referenzmodell) bietet, ist weit aufwendiger als die eines üblichen Namendienstes wie DNS und kann mit dem Dienst, den die gelben Seiten liefern, verglichen werden. Aus der Angabe eines oder mehrerer Attribute wird das Objekt mit seinen sämtlichen Attributen gefunden und aufgezeigt. In Dateiverzeichnissen sind Objekte zusammen mit ihren relevanten Attributen abgelegt. Beispielsweise kann man fragen: „Welche Rechner im Institut sind Macintosh-Rechner und mit dem MacOS 10.1 Betriebssystem ausgestattet?“ Oder es kann in Analogie zu den gelben Seiten auf die Angabe der Telefonnummer 0711-7816-387 Folgendes zurückkommen: {”Name = Paul Levi”, ”Telephon Number = 0711-7816-387”, ”e-mail-address= levi@informatik.uni-stuttgart.de”, …}.

9.2 Zeitdienste

Wir können das Informationsmodell – und hier insbesondere das Interaktionsmodell – und das Kontrollmodell eines Operationsprinzips von verteilten Systemen ohne Kennt-

nis der Zeit in den einzelnen Rechnerknoten nicht umsetzen. Konkreter gesprochen: Wir brauchen Zeitangaben, wenn wir messen, regeln, kausale Abhängigkeiten zwischen Ereignissen (Aktionen) feststellen und Ereignisse, die in einzelnen Prozessen stattfinden, synchronisieren. Der Begriff der Synchronisation bedeutet für uns immer die zeitliche Abstimmung (Koordination) von Ereignissen. Dies ist beispielsweise für die Interaktion zwischen Prozessen in verteilten Systemen deutlich schwieriger als in zentralen Systemen. Die Synchronisation erfolgt üblicherweise durch Uhren. Es hat sich jedoch herausgestellt, dass die exakte Zeit physikalischer Uhren für die Synchronisation nicht immer notwendig ist, denn durch so genannte logische Uhren (logische Zeit), die nur kausalen Prinzipien gehorchen, lässt sich beispielsweise die Prozess-Synchronisation ebenfalls bewerkstelligen [Lamport 78], [Tanenbaum 95]. Wir wollen uns in diesem Abschnitt jedoch auf die Synchronisation durch physikalische Uhren, die mit Hilfe eines Zeitdienstes möglich ist, konzentrieren. Wenn wir daher im Folgenden von Uhren sprechen, dann meinen wir damit vor allem reale Uhren (Hardwareuhren).

9.2.1 Zeit in verteilten Systemen

In einem zentralen System gibt es eine globale Zeit, die für alle Komponenten dieses Systems identisch ist. In einem verteilten System ist dies nicht möglich, denn obwohl physikalisch eine einzige Zeit für die ganze Welt, z. B. durch UTC (astronomische Zeit), welche die atomare Zeit (International Atomic Time, IAT) an unsere terrestrischen Gegebenheiten anpasst (Einführen von Schaltsekunden, engl. leap seconds), existiert, gelingt es nicht, die Uhren auf sämtlichen Rechnern so untereinander abzustimmen, dass alle dieselbe Zeit anzeigen, weil es kein Synchronisationsverfahren gibt, das alle individuellen Abweichungen der verschiedenen verteilten Uhren so ausgleichen kann, dass sie alle mit einer perfekten Uhr übereinstimmen. Die individuellen Abweichungen entstehen durch die *Uhrendrifts* (engl. clock drifts), die von ihren unterschiedlichen Schwingungsverhalten der Quarzoszillatoren herrühren und sowohl von der Temperatur als auch dem Luftdruck abhängig sind. Eine Uhr wird dabei durch die Kombination eines Quarzoszillators mit einem diskreten Zähler definiert, der die Schwingungen seit seiner Initialisierung zählt. Anders ausgedrückt: Es gibt keine perfekte Synchronisation von Uhren in verteilten Systemen und daher auch keine Möglichkeit, in ihnen eine globale Zeit einzurichten. Aus diesem Grunde existiert auch kein exakter globaler Systemzustand in einem verteilten System, in dem beispielsweise festgehalten wird, welche Prozesse zu einem bestimmten Zeitpunkt aktiv und welche passiv sind, sondern nur ein angenäherter.

Zeit spielt in verteilten Systemen aus verschiedenen Gründen eine sehr wichtige Rolle. Erstens müssen wir sagen können, wann ein bestimmtes Ereignis auf einem Rechner stattgefunden hat, damit vor allem die Synchronisation von Ereignissen (Aktionen) auf verschiedenen Rechnern vorgenommen werden kann. Zweitens werden möglichst genaue Zeitangaben benötigt, um die Konsistenzhaltung verteilter Daten zu gewährleisten. Drittens kann ohne Zeitvorgaben die Unterscheidung von synchronen und asynchronen Netzwerken nicht vorgenommen werden.

Lassen Sie uns den ersten Grund detaillieren und nehmen wir als Beispiele eine Banküberweisung und eine synchrone bzw. isochrone Übertragung von Daten, dann ist es

für den empfangenden Prozess ganz wesentlich, dass die Zeitmarken auf den zu übertragenden Daten korrekt sind bzw. die beiden Uhren für die Übertragung gleich getaktet sind. Verallgemeinernd soll damit ausgedrückt werden, dass eine Synchronisation der Ereignisse, die von verschiedenen Prozessen durch ihre Aktionen, wie Sende- oder Empfangsoperationen, erzeugt werden, nur mit Hilfe der Zeit erfolgen kann, denn nach dem *Kausalitätsprinzip* gilt: Ein Ereignis e, welches ein anderes Ereignis e′ verursacht, muss zeitlich vor dem Ereignis e′ auftreten. Eine Nachricht kann erst empfangen werden, wenn sie zuvor abgeschickt wurde. Es ist nicht möglich, dass eine Nachricht eintrifft, bevor sie abgeschickt wurde. Diese Notwendigkeit, Ereignisse zeitlich zu ordnen, kann man sich in einem zweidimensionalen Ereignis-Zeit-Diagramm klar machen.

Wir gehen von einer Gruppe von 4 Prozessen P_1 bis P_4 aus, die untereinander E-Mails austauschen. Prozess P_1 sendet an alle drei anderen Prozesse eine Nachricht N_1 mit der Aufforderung zur gemeinsamen Besprechung. Prozess P_2 bestätigt diese Nachricht durch N_2 und schickt seinen „Replay“ auch an alle drei anderen Prozesse. Prozess P_3 verfährt mit der Rückantwort N_3 nach demselben Muster. Prozess P4 hat noch nicht geantwortet, hat aber alle Nachrichtenaustausche beobachtet. Jetzt stellt sich die Frage, in welcher zeitlichen Reihenfolge sieht er bei seiner Beobachtung die Nachrichten bei sich eintreffen. Hierbei hat jede abgesandte Nachricht eine Zeitmarke (engl. time stamp) t_1 bis t3. Wenn die Uhren auf den Rechnern der Prozesse P_1 bis P_3 richtig synchronisiert sind, also nach dem Kausalitätsprinzip $t_1 < t_2 < t_3$ gilt, dann sieht P_4 die Nachrichten in der korrekten Reihenfolge N_1, N_2 und N_3 (Abb. 9.3). Wegen der unabhängigen Verzögerungen bei den einzelnen Nachrichtenübermittlungen und bei fehlender Uhrensynchronisation kann Prozess P_4 die inkorrekte Reihenfolge N_3, N_1 und N_2 erhalten (Abb. 9.4).

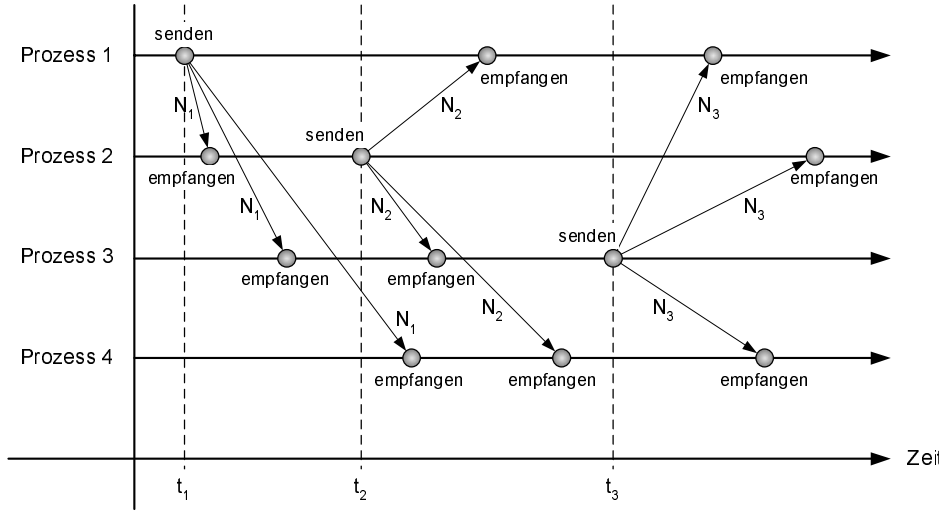

Abbildung 9.3: Ereignis-Zeit-Diagramm zur zeitlichen Einordnung von Ereignissen. Die Ereignisse sind die Send- und Receive-Operationen der Prozesse P_1 bis P_4. Nachricht N_1: Besprechung einberufen, Nachrichten N_2 und N_3: Besprechung bestätigen. Die Ereignisse erfolgen in der korrekten Reihenfolge

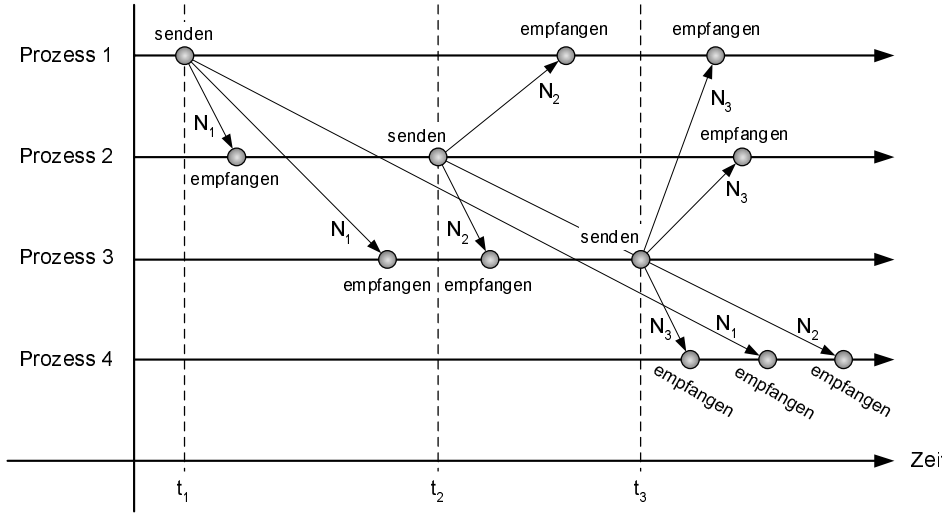

Abbildung 9.4: Ereignis-Zeit-Diagramm zur zeitlichen Einordnung von Ereignissen. Die Ereignisse erfolgen in einer inkorrekten Reihenfolge

9.2.2 NTP: Zeitdienst zur externen Uhrensynchronisation im Internet

Wir wissen bereits, dass es trotz des Vorhandenseins einer korrekten Zeit, die weltweit gilt (UTC), nicht möglich ist, alle Uhren eines verteilten Systems so zu synchronisieren, dass sie alle mit der gültigen, physikalischen Weltzeit übereinstimmen. Diese Aussage ist im absoluten Sinne zu werten. Viele weitere Anwendungen, wie die Überprüfung der Konsistenz verteilter Daten (z.B. durch Zeitmarken), die Überprüfung der Authentizität von Anfragen, die an einen Server gestellt werden, die Eliminierung von mehrfachen Aktualisierungen (engl. updates) von Dateien und die Koordination von kooperativen Fertigungsaufgaben kommen mit synchronisierten Uhren aus, die absolut gesehen nicht korrekt laufen, sondern innerhalb einer vorgegebenen Fehlerschranke, die durch die Driftrate D einer Uhr definiert wird, voneinander abweichen. Dabei gibt die *Driftrate* einer Uhr an, welche Zeitdifferenz zwischen ihr und einer nominal perfekt gehenden Uhr pro Zeiteinheit existiert. Gängige Werte für die Driftrate von Quarzuhren liegen bei $D = 10^{-6}$ /Sekunde bis 10^{-8} /Sekunde.

Eine Hardwareuhr H, wie wir eine reale Uhr im Gegensatz zu einer Softwareuhr C, deren Wert von einem Betriebssystem aus der Anzeige einer Hardwareuhr gebildet wird, zählt in einem digitalen Zähler (z.B. 128 Bit Zähler) die Anzahl der Schwingungen, die seit der Startzeit t_0 bis zum Zeitpunkt t erfolgt ist. Die Zeitanzeige der Hardwareuhr H zum exakten Zeitpunkt t sei H(t). Im fehlerfreien Fall gilt: $H(t) = H(t_0) + (t - t_0)$.Wir wissen aber bereits, dass H(t) abhängig ist von der Frequenz, welche die Zählrate R(t) definiert, und von der Driftrate D(t), welche die Änderungsrate der Frequenz (erste Ableitung der Frequenz) angibt. Mathematisch lassen sich solche Zusammenhänge da-

durch formulieren, dass für H(t) eine Taylorentwicklung bis zur zweiten Ordnung (mit Restfehler $R_3(t)$) angesetzt wird:

$$\begin{aligned} H(t) &= H(t_0) + H'(t_0)(t - t_0) + \frac{1}{2}H''(t_0)(t - t_0)^2 + R_3(t) \\ &= H(t_0) + R(t_0)(t - t_0) + \frac{1}{2}D(t_0)(t - t_0)^2 + R_3(t)\,. \end{aligned}$$

Hierbei wurden die zuvor genannten Ersetzungen vorgenommen: Zählrate und Driftrate $R(t_0) = H'(t_0)$, die pro Zeiteinheit gemessen wird. Die Zählrate ist dimensionslos und wird als Frequenzverhältnis zwischen der korrekten Frequenz und der tatsächlichen üblicherweise davon abweichenden Frequenz dargestellt. Im Idealfall ist $R(t_0) = 1$ und $D(t_0) = R'(t_0) = 0$ – dann ist die Uhr *genau* – und es gilt der zuvor genannte Zusammenhang, der für $t_0 = 0$ und $H(t_0) = 0$ besonders einfach wird: $H(t) = t$.

Die Uhrensynchronisation von NTP bezieht sich auf die Anzeige H(t) von Hardwareuhren und die damit erzeugten Zeitmarken. Bevor wir darauf eingehen, muss noch unterschieden werden, ob eine externe oder interne Synchronisation vorgenommen wird.

Bei einer *externen Synchronisation* geht man von einer externen Quelle Q(t) der UTC-Zeit aus und vergleicht jede individuelle Hardwareuhr Hi(t) mit Q(t), wenn bezüglich einer oberen Schranke $S > 0$, gilt: $|Q(t) - H_i(t)| < S$, für $i = 1, \ldots, n$. Alle Uhren H_i sind innerhalb der Schranke s genau. Die Uhren Hi sind *intern synchronisiert*, wenn sie untereinander ohne äußere Quelle Q(t) innerhalb der oberen Schranke s übereinstimmen. Formal gilt: $|H_i(t) - H_j(t)| < S$, für alle $i, j = 1, \ldots, n$.

Aus diesen beiden Definitionen folgt mit Hilfe der Dreiecksungleichung, dass ein extern synchronisiertes System innerhalb einer oberen Schranke von 2S intern synchronisiert ist. Die Umkehrung, dass intern synchronisierte Uhren auch extern synchronisiert sind, trifft nicht zu. Als obere Schranke wird für beide Synchronisationsarten in der Regel S = D gewählt.

Aus Sicht eines Programmierers werden die Zeitmarken t_i durch ein Programm des verwendeten Betriebssystems, das dafür eine Softwareuhr C(t) verwendet, erzeugt. Die beiden Uhren H(t) und C(t) hängen folgendermaßen zusammen: $C(t) = \alpha H(t) + \beta$, dabei ist α ein Skalierungsfaktor und β ein Verschiebungsfaktor, die beide dazu dienen, solche Zeitangaben zu erzeugen, die für Anwendungen geeigneter erscheinen als die direkte Angabe der physikalischen Uhr H(t). Programmtechnisch läuft die Aktualisierung von C(t) so ab, dass der lokale Zeit-Server von NTP, der in einem Host installiert ist, H(t) synchronisiert und den aktuellen Wert an das Betriebssystemprogramm übergibt, welches daraus die Anzeige C(t) generiert.

Ferner muss bei der Synchronisation zwischen synchronen und asynchronen Netzwerken unterschieden werden. Die gesamte Skala der möglichen Synchronisationsverfahren von Softwareuhren reicht daher von interner Synchronisation in synchronen Netzwerken bis externer Synchronisation in asynchronen Netzwerken. Wir wollen uns hier auf den zuletzt genannten Fall konzentrieren und uns als Beispiel das Network Time Protocol (NTP) vornehmen, das dem Internet-Zeitdienst zugrunde liegt, wobei allerdings die Server dieses Dienstes sich auch intern synchronisieren [Mills 91], [RFC 1305]. Bei diesem Protokoll wird unter Synchronisation nicht nur die von der Zeit,

sondern auch die von Frequenzen berücksichtigt. Der letztere Aspekt hat sehr viel mit regelungstechnischen Details (z.B. Nachlaufsynchronisation, engl. Phase-Locked-Loop, PLL, [Tietze, Schenk 99]) zu tun und bleibt daher hier weitgehend unberücksichtigt.

Der NTP-Dienst wird durch ein Netzwerk von Zeit-Servern erbracht, die über das Internet verteilt sind. Einzelne Server werden in lokale Subnetze eingeordnet, die als *Synchronisation-Subnetze* bezeichnet werden. Diese Teilnetze werden in einer logischen Hierarchie als gerichtete Bäume (Def. 2.7.7) angeordnet, deren einzelne Ebenen als ein *Stratum* bezeichnet werden und durch die Weglänge zur Wurzel charakterisiert werden (Abb. 9.5 a). Der Wurzelknoten (Stratum 1) wird extern durch eine UTC-Quelle synchronisiert, Stratum 2 Server werden durch den Wurzel-Server synchronisiert, Stratum 3 Server werden durch Stratum 2 Server synchronisiert usw. Die Blätter dieses Baums bilden die Arbeitsrechner der Anwender. Die dicken gerichteten Kanten (Pfeile) zeigen Synchronisationspfade (Zeitfluss) an und die Zahlen in den Knoten geben das Stratum an. Die dünnen ungerichteten Kanten zeigen Informationswege an, über die ebenfalls Zeitangaben fließen, die aber nicht notwendigerweise für die Synchronisation, sondern für die Rekonfiguration (Selbstorganisation) des Netzes benötigt werden.

Die Stratum-Nummer ist einfach zu bilden, wenn die Server bereits in die Strata einsortiert sind. Bislang haben wir aber noch nicht berücksichtigt, wie ein Server eine solche Nummer erhält. Diese Zuweisung ist recht aufwändig, da z.B. die Auflösung (minimale Zeitabstände), die Stabilität (Frequenzstabilität), die Zuverlässigkeit (Verfügbarkeit) der Uhr eines Servers und die Zuverlässigkeit eines Übertragungskanals zwischen zwei Servern dafür herangezogen wird. Ferner gilt, dass die Uhren der Zeit-Server mit einer höheren Stratum-Nummer ungenauer gehen als die mit einer kleineren Nummer.

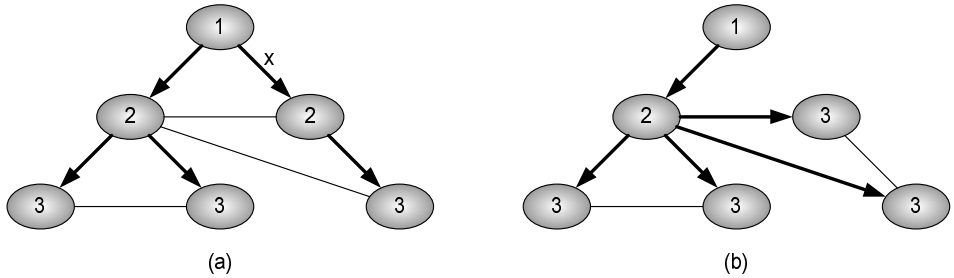

Abbildung 9.5: (a) Exemplarisches Synchronisations-Subnetz, dessen Knoten Server dieses Netzes darstellen. (b) Die Rekonfiguration des Subnetzes (der Strata), wodurch ein Server vom Stratum 2 zum Stratum 3 versetzt wurde

Abbildung 9.5a zeigt ein bereits aufgebautes Subnetz. Es wird aus einem Graphen gewonnen, dessen Knoten aus vielen redundanten Servern bestehen, bei denen auch mehrere Server in der obersten Ebene (Strata 1) vorhanden sein können. Das Organisationsprinzip, nach dem aus einem Graphen von Knotenrechnern ein baumartiges Subnetz gebildet wird, besteht darin, dass mit den aufsteigenden Stratum-Nummern

von der Wurzel aus ein minimaler *Spannbaum* aufgebaut wird. Dies ist ein Spannbaum (Def. 2.7.12), dessen Summe der Kosten sämtlicher Kanten minimal ist. Die Kosten einer Kante werden unter anderem festgelegt durch die Stratum-Nummer des Endknotens einer Kante, seinen Synchronisationsabstand (engl. synchronization distance) zur Wurzel, worunter die Dauer eines vollständigen Nachrichtenzyklus (engl. round-trip time) zu verstehen ist, der sich zusammensetzt aus dem Senden einer Nachricht zur Wurzel, und dem Empfang der Antwortnachricht durch den Absender und einer Filterdispersion ε, die ein Gütemaß für die Uhrenabweichungen (engl. Offsets) darstellt.

Ein Synchronisations-Subnetz rekonfiguriert sich selbst. Wenn z. B. der primäre Server (Wurzel) keine UTC-Daten mehr erhält, wird er zu einem Stratum 2 Server und innerhalb dieses Stratums synchronisieren sich die sekundären Server intern; wenn ein Kanal zwischen Stratum 1 und einem Knoten in Stratum 2 ausfällt, wird der sekundäre Server zu einem tertiären Server (Stratum 3). Abbildung 9.5b verdeutlicht den zuletzt genannten Fall der Selbstorganisation, bei dem die mit x gekennzeichnete Verbindung ausgefallen ist.

Wenn ein Synchronisations-Subnetz aufgebaut ist, arbeiten die Server dieses Netzes nach verschiedenen Interaktionsmustern zusammen, abhängig davon, in welchem Operationsmodus sie sind, welcher wiederum in Abhängigkeit von der geforderten Genauigkeit der Zeitangabe und der dafür verfügbaren Zeit ausgewählt wird. Wir wollen hier auf die beiden wichtigsten eingehen.

Im Operationsmodus *multicast mode* kommt das Client-Server-Interaktionsmuster zum Tragen. Ein oder mehrere Zeit-Server mit höherer Stratum-Nummer benachrichtigen periodisch andere Zeit-Server, die sich in einem niedrigeren Stratum befinden, mit aktuellen Zeitmarken. Diese aktualisieren dann ihre Uhren auf der Basis von geschätzten Verzögerungen (engl. delays) von einigen Millisekunden, die durch die Übertragung zustande kommen. Dieser Modus ist für schnelle lokale Netze (LANs) gedacht, bei denen es nicht auf eine hohe Zeitgenauigkeit ankommt.

Im Operationsmodus *symmetric mode* werden alle Server als Partner betrachtet, die nach dem Interaktionsmuster „Team“ kooperieren. Diese Betriebsart liefert die höchste Genauigkeit, da die Server alle gegenseitig periodisch immer Paare von Nachrichten (Nachrichtenzyklus) mit Zeitmarken austauschen, um lokal ihre Uhren neu einzustellen. Jede Nachricht enthält dabei drei Zeitmarken: Absendezeitpunkt t_{i-3} und Empfangszeitpunkt t_{i-2} der zuletzt zwischen einem Paar übertragenen Nachricht und die Zeitangabe t_{i-1}, wann die aktuelle Nachricht weggeschickt wurde. Der Empfänger der letzten Nachricht notiert sich den Empfangszeitpunkt t_i. Alle vier relevanten Zeitmarken werden in Abbildung 9.6 beim Austausch der Nachrichten N_{j-1} und N_j zwischen den beiden Partner-Servern A und B angezeigt. Beim Abschicken von N_{j-1} werden neben t_{i-3} auch die beiden Zeitmarken t_{i-5} (Sendezeitpunkt von B) und t_{i-4} (Empfangszeitpunkt durch A) in N_j, die sich auf die Nachricht N_{j-2} beziehen, eingetragen.

Server A schätzt mit diesen vier Zeitmarken den Zeitversatz (engl. offset) o_i der Uhr von B bezüglich A zum Zeitpunkt t_i und berechnet die Verzögerung (engl. delay) d_i des Nachrichtenzyklus, welche durch die totale Übertragungszeit der beiden Nachrichten zustande kommt. Diese beiden Parameter lassen sich auf folgende Weise berechnen: Seien o der wahre Zeitversatz, t die Übertragungsdauer von m und t die Übertragungs-

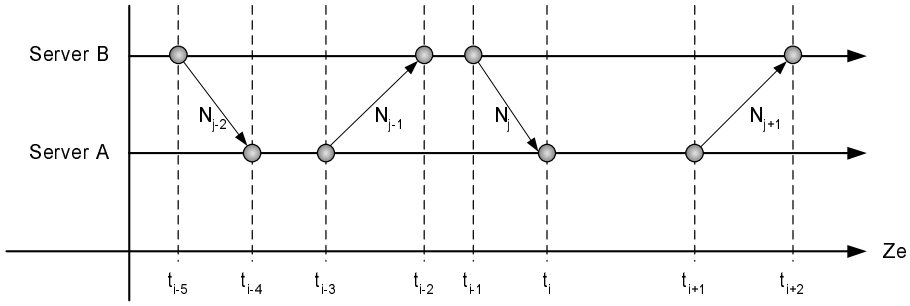

Abbildung 9.6: Aufbau eines Nachrichtenzyklus zwischen zwei NTP-Partnern A und B, der durch den gegenseitigen Austausch der Nachrichten N_{j-1} und N_j entsteht. Dadurch kennt A die vier Zeitmarken t_{i-3}, t_{i-2}, t_{i-1} und t_i

dauer von m′, dann gilt für die beiden Zeitmarken t_{i-2} und t_i:

$$t_{i-2} = t_{i-3} + t + o \text{ und } t_i = t_{i-1} + t' - o.$$

Dies führt für die Verzögerung d_i und den Offset o zu folgenden Resultaten:

$$d_i = t + t' = t_{i-2} - t_{i-3} + t_i - t_{i-1} = a - b, \text{ mit } a = t_{i-2} - t_{i-3}, b = t_{i-1} - t_i \text{ und}$$
$$o = (a + b)/2 + (t' - t)/2 = o_i + (t' - t)/2, \text{ mit } o_i = (a + b)/2.$$

Hierbei ist o_i der berechnete und o der wahre Offset zum Zeitpunkt t_i. Den wahren Offset kennen wir nicht, aber wir können folgendermaßen abschätzen: Es gilt $t + o = a$, und da wir annehmen, dass $t \geq 0$ ist, folgt $t = a - o \geq 0$, was nach sich zieht, dass $o \leq a$ ist. Ferner gilt $t' = -b + o$, woraus mit dem gleichen Argument $b \leq o$ folgt, sodass insgesamt für o die Ungleichung $b \leq o \leq a$ gilt. Diese Ungleichung lässt sich äquivalent ausdrücken durch:

$$o_i - d_i/2 \leq o \leq o_i + d_i/2.$$

Somit liefert o_i eine Abschätzung des Uhren-Offsets o und d_i liefert ein Genauigkeitsmaß für diese Abschätzung.

Umgekehrt kann B das Paar (o_{i+2}, d_{i+2}) zum Zeitpunkt t_{i+2} bestimmen, wenn er die nächste Nachricht nach N_j (N_{j+1}) erhält, wodurch er neben t_{i-1} und t_i auch t_{i+1} und t_{i+2} kennt. So kann jeder Server eines Synchronisation-Subnetzes seine Differenz im Uhrengang (Offset) und Verzögerung durch Nachrichtenübertragung bezüglich jedes einzelnen anderen Partners bestimmen.

Der paarweise Abgleich der Uhren geschieht nicht durch einen einzigen Nachrichtenzyklus, sondern durch eine Folge von Paaren (o_i, d_i), (o_{i-1}, d_{i-1}), … , (o_{i-n+1}, d_{i-n+1}), die im statistischen Sinn als Stichprobe betrachtet werden, um daraus die beste Schätzung $(\hat{o}, \hat{d})$ zu bestimmen. Die Güte der hieraus resultierenden Abschätzung für o wird bestimmt durch:

$$\hat{o} - \hat{d}/2 - \varepsilon \leq o \leq \hat{o} + \hat{d}/2 + \varepsilon,$$

wobei ε als Filter-Dispersion bezeichnet wird, die zusätzlich den Einfluss eines maximalen Fehlers beschreibt. Je größer die Filter-Dispersion ist, umso unzuverlässiger sind die Daten. Im statistischen Sinn kann das Intervall $[\hat{o} - \hat{d}/2 - \varepsilon\,,\ \hat{o} + \hat{d}/2 + \varepsilon\,]$ als ein Konfidenzintervall für o betrachtet werden.

Als Folge dieser *Datenfilterung*, die jeder Server vornimmt und die nicht davon abhängig ist, mit welchem anderen Server die Nachrichtenzyklen aufgebaut wurden, da Rechner und Nachrichtenkanäle immer wieder ausfallen können, hat jeder Partner im Synchronisation-Subnetz eine quantitativ bewertete Schätzung für o. Im Anschluss daran wird von allen Partnern derjenige als Synchronisationsquelle ausgewählt, der die beste Schätzung und eine möglichst niedrige Stratum-Nummer hat (engl. *peer selection*). Der zuletzt genannte Faktor rührt davon her, dass die Zeitfehler durch jede Synchronisationsebene zunehmen.

Zum Schluss werden, ausgehend von der Frequenz des ausgewählten Zeit-Servers, die lokalen Frequenzen der Uhren der restlichen Zeit-Server in einer Regelschleife (Nachlaufsynchronisation) mit den Datenfiltern aktualisiert.

Als Schutz gegen Eindringlinge wird eine lückenlose Authentifizierungskette aufgebaut, die von den Blättern eines Subnetzes bis zur Wurzel reicht. Zusätzlich wird nach dem symmetrischen DES-Algorithmus (Abschn. 9.3.3.1) die Prüfsumme verschlüsselt.

Die Clients sind alle Programme in Anwenderrechnern, die bei ihrem lokalen Zeit-Server (Blattknoten) diesen Dienst anfordern. Ein Client ruft NTP über einen Server des nächstgelegenen Synchronisations-Subnetzes auf, um seine lokale Uhr möglichst genau mit der UTC zu synchronisieren. Dieses Subnetz läuft ständig in einem der erlaubten Modi, um immer die aktuellste Zeitangabe, in Abhängigkeit des Subnetz-Zustandes, zu generieren, wovon der Client allerdings nichts merkt.

Mit dem NTP können Uhren von Clients im Internet bis auf etwa 10 Millisekunden genau mit dem UTC-Wert synchronisiert werden, in einem Intranet (lokalen Netz) können die Uhren sogar bis auf 1 Millisekunde genau abgeglichen werden.

9.3 Sicherheitsdienste

Der Begriff der Sicherheit hat im Deutschen zwei Bedeutungen. Er beinhaltet zum einen den Schutz vor fremden, unerlaubten Eingriffen (engl. security), gegenüber Einrichtungen oder Menschen (z.B. Objekt- oder Personenschutz) oder in unserem Zusammenhang z.B. gegen das unerlaubte Lesen von E-Mails. Zum anderen meint er auch den Schutz gegen das Fehlverhalten eines Systems z.B. gegenüber Nutzern im Sinne der Betriebssicherheit (engl. safety). Um diese Mehrdeutigkeit aufzulösen, wird im deutschsprachigen Raum häufig der englische Begriff zusätzlich erwähnt. Wir beschäftigen uns in diesem Abschnitt ausschließlich mit dem Sicherheitsbegriff im zuerst genannten Sinne (security).

Das ständige Wachstum von Internet und WWW führt zu einer deutlichen Zunahme ihrer Dienste, wobei vertrauliche und private Daten verwaltet und übertragen werden, und erzeugen somit auch eine wachsende Abhängigkeit von diesen Diensten. Beispiele

für diese Dienste, die auf Internet- und WWW-Diensten aufbauen, sind finanzielle Transaktionen, elektronischer Wareneinkauf, Prüfungsnoten in einer Hochschule und Patientendaten. Mit zunehmendem Verbreitungsgrad dieser Dienste steigt gleichzeitig die Anzahl böswilliger und schadenfroher Attacken gegen verteilte Systeme. Damit in diesen die Daten gesichert sind und Angriffe gegen sie abgewehrt werden können, müssen in ihnen auch Sicherheitsmaßnahmen integriert werden. Durch sie werden grundlegende Forderungen der Datensicherung wie Geheimhaltung, Integrität und Authentizität erfüllt. Daher haben wir in das Operationsprinzip ein Sicherheitsmodell mit aufgenommen (Abschn. 8.4.1), welches die gefährdeten Objekte, wie Daten, Prozesse und Kanäle identifiziert und geeignete Sicherheitsstrategien (z.B. wer muss sich wem gegenüber authentifizieren, wer darf welche Klasse von Dokumenten lesen) und Sicherheitsmechanismen (z.B. Objektzugriff über Kontrolllisten, eingeschränkte Dokumentenverteilung) zur Abwehr gegen unautorisierte Zugriffe beschreibt. Sicherheitsdienste eines verteilten Systems setzen diese Strategien und Mechanismen algorithmisch um. Sie sind typischerweise zuständig für die Authentifizierung von Nutzern, die Integrität von Nachrichten und den Schutz von Ressourcen (Rechner, Kanäle, Speicher, Prozesse etc.), um ihre ständige Verfügbarkeit zu gewährleisten.

Der Bedarf an all diesen Diensten lässt sich aus den Kriterien ableiten, die Sicherungsverfahren erfüllen müssen. Man kann die Forderung nach einzelnen Diensten weiter verstärken, wenn die Sicherheitslücken bisheriger verteilter Systeme aufgedeckt werden. Von den unterschiedlichen Ansätzen, Sicherheitsdienste zu implementieren, wollen wir uns auf die wichtigste Klasse von Sicherheitsalgorithmen konzentrieren. Dies sind kryptografische Algorithmen, die verwendet werden, um Daten vor Attacken zu schützen. Wir werden hier die Dienste für die Verschlüsselungen von Daten, die Erzeugung von digitalen Signaturen mit öffentlichen Schlüsseln und die Zertifizierung (Beglaubigung) von öffentlichen Schlüsseln und deren Eigentümern behandeln.

9.3.1 Sicherheitskriterien, Bedrohungen und Attacken

Beim Umgang mit persönlichen, vertraulichen, geheimen, brisanten und wichtigen Daten sollte immer deren Sicherheit gewährleistet sein. Sicherheit ist vorhanden, wenn eine Reihe spezieller Kriterien erfüllt ist. Sicherheitsdienste sind all jene Dienste, die dazu dienen, diese Kriterien zu erfüllen. Das Ausmaß der Sicherheit hängt dabei von der Anzahl der erfüllten Kriterien ab. Im Einzelnen sind die folgenden 5 Sicherheitskriterien zu nennen, die ein volles Maß an Sicherheit bieten, sofern sie alle erfüllt sind:

1. *Vertraulichkeit* (engl. privacy, secrecy)
   Kein unberufener (nicht autorisierter) Dritter darf die Daten lesen. Nur der autorisierte Empfänger darf auf die Daten zugreifen.
2. *Integrität* (engl. integrity)
   Daten müssen geschützt sein gegen unberechtigtes Schreiben, worunter jede Art der Modifikation, wie Einfügen, Löschen, Verändern der Daten zu verstehen ist. So darf bei einem Bankgeschäft nicht einfach das Komma zu seinen Gunsten verschoben werden.

3. *Verfügbarkeit* (engl. availability)
   Schutz von Systemressourcen vor unberechtigten Zugriffen, um sie für autorisierte Zugriffe verfügbar zu halten. So sollte beispielsweise verhindert werden, dass durch einen ständigen unautorisierten Aufruf eines Dienstes dieser für berechtigte Anfragen gesperrt bleibt.
4. *Verbindlichkeit* (engl. liability)
   Ein Abstreiten von verbindlich zugesagten Aktionen (Verträgen) oder, allgemeiner, das Ableugnen der Urheberschaft bestimmter Vorgänge ist nicht möglich. Ein Empfänger kann nicht erfolgreich abstreiten, dass er eine bestimmte Nachricht nicht erhalten hat.
5. *Anonymität* (engl. namelessness)
   Es muss möglich sein, zu verbergen, wer mit wem wie oft kommuniziert oder wie oft jemand auf eine bestimmte Datei zugreift. Dieses Kriterium steht im Widerspruch zur Authentizität. Es sollte jedoch nicht möglich sein, dass Firmen Benutzerprofile erstellen, um dann den Benutzer mit auf ihn zugeschnittener Werbung zu überhäufen.

Digitale Signaturen bestätigen nicht nur wie herkömmliche Unterschriften die Identität eines Benutzers, sondern auch die Integrität und Verbindlichkeit von Daten, wobei die Verbindlichkeit durch zwei Punkte geschaffen wird: Erstens kann ein Empfänger die Identität des Absenders verifizieren und zweitens kann der Absender nicht abstreiten, dass er diese bestimmte Nachricht abgeschickt hat. Durch eine digitale Signatur kann so beispielsweise ein Dokument (Überweisung, Versicherungspolice) als ein echtes, verbindliches Dokument ausgewiesen werden. Zusätzlich kann eine Zertifizierungstelle eingeschaltet werden, um z. B. die Echtheit des verwendeten öffentlichen Schlüssels (Zuordnung Schlüssel zu Person) oder die Rechtmäßigkeit eines Bankkontos durch ein digitales Zertifikat zu bestätigen. Diese Bestätigung ist notwendig, um ganz sicher zu sein, ob etwa der öffentliche Schlüssel, der zum Dechiffrieren der digitalen Signatur eingesetzt wird, auch tatsächlich zu dem Absender gehört, der er vorgibt zu sein. Ein digitales Zertifikat entspricht einem Ausweis, der von einer autorisierten und vertrauenswürdigen Behörde ausgestellt wurde.

Die möglichen Sicherheitsbedrohungen, welche die Gültigkeit dieser Kriterien in Frage stellen können, werden in drei große Klassen eingeteilt:

Informationsverlust: Beschaffung von Information durch nicht autorisierte Abnehmer.

Informationsfälschung: unberechtigte Veränderung von Information.

Vandalismus: Zerstörung der korrekten Operationen eines Systems ohne Vorteilnahme des Angreifers (z. B. Hackers).

Die fünf oben genannten Sicherheitsbedingungen und die Gefährdungsklassen, welche diese bedrohen, gehen alle von der Sichtweise der Anwender aus. Details der internen Mechanismen in verteilten Systemen zur Gewährleistung dieser Forderungen bleiben hier unberücksichtigt, müssen aber gleichwohl von den Sicherheitsdiensten beachtet werden. Diese Details werden deutlicher, wenn wir die möglichen Angriffspunkte und die damit verknüpften Attacken kurz vorstellen.

In dieser Vorstellung gehen wir davon aus, dass ein *Eindringling* (engl. intruder) vorhanden ist und entweder am Endsystem (z.B. Prozesse von verteilten Anwendungen), am (logischen) Kommunikationskanal, an netzinternen Kommunikationsknoten wie Netzwerkanschluss, Router, Bridges, Switches, Repeaters, oder letztlich an dem Übertragungsmedium selbst angreift.

Mögliche Attacken auf Übertragungskanäle und interne Vermittlungsknoten sind:

- Änderung (Generieren, Löschen, Verzögern, Reihenfolge ändern) des Nachrichtenstroms auf einem Übertragungskanal (engl. message tampering).
- Maskerade (engl. masquerading) eines Eindringlings als autorisierter Benutzer oder Programm, um einen unbemerkten Einstieg für weitere Programme vorzubereiten (trojanisches Pferd).
- Lesen von Nachrichteninhalten durch Abhören des Nachrichtenstroms.
- Unberechtigtes Kopieren von Nachrichten als eine Form eines Lauschangriffs (engl. eavesdropping).
- Überlastung von Diensten (engl. denial of services) durch „Fluten“ der Übertragungswege und Rechner durch pausenloses Erzeugen eines unerlaubten Nachrichtenstroms.
- Analyse des Nachrichtenverkehrs, um Rückschlüsse auf Nachrichteninhalte zu gewinnen.

Übliche Attacken auf Hosts und Vermittlungsrechner sind:

- Einloggen eines nicht autorisierten Nutzers, um unerlaubte Aufrufe von Prozessen auf Clients oder Servern zu starten oder diese Prozesse beispielsweise durch Viren (Aktivierung auf eigenem Rechner bei Aufruf) oder Würmer (Internet-Kettenaufrufe von Programmen auf anderen Rechnern) zu blockieren.
- Unerlaubte Zugriffe auf lokale oder entfernte Daten.
- Abblocken von Diensten durch pausenlose Dienstaufrufe.

9.3.2 Sicherheitslücken und Sicherheitsmechanismen

9.3.2.1 Sicherheitslücken

Internetprotokolle übertragen Nachrichten unverschlüsselt im Klartext. Datenpakete durchlaufen häufig viele Vermittlungsrechner verschiedener Organisationen in mehreren Ländern. Sie sind dabei in jedem Teilnetz potenziell den zuvor geschilderten Bedrohungen und Attacken ausgesetzt. Wir wollen daher einige Sicherheitslücken aufzeigen, die bei nicht durch Sicherheitsdienste geschützten Nachrichten auftreten [Fuhrberg, Häger, Wolf 01], [Schiffer, Templ 99]. Beginnen wir mit ausgesuchten Internetdiensten.

Der E-Mail-Dienst mit SMTP bietet keine Sicherheit für die Authentizität des Absenders einer Nachricht. Der Absender einer E-Mail kann unbemerkt gefälscht werden. Der Dienst kann durch Verfügbarkeitsattacken behindert oder gar zum Zusammenbruch gebracht werden, wenn die E-Mail-Server mit Aufrufen überschwemmt werden. Der Dateitransfer-Dienst mit FTP ermöglicht unerlaubte Zugriffe auf Dateien, die sich auf einem FTP-Server befinden, wenn die Dateiverwaltung nicht nur Lese-, sondern

auch Schreibrechte vorsieht und beispielsweise Passwortdateien nicht genügend schützt. Der Zeitdienst (NTP) lässt sich auf diese Weise beeinflussen, sodass die Systemzeit verändert werden kann und Zeitmarken dadurch möglicherweise ungültig werden.

Der WWW-Dienst für die Übertragung von Web-Seiten mit HTTP bietet ebenfalls keine Garantie für eine ungefährdete Übermittlung vertraulicher Daten, da sie ebenfalls im Klartext versandt werden. Ferner können Browser, die automatisch jedes Programm ausführen, das in Web-Seiten eingebettet ist, sich als Sicherheitsrisiko erweisen. So kann es möglich sein, dass „arglistiger" Code heruntergeladen wird, der sich als trojanisches Pferd erweist und Dateien löscht oder private Daten liest. Dieses Beispiel weist deutlich auf die Gefahr hin, die im mobilen Code steckt. Hier liefert die Programmiersprache Java das bekannteste Beispiel für den Einsatz mobilen Codes. Als einfache, doch ungenügende Sicherungsmaßnahme, die von Java ergriffen wird, ist die Separierung lokaler Dateien von heruntergeladenen Dateien. Dies wird mit dem Verbot verknüpft, dass von den heruntergeladenen Dateien kein Zugriff auf lokale Dateien und andere Ressourcen erlaubt ist (Abschn. 9.4.1.2).

Beim mobilen Rechnen kann es vorkommen, dass die verschiedenen Stationen eines Benutzers verfolgt und registriert werden, was das Vertraulichkeitskriterium verletzt. Des Weiteren kann die Organisation, die den entfernten Zugriff eines portablen Rechners auf das „heimische" Netz des Benutzers (z. B. Intranet) erlaubt, dessen Verbindung zu seinem lokalen Netz dazu missbrauchen, vertrauliche Daten hinter deren lokalen Firewalls bloßzulegen oder gar ein trojanisches Pferd einzuschmuggeln, welches dieses entfernte Netz für Angriffe von außen unbemerkt öffnet.

Wenn lokale Funknetze eingesetzt werden, können Unbefugte außerhalb eines Gebäudes auf den Übertragungsweg zwischen Rechner und Basisstation (vgl. Abb. 8.9) des Funknetzes zugreifen und die Daten, die von einzelnen Rechnern verschickt werden, abhören oder gar für eigene Zwecke manipulieren. Einfache, aber nicht ausreichende Schutzmaßnahmen, die neben einer Verschlüsselung durchgeführt werden können, bestehen darin, dass jeder Client einen eigenen Funk-LAN-Namen erhält, unter dem er sich bei der Basisstation anmelden muss und zusätzlich jede Ethernet-Adresse – bzw. MAC-Adresse, das ist die eindeutige Hardwareadresse der MAC-Schicht (Tab. 8.2) – lokal registriert wird und nur dann, wenn diese zugewiesene Adresse auch verwendet wird, ist eine Anmeldung bei der Basisstation möglich.

Sicherheitsprobleme treten auch bei Diensten auf, deren Protokolle z. B. auf der Basis von Prozedur-Fernaufrufen operieren, die typischerweise von einer Middleware angeboten werden. Beispiele hierfür liefern die verteilten Dateisysteme NFS (Network File System) von Sun und AFS (Andrew File System), welches an der CMU (Carnegie Mellon University) entwickelt wurde. Typische Sicherheitslücken kann es hier geben, wenn die Zustandsinformation über Dateien, welche von einem Client auf einem Datei-Server geöffnet sind, von nicht autorisierten Dritten gelesen oder modifiziert werden und die Subjekte (Benutzer, Prozess) ihre Identität gegenüber bestimmten Objekten (entfernte Dateien) nicht verifizieren (Einweg-Authentifizierung).

Nicht geschützte X-Windows-Server, die von Unix-Systemen die uneingeschränkte Kontrolle von Tastatur und Bildschirm auf lokalen Rechnern haben, können einem Ein-

dringling, der irgendwo vom Internet aus auf diesen Rechner zugreift, jede Tastatur- und Bildschirmeingabe melden, ohne dass der Betroffene merkt, dass er belauscht wird.

Insgesamt sollte mit dieser kleinen Aufzählung von Sicherheitsmängeln darauf hingewiesen werden, dass den Sicherheitsaspekten in verteilten Systemen noch immer nicht die Beachtung geschenkt wird, die eigentlich angebracht wäre.

9.3.2.2 Sicherheitsmechanismen

Bei der Konzeption von Sicherheitsmechanismen geht man von einer Grundstruktur aus, die für alle Sicherungsprobleme gültig ist:

1. Zerlege die Welt in *Objekte* und *Subjekte*. Objekte sind die passiven Elemente Rechner, Programme und Dateien. Subjekte sind Benutzer, Prozesse und alle aktiven Module, die auf Objekte zugreifen.
2. Installiere durch geeignete Mechanismen *Zugriffskontrollen*, die für jedes individuelle Objekt nur Zugriffe von autorisierten Subjekten erlauben. Zugriffskontrollen werden oft durch Zugriffskontroll-Listen oder Befähigungslisten (engl. capability lists) installiert.
3. Stelle durch spezielle *Verfahren* sicher, dass die Sicherheitsmechanismen nicht umgangen werden. Die Basisverfahren, welche die Umgehung von Zugriffskontrollen verhindern, sind: Authentifizierung, Zertifizierung (Beglaubigung) von öffentlichen Schlüsseln und deren Eigentümern, Aufbau von Brandmauern (z.B. mit HTTP über SSLP [Secure Socket Layer Protocol], HTTPS).

Die Authentizität von Subjekten und Objekten ist eine wesentliche Basis dafür, dass die Sicherheitskriterien Integrität und Verbindlichkeit hergestellt werden. Authentifizierung ist das Verfahren, um die Identität der beteiligten Subjekte und Objekte zu beweisen, d.h. die beteiligten Kommunikationsinstanzen zu verifizieren. Im Detail unterscheiden sich die Authentifizierungsverfahren für Nachrichten und Benutzer. Nachrichten werden vor allem durch digitale Signaturen zu authentischen und verbindlichen Dokumenten. Bei der Authentifizierung von Personen gegenüber Computern (Ein-Weg-Authentifizierung) werden typischerweise Passwörter oder Chipkarten, die kryptografische Verfahren enthalten, um z.B. mit Hilfe eines geheimen Schlüssels die Identifikation eines Benutzers zu verifizieren, eingesetzt. Hierdurch lässt sich feststellen, ob die angegebene Quelle der Daten der tatsächlichen entspricht (Identität der Quelle). Beim Einsatz von Chipkarten erfolgt die umgekehrte Authentifizierung des Rechners zum Menschen nach den gleichen Verfahren. Bei einer wechselseitigen Authentifizierung zweier Kommunikationspartner (Zwei-Weg-Authentifizierung) z.B. mit asymmetrischen Verschlüsselungsalgorithmen werden dafür öffentliche und private (geheime) Schlüssel verwendet. Im alltäglichen Leben authentifiziert der Ausweis die einzelne Person, und die Zwei-Weg-Authentifizierung entspricht der gegenseitigen Überprüfung der Ausweise. Bei einer Grenzkontrolle wird eine Ein-Weg-Authentifizierung des Reisenden durch einen Zöllner vorgenommen.

Die wichtigsten Verfahren, um die Sicherheitskriterien Vertraulichkeit, Integrität und Verbindlichkeit zu gewährleisten und auch den zusätzlichen Dienst der Zertifizierungen zu garantieren, sind kryptografische Algorithmen. Diese werden im nächsten Abschnitt besprochen.

9.3.3 Kryptografische Algorithmen

Kryptografie ist die Wissenschaft der Verschlüsselungstechniken [Beutelspacher 93], [Bauer 94]. Ein Kunstwort, das sich aus den zwei griechischen Wörtern für geheim (griech. κρυπτος) und schreiben (griech. γραφειν) zusammensetzt. Ausgangspunkt ist der Klartext (engl. plaintext), der verschlüsselt (chiffriert) wird, um daraus einen Chiffretext (engl. ciphertext) zu erzeugen, der dann übertragen und vom Empfänger entschlüsselt (dechiffriert) wird, um ihn zu lesen.

Ein Chiffrierverfahren, das seit Jahrtausenden verwendet wird, ist die Erzeugung einer *Substitutionschiffre*. Ausgangspunkt ist ein Klartextalphabet (Def. 2.1.7), das durch eine Chiffrierfunktion auf ein Chiffrealphabet abgebildet wird, wodurch einzelne Zeichen des Eingabealphabets durch Zeichen des Ausgabealphabets, das auch identisch mit dem Eingabealphabet sein kann, ersetzt werden [Fumy, Kessler 99]. So hat bereits der alttestamentarische Prophet Jeremias um ca. 600 v. Chr. in Kap. 25, Vers 26 eine Substitutionschiffre verwendet, um „Babylon“ zu verschlüsseln. Der Chiffrieralgorithmus bildet dabei wie folgt ab: Ein Buchstabe, der von vorne gezählt an n. Stelle steht, wird ersetzt durch einen Buchstaben, der rückwärts gezählt an n. Stelle steht. Die Wirkungsweise einer solchen Verschlüsselung kann durch eine Tabelle definiert werden (Tab. 9.1).

Tabelle 9.1: Eine Substitutionschiffre für das lateinische Alphabet. In der oberen Zeile steht das Klartextalphabet, in der unteren das Chiffretextalphabet

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
z	*y*	*x*	*w*	*v*	*u*	*t*	*s*	*r*	*q*	*p*	*o*	*n*	*m*	*l*	*k*	*j*	*i*	*h*	*g*	*f*	*e*	*d*	*c*	*b*	*a*

Hiermit kann „Babylon“ durch „Yzybolm“ und „Informatik“ durch „Rmulinzgrp“ verschlüsselt werden. Diese Chiffrierung gilt allerdings nur in unserem lateinischen Alphabet. An der besagten Bibelstelle steht daher nicht „Yzybolm“, sondern „Scheschach“ statt „Babel“, da das althebräische Alphabet 22 Konsonanten verwendet und Babylon als Babel bezeichnet. Es entspricht „Sch“ (hebr. Schin) dem 2. und „Ch“ (hebr. Kaph) dem 12. Buchstaben von hinten, und von vorne gezählt ist „B“ (hebr. Bet) der 2. und „L“ (hebr. Lamed) der 12. Buchstabe. Ferner ist zu beachten, dass die Vokale a und e nicht durch eigene Buchstaben separat geschrieben werden, sondern ursprünglich nur gesprochen wurden.

Cäsar benutzte ebenfalls eine Substitutionschiffre, bei der die Buchstaben lediglich um drei Stellen nach hinten verschoben wurden, aus einem a wurde ein d, aus einem b ein e, usw. In dieser sehr einfachen Verschlüsselung würde die Substitutionschiffre für „Informatik“ lauten: „Lqirupdwln“. Die Entschlüsselung kann durch dieselbe Tabelle erfolgen; es muss nur von der unteren Zeile in die obere übersetzt werden.

Bei einer Transpositionschiffre werden nicht die Zeichen ausgetauscht, wie bei der Substitutionschiffre, sondern die Zeichen des Klartextes bleiben und nur ihre Position wird verändert, d.h. die Zeichen werden permutiert. Die älteste uns bekannte Transpositionschiffrierung fand ca. 500 v. Chr. in Sparta zur militärischen Nachrichtenübermittlung statt. Verwendet wurde eine Skytala, ein Stab von einem vorgegebenen Durchmes-

ser. Um diesen wurde ein schmaler Pergamentstreifen spiralförmig, wie ein Verband, gewickelt und längs der Skytala lückenlos beschrieben. Der Empfänger einer Nachricht wickelte den Chiffriertext um einen Stab desselben Durchmessers und konnte dann die Nachricht im Klartext lesen.

Eine moderne Version dieses Verfahrens besteht darin, den Klartext in eine Matrix zeilenweise einzugeben (Leerzeichen und Satzzeichen werden dabei unterdrückt) und spaltenweise zu übertragen, wobei auch noch die Spaltenreihenfolge permutiert werden kann. Ein Beispiel liefert folgende Matrix, wobei * ein „Füllzeichen“ ist:

$$\begin{pmatrix} D & i & e & S & i & c \\ h & e & r & h & e & i \\ t & v & o & n & N & a \\ c & h & r & i & c & h \\ t & e & n & i & s & t \\ d & a & s & Z & i & e \\ l & d & e & r & K & r \\ y & p & t & o & g & r \\ a & f & i & e & * & * \end{pmatrix}$$

Klartext:
Die Sicherheit von Nachrichten ist das Ziel der Kryptografie.

Chiffretext:
ievheadpfshniizroedhtctdlyaerornseticiahterr*iencsikg*

Schlüssel:
(52 Buchstaben, 6 Spalten, Spaltenreihenfolge: 2, 4, 1, 3, 6, 5)

Der Schlüssel zum Kodieren und Dekodieren wird gegeben durch die Anzahl der Buchstaben, der Spalten und der Spaltenpermutation. Der Empfänger benutzt denselben Schlüssel, er trägt den Klartext spaltenweise ein und liest ihn zeilenweise aus (Transposition von Spalten und Zeilen).

Chiffrieralgorithmen benutzen immer einen *kryptografischen Schlüssel*. Dies ist ein charakteristischer Parametersatz, der von Chiffrier- und Dechiffrierfunktionen gebraucht wird, um Daten zu ver- oder entschlüsseln. Kryptografische Schlüssel werden hier von uns immer nur kurz Schlüssel genannt, da keine Verwechslungsgefahr mit den Datensatzschlüsseln (Def. 3.6.2) besteht. So war die Tabelle 9.1 ein Schlüssel für die Substitutionschiffrierung, die Zahl 3 ein Schlüssel für die „Cäsar-Chiffre“ und der Durchmesser der Skytala der Schlüssel für die spartanische Transpositionschiffrierung. In allen drei aufgezeigten Fällen wurden dieselben Schlüssel sowohl zum Chiffrieren als auch zum Dechiffrieren eingesetzt. Solche kryptografischen Algorithmen nennt man daher symmetrische Verfahren.

Verschlüsselungsalgorithmen lassen sich auf jedes Alphabet anwenden. In der Informatik interessiert vor allem das binäre Alphabet, weshalb alle Arten von Texten Bitfolgen sind und auch die Schlüssel durch endliche oder unendliche Bitfolgen über dem binären Alphabet gebildet werden. Formal gilt folgende Definition:

Definition: Kryptografischer Algorithmus (9.3.1)
Ein *kryptografischer Algorithmus* (Verschlüsselungsalgorithmus) KA ist ein Quintupel KA = (**K**, **P**, **C**, E, D), das sich aus drei Mengen und zwei Funktionen zusammensetzt:

1. **K** ist der *Schlüsselraum*, der aus allen möglichen Schlüsseln aufgebaut wird.
2. **P** ist der *Klartextraum*, der die Menge aller möglichen Klartexte enthält.

3. **C** ist der *Chiffretextraum*, der die Menge aller möglichen Chiffretexte enthält.
4. E ist die Chiffrierfunktion E: $\mathbf{K} \times \mathbf{P} \rightarrow \mathbf{C}$, wobei der Chiffrierschlüssel $K_E \in \mathbf{K}$ verwendet wird.
5. D ist die *Dechiffrierfunktion* D: $\mathbf{K} \times \mathbf{C} \rightarrow \mathbf{P}$, wobei der Dechiffrierschlüssel $K_D \in \mathbf{K}$ verwendet wird.

Die folgenden beiden Bedingungen müssen durch KA erfüllt werden:

a) Für jeden Codierschlüssel $K_E \in \mathbf{K}$ existiert ein Decodierschlüssel $K_D \in \mathbf{K}$, sodass für alle Klartexte $P \in \mathbf{P}$ und Chiffretexte $C \in \mathbf{C}$ gilt: $D(K_D, E(K_E, P)) = D(K_D, C) = P$.
b) Ohne Kenntnis des Dechiffrierschlüssels K_D ist es praktisch nicht möglich, aus einem Chiffretext $C = E(K_E, P)$ den Dechiffrierschlüssel K_D oder den zugehörigen Klartext P zu bestimmen.

Abbildung 9.7 zeigt den Ablauf eines kryptografischen Algorithmus.

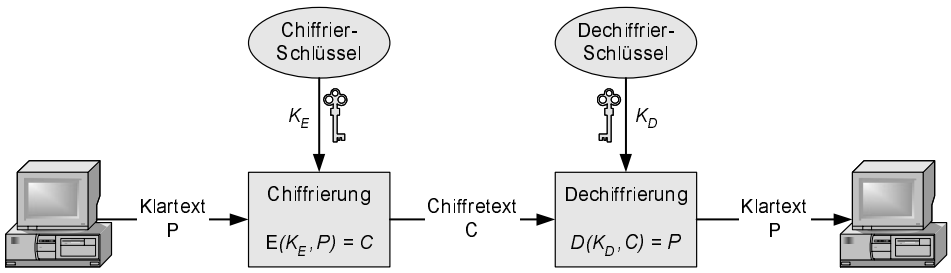

Abbildung 9.7: Austausch von Nachrichten durch einen kryptografischen Algorithmus

Kryptografische Algorithmen sind sehr gut geeignet, sowohl die Sicherheitskriterien 1, 2 und 4 als auch Verfahren zur Authentifizierung durch digitale Signaturen zu verwirklichen. Abhängig davon, welche Schlüssel verwendet werden, unterscheidet man symmetrische und asymmetrische kryptografische Algorithmen.

9.3.3.1 Symmetrische kryptografische Algorithmen

Symmetrische kryptologische Algorithmen sind dadurch gekennzeichnet, dass beide Kommunikationspartner, hier Alice und Bob genannt, ein gemeinsames Geheimnis teilen. Sie benutzen dieselben oder leicht ineinander transformierbare Schlüssel zum Verschlüsseln (engl. encryption) und Entschlüsseln (engl. decryption), bei Schlüsselgleichheit gilt: $K_E = K_D = K$. Dadurch entsteht eine symmetrische Verschlüsselungsbeziehung zwischen Alice und Bob. Den gemeinsamen Schlüssel K müssen aber beide über einen sicheren Kanal austauschen. Ein *sicherer Kanal* entsteht durch die Verschlüsselung und die Authentifizierung von Sender (Alice) und Empfänger (Bob), welche die Vertraulichkeit und Integrität der über diesen Kanal übermittelten Daten garantiert. Ferner wird jede Nachricht mit einer Zeitmarke versehen, um deren Wiederholungen und Umorientierungen zu verhindern.

Besitzen beide Partner ihre geheimen Schlüssel, werden die verschlüsselten Nachrichten über einen regulären, d.h. nicht weiter gesicherten Kanal, der als unsicherer Kanal bezeichnet wird, übertragen. Ohne Authentifizierung sichern symmetrische Algorithmen nur die Vertraulichkeit von Daten.

Ein Beispiel für einen symmetrischen kryptografischen Algorithmus liefert Data Encryption Standard (DES). Dieser Algorithmus wurde vom NBS (National Bureau of Standards) 1977 in den USA eingeführt und wird beim elektronischen Handel und für das Internetbanking immer noch häufig verwendet, obwohl er nicht mehr ganz den heutigen Sicherheitskriterien genügt.

DES ist ein mehrstufiger Algorithmus, der Substitutions- und Transpositionschiffrierungen kombiniert. Er zerlegt den Klartext in binär codierte Blöcke (Blockchiffre) der Länge von 64 Bit und operiert mit 64-Bit-Schlüsseln, welche aber tatsächlich nur 56 Bit lang sind, da 8 Schlüsselbits als Paritätbits dienen. Damit enthält der Schlüsselraum $2^{56} = 7{,}20575 \cdot 10^{16}$ verschiedene Schlüssel. Das Verfahren selbst ist bekannt, nur die Schlüssel sind, wie es bei einem symmetrischen Verfahren sein muss, geheim.

Abbildung 9.8 verdeutlicht den prinzipiellen Ablauf von DES.

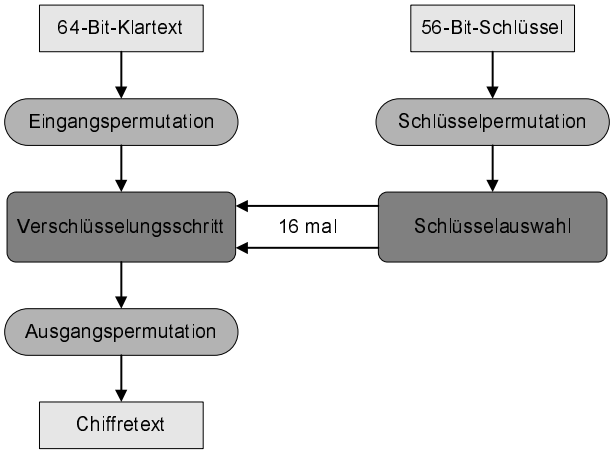

Abbildung 9.8: Skizze des Ablaufs von DES (Data Encryption Standard)

Ein 64-Bit-Block dient als Eingabe. Er wird permutiert und anschließend in 16 gleichartigen Verschlüsselungsschritten mit 16 verschiedenen Schlüsseln chiffriert. In einem Verschlüsselungsschritt wird dabei aus dem Schlüsselraum nach einem vorgegebenen Schema ein Schlüssel ausgewählt, dieser wird dann dem Klartext-Block durch eine XOR-Operation verknüpft und danach mit Hilfe von 8 Substitutionsboxen S_i (i = 1 – 8), die in Matrixform dargestellt werden, ersetzt. Beispielsweise sieht die Substitutionsbox S_1 wie in Tabelle 9.2 aus.

Die Substitution durch diese Box geschieht so, dass je 6 Bit eingegeben werden $b_1\ b_2\ b_3\ b_4\ b_5\ b_6$, wobei $b_1\ b_2$ als Zeilenindex und $b_3\ b_4\ b_5\ b_6$ als Spaltenindex genommen werden. So wird $b_1\ b_2 = 01 = 1$ und $b_3\ b_4\ b_5\ b_6 = 1010 = A$ ersetzt durch $1100 = C$.

Tabelle 9.2: Substitutionsbox S1 des DES

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
1	0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	8
2	4	1	E	8	D	6	2	B	F	C	9	7	3	A	5	0
3	F	C	8	2	4	9	1	7	5	B	3	E	A	0	6	D

Der wesentliche Vorteil symmetrischer Verfahren liegt darin, dass sich damit Datenraten von verschlüsselten Texten erzielen lassen, die etwa tausendfach höher liegen als die von asymmetrischen Verfahren. So kann mit einem DES-Chip eine Datenrate von ca. 7 GBit/s erreicht werden. Allerdings führt diese Tatsache auch dazu, dass sich solche Verfahren leichter analysieren (knacken) lassen. So bereitet der „kleine“ Schlüsselraum von 2^{56} möglichen Schlüsseln heute keine prinzipiellen Schwierigkeiten mehr, alle möglichen Schlüssel durchzuprobieren (engl. brute-force attack). DES-verschlüsselte Daten können bereits in wenigen Stunden entziffert werden. Dieses Verfahren kann daher nicht mehr als sicher gelten und Nachfolgeverfahren wie AES (Advanced Encrytion Standard) benutzen daher 128 Blockchiffren mit variabler und erhöhter Schlüssellänge von 128, 192 oder 256 Bit ($2^{256} = 1{,}1577921 \cdot 10^{79}$).

Das Haupteinsatzgebiet symmetrischer Verfahren liegt im Schutz der Vertraulichkeit. Durch zusätzliche Maßnahmen kann auch die Integrität der Daten erreicht werden. Hierfür kann der Chiffretext entweder über einen sicheren Kanal übertragen werden oder der ursprüngliche Chiffretext wird um eine kryptografische Prüfsumme, die auch Authentifizierungcode (engl. Message Authentication Code, MAC) genannt wird, da sie die Authentizität der Nachricht verifizierbar macht, ergänzt. Allerdings lässt sich dadurch bei symmetrischen Algorithmen noch keine Verbindlichkeit im Sinne der Abstreitbarkeit gegenüber Dritten herstellen. Dies bedeutet, dass hiermit die Verbindlichkeit der Daten nicht gewährleistet werden kann, sondern nur ihre Integrität gesichert ist, ohne aber genau zu wissen, ob der Sender der Nachricht auch tatsächlich der ist, der er zu sein vorgibt.

Eine gängige Sicherung der Datenintegrität besteht darin, kryptografische Hashfunktionen zu verwenden, um diese Prüfsummen zu erzeugen. Eine kryptografische Hashfunktion („Hackfleisch-Funktion“) h berechnet für einen beliebig langen Datensatz P einen Hashwert h(M) fester Länge l, der als Nachrichtenextrakt (engl. message digest) bezeichnet wird, bei dem es praktisch unmöglich ist – d.h. der exakte mathematische Beweis ist noch nicht gefunden –, aus diesem Hashwert auf den Eingabetext M zu schließen, d.h. die Berechnung der Umkehrfunktion h^{-1} ist praktisch nicht möglich (Einweg-Funktion). Gegenwärtig wird häufig die Hashfunktion MD5 (Message Digest5) verwendet, die einen Hashwert, der 128 Bit lang ist [RFC 1321], berechnet. Der Hashwert wird ebenfalls verschlüsselt und zusammen mit den chiffrierten Nutzdaten übertragen. Der Empfänger wendet auf den dechiffrierten Text wiederum die Hashfunktion h an und vergleicht diesen Hashwert mit dem übertragenen Wert. Stimmen beide überein, wurden die ursprünglichen Daten unverändert übertragen. Stimmen die beiden Werte nicht überein, ist die Integrität der Daten verletzt worden. Wenn

die Datenintegrität im Vordergrund steht, wird nur der Hashwert verschlüsselt und die Nachricht im Klartext übertragen.

Kryptografische Hashfunktionen, die auch als sichere Hashfunktionen bezeichnet werden, kommen nicht nur bei symmetrischen, sondern auch bei asymmetrischen Chiffrierverfahren zum Einsatz, da diese Funktionen die Integrität und nicht die Geheimhaltung der Daten sichern. Bei asymmetrischen Algorithmen dienen diese Funktionen auch dazu, digitale Signaturen zu erzeugen, mit denen beispielsweise die verbindliche Authentizität von Dokumenten sichergestellt wird (Abschn. 9.3.4).

Wir fassen zusammen: Symmetrische kryptografische Verfahren haben den Vorteil, dass sie hohe Verschlüsselungsraten erreichen. Sie bergen aber auch zwei wesentliche Nachteile in sich. Erstens brauchen Sender und Empfänger einen gemeinsamen, geheimen Schlüssel und die Übermittlung dieser Schlüssel, z. B. über einen sicheren Kanal oder über eine Zertifizierungsstelle oder Schlüsselverteilzentrale (engl. Trusted Third Party, TTP), ist in großen, offenen Netzen sehr aufwändig. Zweitens können sie nur die beiden Sicherheitskriterien Vertraulichkeit und Integrität garantieren, aber nicht die Verbindlichkeit von Daten.

9.3.3.2 Asymmetrische kryptografische Algorithmen

Öffentliche und private Schlüssel

Verfahren der asymmetrischen Kryptografie (engl. public key cryptography) gleichen die beiden zuvor genannten Nachteile aus, d. h. es braucht vor allem kein Schlüsselaustausch stattzufinden. Allerdings ist dafür als Preis eine deutlich geringere Verschlüsselungsrate zu zahlen. Das primär charakterisierende Merkmal dieser Verfahren ist der Gebrauch zweier verschiedener Schlüssel. Jeder Nutzer hat einen *öffentlichen Schlüssel* K_{pub} (engl. public key) und einen geheimen, *privaten Schlüssel* K_{priv} (engl. private key), den nur er kennt. Auf diese Weise wird die Asymmetrie erzeugt; Sender und Empfänger verwenden unterschiedliche Schlüssel.

Ein asymmetrischer kryptologischer Algorithmus erfüllt folgende Bedingungen:

1. Die Chiffrierfunktion E und die Dechiffrierfunktion D sind öffentlich.
2. Der Chiffrierschlüssel K_E ist öffentlich.
3. Der Dechiffrierschlüssel K_D ist privat, geheim und es gilt die Ungleichheit: $K_E \neq K_D$.
4. Aus der Kenntnis des öffentlichen Chiffrierschlüssel K_E muss es praktisch unmöglich sein, den geheimen Dechiffrierschlüssel K_D zu berechnen (Einweg-Funktion).
5. Nach dem Chiffrieren eines Klartextes $P \in \mathbf{P}$ mit dem öffentlichen Schlüssel K_E und dem Dechiffrieren mit dem privaten Schlüssel K_D wird wieder der Klartext erzeugt:
   $D(K_D, E(K_E, P)) = D(K_D, C) = P$.

Die asymmetrische Verschlüsselung von gewöhnlichen Nachrichten ist damit wie folgt durchzuführen. Wer den öffentlichen Schlüssel, z. B. mit der Länge von 2048 Bit, von Bob $K_{E,\,Bob}$ kennt, verschlüsselt seine Nachricht P mit diesem Schlüssel $E(K_{E,\,Bob}, P)$ und schickt sie an Bob. Dieser entschlüsselt dann den erhaltenen Chiffretext C mit sei-

nem privaten Schlüssel $K_{D, Bob}$ (Länge 2048 Bit) und liest den Klartext P = ($K_{D, Bob}$, C). Dadurch kann nur ein Empfänger, der den privaten Schlüssel kennt, den Klartext P lesen, womit die Vertraulichkeit der Daten gesichert ist.

Ein typischer Vertreter asymmetrischer kryptologischer Algorithmen ist der von Rivest, Shamir und Adleman entwickelte und nach ihnen benannte RSA-Algorithmus [Rivest 78], der quasi einen Standard für diese Art von Algorithmen darstellt. Dieses Verfahren beruht auf dem Ansatz, eine Einweg-Funktion für die Schlüsselerzeugung durch die Primfaktorenzerlegung natürlicher Zahlen zu erzeugen. Wir wollen im Folgenden erst kurz auf die zahlentheoretischen Grundlagen dieses Verfahrens eingehen, bevor wir danach den eigentlichen Algorithmus beschreiben.

Zahlentheoretische Grundlagen

Eine Primzahl ist eine natürliche Zahl p ($p \geq 2$), die außer 1 und p keine weiteren Zahlen als Teiler besitzt. Die ersten Primzahlen lauten: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, … Die bisher größte entdeckte Primzahl ist $2^{1257787} - 1$ und hat 378 632 Dezimalstellen. Nach dem Fundamentalsatz der Arithmetik lässt sich jede natürliche Zahl n ($n \geq 2$), die selbst keine Primzahl ist, in ein Produkt von Primzahlen p_i zerlegen: $p_1^{n_1} p_2^{n_2} \ldots p_s^{n_s}$, [Bronstein 01]. So gilt beispielsweise $672 = 2^5 \cdot 3 \cdot 7$. Die Anzahl der Primzahlen $\pi(x)$ mit $\pi(x) \leq x$, $x \in \mathbf{R}$ und $x \geq 2$, wird asymptotisch durch $\pi(x) \approx \lim_{x \to \infty} \frac{x}{\ln(x)}$ angegeben. Es gibt demnach $\frac{2^{256}}{\ln 2^{256}} \approx 6{,}5 \cdot 10^{74}$ Primzahlen der Länge von 256 Bit. Daraus schließen wir, dass zur Verschlüsselung genügend verschiedene Primzahlen vorhanden sind. Doch sollte man sich von dieser großen Zahl nicht täuschen lassen; es gibt wesentlich weniger Primzahlen als natürliche Zahlen, denn es gilt: $\lim_{x \to \infty} \frac{\pi(x)}{x} = 0$.

Jetzt müssen wir noch die Modulo-Funktion mod und die Euler-Funktion $\varphi(n)$ kurz beschreiben. Seien p und q zwei ganze Zahlen ($p, q \in \mathbf{Z}$), dann ist $p \equiv q \bmod M$ (lies: p und q sind *kongruent* modulo M), wenn gilt $p = q + kM$, wobei k eine beliebige ganze Zahl und M eine bestimmte, positive natürliche Zahl ist, die als *Modulus* bezeichnet wird.

Anders ausgedrückt bedeutet dies, dass p – q durch M teilbar ist. Es gibt genau M verschiedene Restklassen, die den gaußschen Restklassenring $Z_M = \{[0], [1], \ldots, [m-1]\}$ definieren, wobei alle Elemente einer Restklasse mit demselben Rest durch M teilbar sind. Hierbei beschreibt z.B. $[0] = \{z \mid z \in \mathbf{Z}, z \sim 0\}$ die Äquivalenzklasse, die durch die Menge aller zu 0 äquivalenten Elementen der Menge **Z** gehören. Beispielsweise gilt für m = 2: $[0] = \{0, \pm 2, \pm 4, \ldots\}$ und $[1] = \{\pm 1, \pm 3, \pm 5, \ldots\}$, wobei die Äquivalenzrelation ~ hier durch die Modulo-Funktion vorgegeben wird. So stehen $0 \sim 2$ für $0 \equiv 2 \bmod 2$, und $0 \sim 4$ für $0 \equiv 4 \bmod 2$, usw.

Statt immer die Differenz zweier Zahlen zu nehmen und sie durch M zu teilen, ist es einfacher und gleichwertig, jede einzelne Zahl einer Restklasse direkt ganzzahlig durch M zu teilen und nur den Rest dieser Division zu betrachten. So sind alle Elemente der Restklasse [0] gerade Zahlen, die sich mit Rest 0 durch 2 teilen lassen und die Elemente

der Restklasse [1] lassen sich alle mit Rest 1 durch 2 teilen und stellen daher die Menge der ungeraden Zahlen dar. Diese Sichtweise kann durch die kommutierte Schreibweise p = kM + q sofort verdeutlicht werden. Betrachten wir q = r als einen festen Rest, dann gehören alle Zahlen p, die dadurch entstehen, dass zu r alle Vielfachen von M hinzuaddiert werden, zu Restklasse [r]. Zum Beispiel gilt für M = 7: [5] = {0, ± 5, ± 12, ± 19…}, da p = 7 k + 5 mit k= 0, ± 1, … ist.

Mit der Modulo-Funktion wird eine *modulare Arithmetik* definiert [Cormen et al. 01]. Beispiele hierfür sind (M = 7):

Addition: $(5 + 6) \bmod 7 = 11 \bmod 7 = 4; (5+6) \equiv 4 \bmod 7$

Negation: $-6 \bmod 7 = (-6 + 7) \bmod 7 = 1 \bmod 7 = 1; -6 \equiv 1 \bmod 7.$

Subtraktion: $(9 - 6) \bmod 7 = (9 + (-6)) \bmod 7 = (9 + 1) \bmod 7 = 10 \bmod 7 = 3;$
$(9 - 6) \equiv 3 \bmod 7.$

Multiplikation: $(9 \times 6) \bmod 7 = 54 \bmod 7 = 5 \bmod 7; (9 \times 6) \equiv 5 \bmod 7.$

Multiplikative Inverse: $3^{-1} \bmod 7 = 5 \bmod 7$; $3^{-1} \equiv 5 \bmod 7$, da $3 \times 3^{-1} = 3 \times 5 \bmod 7 = 1$ gilt.

Division: $\frac{5}{3} \bmod 7 = 5 \times 3^{-1} \bmod 7 = 5 \times 5 \bmod 7 = 4; \quad \frac{5}{3} \equiv 4 \bmod 7.$

Eine multiplikative Inverse von n existiert nur dann, wenn n und der Modulus M teilerfremd sind, denn nur in diesem Fall gibt es zu einem n eine multiplikative Inverse n^{-1} für die gilt $n \times n^{-1} \equiv 1 \bmod M$. Dies trifft dann zu, wenn der Modulus M eine Primzahl ist. Und der Restring Z_M wird dann zu einem Feld (Galois Feld). Zwei natürliche Zahlen p und q sind genau dann *teilerfremd* (relativ prim), wenn deren größter gemeinsamer Teiler ggT gleich 1 ist: ggT(p, q) = 1. Der ggT wird mit dem Algorithmus von Euklid bestimmt [Schöning 01].

Die eulersche φ(n)-Funktion berechnet für eine von Null verschiedene natürliche Zahl n, die Anzahl aller positiven natürlichen Zahlen m ($m \le n$), die teilerfremd zu n sind. Wenn n keine Primzahl ist, d.h. die zuvor angegebene Faktorisierung in Primzahlen p_i möglich ist, dann gilt: $\varphi(n) = n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right) \ldots \left(1 - \frac{1}{p_s}\right) = n \prod_{p_i \mid n} \left(1 - \frac{1}{p_i}\right)$, wobei das Produkt über alle Primzahlen (ohne Berücksichtigung der Potenzen) zu bilden ist, die Teiler von n sind. Wenn n eine Primzahl ist, dann vereinfacht sich die Formel zu $\varphi(n) = n - 1$, denn alle natürlichen Zahlen von 1 bis n–1 sind in diesem Fall teilerfremd zu n und die obige Formel vereinfacht sich zu: $\varphi(n) = n\left(1 - \frac{1}{n}\right) = n - 1$.

Für n = 672 gilt beispielsweise $\varphi(672) = 672\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{7}\right) = 672 \cdot \frac{2}{7} = 192$ und für n = 7 ist φ(7) = 6, da alle natürlichen Zahlen von 1 bis 6 teilerfremd zu 7 sind.

Mit der eulerschen φ-Funktion kann auch die Anzahl der Restklassen berechnet werden, die zueinander teilerfremd sind. Dazu betrachten wir nur die Repräsentanten der

einzelnen Äquivalenzklassen, welche durch die Menge $Z'_M = \{0, 1, \ldots, M-1\}$ bestimmt sind, und berechnen $\varphi(M)$.

Als letzten vorbereitenden Punkt benötigen wir noch das eulersche Theorem, denn dieses liefert den wesentlichen Baustein für die Korrektheit des RSA-Verfahrens. Es besagt, dass

$$n^{\varphi(M)} \equiv 1 \bmod M, \ \forall n \in ZM, \ ggT(n,M) = 1.$$

gilt. Diese modulare Kongruenz gilt für alle Zahlen aus dem Restklassenring Z_M, die teilerfremd zum Modulus M (multiplikative Teilgruppe modulo M von Z_M) sind. Dieses Theorem vereinfacht sich für einen Modulus M, der eine Primzahl ist, zu

$$n^{M-1} \equiv 1 \bmod M, \ \forall n \in ZM, \ ggT(n,M) = 1.$$

(kleines Theorem von Fermat) und wird in dieser Form angewendet, um die Identität einer verschlüsselten Nachricht, die entschlüsselt wurde, zu garantieren („Inversenbildung“).

RSA-Algorithmus

Wir wenden uns nun dem eigentlichen RSA-Verfahren zu. Die Primzahlenfaktorisierung zur Berechnung von Einweg-Funktionen wird dadurch vereinfacht, dass nur zwei große Primzahlen (z.B. 256 oder 512 Bit lang) miteinander multipliziert werden, denn es ist einfach, zwei große Primzahlen zu finden und sie zu multiplizieren, aber praktisch unmöglich, rückwärts aus dem Produkt auf die beiden Primfaktoren zu schließen (Faktorisierungproblem). Folgende Schritte werden bei dem RSA-Algorithmus durchgeführt:

1. Schlüsselerzeugung

Ein Teilnehmer wählt zwei große Primzahlen p und q (z.B. 512 Bit lang) aus, berechnet den Modulus $M = p \cdot q$ und den Wert der eulerschen Funktion für M durch $\varphi(M) = M\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right) = (p-1)(q-1)$. Danach wird aus der Menge $Z_{\varphi(M)} = \{0, 1, \ldots, \varphi(M)-1\}$ eine Zahl e ($e < M$) von den $\varphi(\varphi(M))$ teilerfremden Zahlen zu $\varphi(M)$ zufällig ausgewählt und die multiplikative Inverse d von e mod $\varphi(M)$ berechnet: $e\,d \bmod \varphi(M) = 1$. Gelegentlich sagt man dazu auch: e ist modulo $\varphi(M)$ invertierbar.

Der öffentliche Schlüssel zum Chiffrieren ist das Paar $K_{pub} = (e, M)$, der private, geheime Schlüssel zum Dechiffrieren wird durch $K_{priv} = d$ definiert. Üblicherweise übernimmt eine Schlüsselvergabestelle die Berechnung dieser beiden Schlüssel und übergibt dem Teilnehmer die beiden Schlüssel und löscht diese beiden Schlüssel dann bei sich. Wenn ein Teilnehmer die Berechnung selbst vornimmt, müssen die Primfaktoren p und q und somit M nur ihm bekannt sein, denn sonst kann jeder aus dem öffentlichen Schlüssel $K_{pub} = (e, M)$, den privaten Schlüssel $K_{priv} = d$ berechnen.

Als Illustration diene ein Beispiel mit für die Praxis viel zu kleinen, dezimalen Zahlen. Sei $p = 47$ und $q = 59$, dann ist der Modulus $M = 2773$ und $\varphi(2773) = 2668$. Der Klartext

$P \in P$ sei 20. Wähle e = 17, welche Zahl teilerfremd zu 2668 ist. Bestimme die multiplikative Inverse d durch die Gleichung: 17 d mod 2668 = 1, was äquivalent zu 17 d = 2669 ist. Die Lösung ist d = 157. Somit gilt $K_{pub} = (17, 2773)$ und $K_{priv} = 157$.

2. Chiffrierung

Jeder Sender einer verschlüsselten Nachricht muss den öffentlichen Schlüssel des Empfängers $K_{pub} = (e, M)$ kennen, da er mit diesem Schlüssel seinen Klartext chiffriert. Mathematisch erfolgt dies durch eine modulare Exponentiation. Der Sender erzeugt den Chiffretext $C \in C$ einer Klartext-Nachricht P = 20 durch $C = P^e \bmod M$. Bleiben wir bei unserem Zahlenbeispiel, dann lautet der Geheimtext: $C = E(K_{pub}, P) = E((17, 2773), P) = 20^{17} \bmod 2773 = 2624$.

3. Dechiffrierung

Der Empfänger entschlüsselt den erhaltenen Chiffretext mit seinem privaten Schlüssel $K_{priv} = d$ ebenfalls durch eine modulare Exponentiation mit $P = C^d \bmod M$, denn mit Hilfe des kleinen fermatschen Satzes lässt sich beweisen, dass $C^d \bmod M = P^{ed} \bmod M = P$, wegen $e \cdot d \bmod \varphi(M) = 1$, für alle natürlichen Zahlen P gilt. Für unsere Beispielnachricht folgt damit $P = D(K_{priv}, C) = D(157, 2624) = C^d \bmod M = 2624^{157} \bmod 2773 = 20$.

Das bisher beschriebene asymmetrische Verfahren ist sicherer als ein symmetrischer Algorithmus, da die Berechnung des privaten Schlüssels d aus dem öffentlichen Schlüssel (e, M) nur mit einem weitaus größeren Aufwand als bei einem symmetrischen Algorithmus möglich ist, da dieser Aufwand äquivalent dazu ist, aus dem Modulus M die beiden Primfaktoren p und q zu finden. Für einen Modulus mit 140 Dezimalstellen benötigte man ca. drei Monate Rechenzeit, um das Faktorisierungsproblem zu lösen. Allgemein ist die Laufzeitkomplexität des Faktorisierungsproblems durch bisher bekannte Verfahren exponentiell $O[\exp(1{,}923\ (\ln M)^{1/3}\ (\ln \ln M)^{2/3})]$. Eine Attacke gegen den RSA-Algorithmus wäre ein Faktorisierungsalgorithmus, der polynomial ist, d.h. den Modulus M schnell in Primzahlen zerlegt und somit diesen Algorithmus unsicher werden lässt.

Wir haben bislang nur die Vertraulichkeit der Daten durch Chiffrierung gezeigt. Was noch fehlt, ist die Sicherstellung ihrer Integrität und Verbindlichkeit, wobei das letzte Kriterium von symmetrischen kryptografischen Algorithmen nicht erfüllt wird. Die Gewährleistung dieser beiden noch ausstehenden Sicherheitskriterien geschieht häufig mit Verfahren zur Authentifizierung und wird im nächsten Abschnitt behandelt.

9.3.4 Authentifizierung und Zertifizierung

Die Dienste zur Authentifizierung von Daten und von Personen müssen, wie bereits erwähnt, deutlich voneinander getrennt werden. Eine Nachricht, die beispielsweise um den Wert einer kryptografischen Hashfunktion (MAC) ergänzt wird, kann dadurch auf ihre Authentizität und somit auf ihre Datenintegrität überprüft werden. Diese Authentizität ist aber nicht verbindlich gegenüber Dritten. Eine verbindliche Authentizität (verbindlichen Datenintegrität) erhält eine Nachricht erst, wenn sie mit einer digitalen Signatur versehen wird; wenn es sich also um eine unterschriebene Nachricht handelt.

9.3.4.1 Digitale Signatur

Eine digitale Signatur übernimmt die Rolle einer konventionellen Signatur. Sie bestätigt die Authentizität des Absenders und die Integrität seiner von ihm gesandten Nachricht und stellt somit auch die Verbindlichkeit einer Nachricht her. Wegen dieses zusätzlichen Nachweises der Unversehrtheit einer Nachricht ist sie nicht nur eine bloße Unterschrift, sondern signiert jede unterschiedliche Nachricht auch unterschiedlich, sodass man besser von einer digital signierten Nachricht sprechen sollte. Trotzdem hat es sich eingebürgert, von einer digitalen Signatur zu sprechen.

Diese wird üblicherweise mit Hilfe einer kryptografischen Hashfunktion h auf folgende Weise erzeugt. Ein charakteristischer Klartext, der auch den Namen des Eigentümers enthält, wird durch h auf einen Hashwert fester Länge (z.B. 128 Bit) transformiert. Im nächsten Schritt wird dieser Wert mit dem privaten Schlüssel des Eigentümers verschlüsselt. Ein ganz einfaches Beispiel für eine digitale Signatur Sig ist die von Alice: $Sig_{Alice} = E(K_{priv, Alice}, h\ (Alice))$, welche nur ihren Namen chiffriert.

Im allgemeinen Fall wendet Alice die Hashfunktion h auf ihre Nachricht M an, signiert das Resultat mit ihrem privaten Schlüssel: $Sig_{Alice} = E(K_{priv, Alice}, h\ (M))$ und schickt $[M, Sig_{Alice}]$ an Bob. Dieser liest die unverschlüsselte Nachricht M, berechnet h(M) und entschlüsselt danach mit dem öffentlichen Schlüssel von Alice ihre digitale Signatur: $D(K_{pub,\ Alice}, Sig_{Alice})$. Wenn das Resultat dieser Dechiffrierung identisch ist mit dem von ihm berechneten h(M), ist die digitale Signatur von Alice gültig.

Betrachten wir das RSA-Verfahren, um eine digitale Signatur zu erzeugen. Die Unterzeichnerin Alice signiert ihren Klartext P mit ihrem privaten Schlüssel d: $Sig_{Alice} = E(K_{priv, Alice}, P) = P^d \bmod M$. Bob kennt den öffentlichen Schlüssel $K_{pub, Alice} = (e, M)$ von Alice und überprüft, ob $Sig^e_{Alice} \bmod M = P$ zutrifft. Wenn diese Gleichheit gilt, dann ist die Signatur echt. Wenden wir dies auf unser oben genanntes Beispiel an. Die Signatur Sig von P = 20 ist gleich $Sig = 20^{157} \bmod 2773 = 137$. Wir verifizieren: $137^{17} \bmod 2773 = 20$.

Formal wird ein asymmetrischer Signatur-Algorithmus wie folgt definiert.

Definition: Asymmetrischer Signatur-Algorithmus (9.3.2)
Ein *asymmetrischer Signatur-Algorithmus* ASA ist ein 6-Tupel ASA = (**K**, **P**, **S**, **B**, S, V), das sich aus 4 Mengen und zwei Funktionen zusammensetzt:

1. **K** ist der *Schlüsselraum*, der durch die Menge aller möglichen Schlüssel aufgebaut wird.
2. **P** ist der *Klartextraum*, der die Menge aller möglichen Klartexte enthält.
3. **S** ist der *Signaturraum*, der durch die Menge aller Signaturen definiert wird.
4. **B** ist die *boolesche Menge* **B** = {true, false}.
5. S ist die *Signaturfunktion* $S: \mathbf{K} \times \mathbf{P} \rightarrow \mathbf{S}$, die jeden Datensatz mit dem privaten Chiffrierschlüssel $K_{priv} \in \mathbf{K}$ verschlüsselt, um die Signatur $Sig = S(K_{priv}, P)$ zu erzeugen.
6. V ist die *Verifikationsabbildung* $V: \mathbf{K} \times \mathbf{P} \times \mathbf{S} \rightarrow \mathbf{B}$, die für jedes Eingabetripel (K_{pub}, P, Sig) bestimmt, ob $Sig = S(K_{priv}, P)$ wahr ist oder nicht.

Der RSA-Algorithmus ist sehr gut dazu geeignet, digitale Signaturen zu erzeugen. Der private Schlüssel wird verwendet, um die Signatur zu erzeugen, der öffentliche Schlüssel dient dazu, Nachrichten zu chiffrieren, damit sie vertraulich sind. Die Erklärung für diesen unterschiedlichen Schlüsselgebrauch ist offensichtlich. Eine Signatur muss mit Hilfe eines Geheimnisses generiert werden, welches nur dem Sender bekannt ist, doch sie sollte für alle verifizierbar sein.

Die Verbindlichkeit einer Nachricht ist durch eine digitale Signatur hergestellt, welche vor allem die Identität eines Teilnehmers aufzeigt. Von der gesamten Kette der sicheren Nachrichtenübertragung fehlen uns jedoch noch zwei wesentliche Glieder. Erstens muss der Benutzer sich dem Rechner gegenüber – und z.B. im elektronischen Handel auch umgekehrt – authentifizieren, um den Rechnerzugang zu erlangen (Benutzer-Authentifikation). Zweitens muss zusätzlich zur digitalen Unterschrift gewährleistet werden, dass der öffentliche Schlüssel tatsächlich zu dem Teilnehmer gehört, der er vorgibt zu sein (Zertifizierung). Beide Punkte wollen wir im Folgenden untersuchen.

9.3.4.2 Benutzer-Authentifizierung

Personen können mit drei verschiedenen Methoden für eine Zugangsberechtigung eindeutig identifiziert werden: durch charakteristische persönliche Merkmale (z.B. biometrische Merkmale), durch Wissen (z.B. Passwort) oder durch Besitz (z.B. Chipkarte). Wir wollen uns hier nur mit der zuletzt erwähnten Art beschäftigen.

Eine Chipkarte (Smartcard) ist eine kleine Plastikkarte, die einen eigenen Prozessor mit lokalem Speicher (Chip) besitzt, der im Gegensatz zu einer Magnetstreifenkarte beispielsweise kryptografische Algorithmen für die Authentifizierung durchführen kann. Eine solche Karte kann ferner die Zertifizierung samt dem öffentlichen Schlüssel speichern, sodass mit ihr die Benutzung verschiedener Dienste, wie der Bank oder des elektronischen Handels, aber auch der Zugang zu den verschiedenen Rechnern (Büro, zu Hause, unterwegs) und über diese zum Internet erreicht werden kann. Allerdings könnte es dabei passieren, dass diese persönlichen Daten auf allen Rechnern gespeichert sind, zu denen man mit Hilfe der Chipkarte im Lauf der Zeit Zugang erlangt hat.

Abbildung 9.9 verdeutlicht den Ablauf einer Ein-Weg-Authentifizierung von Alice bei einem Rechner (Terminal).

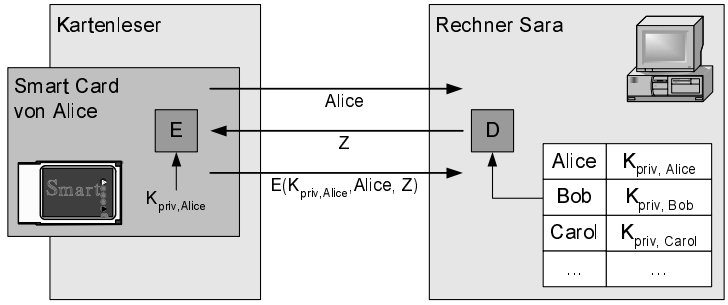

Abbildung 9.9: Ein-Weg-Authentifikation mit Chipkarte. Die Chipkarte wird von einem Kartenleser ausgelesen, der die Übertragungen zum und vom Rechner Sara übernimmt

Auf der Chipkarte ist eine Chiffrierfunktion E und der private Schlüssel K_{priv} gespeichert. Auf dem Rechner Sara befindet sich die zugehörige Dechiffrierfunktion D und eine Liste der privaten Schlüssel aller zugelassenen Personen. Alice meldet sich mit ihren Identifikationsdaten, z.B. ihrem Namen als Passwort. Daraufhin erzeugt Sara eine Zufallszahl Z und schickt sie der Chipkarte. Diese verschlüsselt die Nachricht (Alice, Z) durch $E(K_{priv, Alice}, (Alice, Z))$ und schickt sie an Sara. Wenn sich auf dem Zielrechner nach der Dechiffrierung (Alice, Z) ergibt, hat sich Alice authentifiziert. Dieser Ansatz kann noch etwas sicherer gemacht werden, indem Alice erst ihre PIN (Persönliche Identifikation Nummer) in den Kartenleser eingeben muss, damit sie sich zuerst gegenüber der Karte authentifiziert (Karte verifiziert PIN), bevor sie sich danach dem Rechner gegenüber eindeutig identifiziert.

Die Vorteile dieses Verfahrens liegen darin, dass es geschützt ist gegen die unautorisierte Benutzung bei Verlust der Karte, dass keine Geheimnisse übertragen werden und in der vorhandenen Aktualität, die durch die stets neue Zufallszahlengenerierung geschaffen wird (dynamische Authentifizierung). Der Nachteil liegt in dem Sicherheitsrisiko, dass der Zielrechner den geheimen Schlüssel des Benutzers kennen muss.

Selbstverständlich gibt es auch Authentifizierungsverfahren, bei denen sich nicht nur Alice bei Sara identifizieren muss, sondern sich auch umgekehrt Sara gegenüber Alice ausweisen muss (Zwei-Weg-Authentifikation). Dies ist beispielsweise beim elektronischen Handel wichtig, um zu wissen, ob der angesprochene Rechner eines Händlers auch der ist, der er zu sein vorgibt. Auf Details der gegenseitigen Benutzer-Authentifizierung wollen wir hier aber nicht eingehen, sondern verweisen z.B. auf [Coulouris, Dollimore, Kindberg 01].

Nach einer erfolgreichen Authentifizierung von Alice gegenüber dem Zielrechner Sara kann Alice auf Sara zugreifen, um beispielsweise mit ihrer digitalen Signatur Nachrichten zu verschlüsseln und danach zu versenden. Die Identifikation von Alice diente dabei nur der Zugangsberechtigung zum Zielrechner. Diese wird danach nicht weiter benutzt, um etwa die Verbindlichkeit der versandten Nachrichten zu garantieren, sondern es ist die digitale Signatur, die die Verbindlichkeit (Identität des Absenders und Nichtabstreitbarkeit) und die Integrität der Nachricht herstellt.

Was wir bislang noch nicht wissen, ist, wie die öffentlichen und privaten Schlüssel vergeben und verwaltet werden und warum digitale Zertifikate notwendig sind. Beides wollen wir in einem abschließenden Beispiel für die Zahlung mit einer Chipkarte, welche die Funktionen einer Kreditkarte mit einschließt, verdeutlichen.

9.3.4.3 Digitales Zertifikat

Ein *digitales Zertifikat* ist ein Dokument, das von einer öffentlichen und vertrauenswürdigen Institution, die als *Zertifizierungsstelle* (engl. Trust Center TC; Certification Authority CA) bezeichnet wird, einem Benutzer ausgestellt wird und eine Aussage enthält, die von dieser Stelle digital signiert wurde. Ein solches Zertifikat beglaubigt wie ein Ausweis die Echtheit eines öffentlichen Schlüssels und die Identität des Eigentümers dieses Schlüssels (beglaubigte Authentizität) und damit die Bindung dieses Schlüssels an eine bestimmte Person. Ein solches Zertifikat hat typischerweise (X.509) folgende Einträge:

Subject:	Name des Eigentümers, öffentlicher Schlüssel des Eigentümers.
Issuer:	Name der Zertifizierungsstelle, digitale Signatur dieser Stelle.
Issue Date:	TT/MM/JJ
Expiration Date:	TT/MM/JJ
Administrative Information:	Version, Seriennummer.

Eine Zertifizierungsstelle ist mit einer Ausweisbehörde vergleichbar. Sie berechnet bei einem asymmetrischen Verfahren auf Antrag den öffentlichen und privaten Schlüssel eines Teilnehmers. Das Zertifikat entspricht einem Ausweis. Der öffentliche Schlüssel entspricht z.B. der Staatszugehörigkeit, der Name des Eigentümers bleibt in seiner Bedeutung und die digitale Signatur des Passamtes entspricht der Ausweisnummer und dem Stempel der ausstellenden Behörde.

Zertifiziert werden vor allem öffentliche Schlüssel, denn die Verteilung und Verwaltung dieser Schlüssel muss offiziell und äußerst sorgfältig durch geführt werden. Digital zertifiziert werden aber auch Bankkonten von Kunden durch Zertifizierungsstellen der Banken, Mitgliedschaften bei Krankenkassen z.B. durch die Zertifizierungsstelle der kassenärztlichen Bundesvereinigung, die Zugehörigkeit zur Angestelltenversicherung durch die Zertifizierungsbehörde der Bundesanstalt für Angestellte und die Zugehörigkeit zu einer Hochschule durch deren lokale Rechenzentren.

Allgemein regelt in Deutschland das Signaturgesetz, wer zertifizieren darf und wie es zu geschehen hat. Alle Zertifizierungsstellen sind miteinander vernetzt, wobei die Wurzelinstanz durch die Regulierungsbehörde (Regulierungsbehörde Telekommunikation und Post: RegTP) gebildet wird. Direkt darunter befinden sich die Trust Center der Deutschen Telekom und der Deutschen Post. Eine weitere Zertifizierungsstelle wurde von dem DFN-Verein (DFN: Deutsches Forschungsnetz) eingerichtet [www.dfn-pca.de]. Diese Stelle heißt PCA-DFN (Policy Certification Authority) und soll uns als ein beispielhaftes Zertifikat des öffentlichen Schlüssels für Alice dienen:

Subject:	Alice Brown, mQGiBdxjm……
Issuer	DFN-PCA, id8DBQE85 ……
Issue Date:	01/01/2002
Expiration Date:	31/12/2002
Administrative Information:	V1, 12345678.

Der öffentliche Schlüssel und die digitale Signatur der ausstellenden Behörde sind beide in ASCII-Code dargestellt, gleichwohl unterscheiden sie sich üblicherweise in ihrer Länge deutlich. Die Maximallänge eines öffentlichen Schlüssels beträgt gegenwärtig 4096 Bit, eine häufig verwendete Länge der digitalen Signatur beträgt 1024 Bit. Die Signatur der Instanz kann zusammen mit ihrem öffentlichen Schlüssel dazu verwendet werden, die Authentizität der Zertifizierungsstelle zu verifizieren.

9.3.4.4 Elektronischer Handel

Machen wir uns das Zusammenspiel der Benutzer-Authentifikation, der digitalen Signaturen und digitalen Zertifikate mit einem asymmetrischen kryptografischen Algo-

rithmus an einem Einkaufsszenario mit einer Chipkarte innerhalb eines elektronischen Handels (engl. electronic commerce), wie es in Abbildung 9.10 festgehalten wird, klar. Wir konzentrieren uns dabei zuerst auf die auftretenden Rollen und eingesetzten Interaktionsmuster, um das Operationsprinzip dieses Ablaufes zu verstehen. Unterschieden werden die Rollen: Kunde, Händler, Annehmer (händlerabrechnende Bank), Abrechner (kundenabrechnende Bank) und Zertifizierer. Die zwischen diesen Rollen festgelegten Interaktionsmuster sind die gängigen Geschäfts- und Bestätigungsprozesse. Diese in der alltäglichen Welt üblichen Rollen werden bei der Implementierung jeweils auf die drei typischen Rollen abgebildet, die in Netzwerken vorhanden sind: Client, Server und Peer (Abschn. 8.4.1). Eng verknüpft damit sind die Interaktionsmuster: Client-Server und Team. Die Interaktionen zwischen Konsument und Händler erfolgen über das Internet, zwischen den Banken und den Zertifizierungsstellen (Trust Centers) in dem Finanznetz (Intranet) und die Wechselwirkungen zwischen der oberen Ebene (Konsument, Händler) und der unteren Ebene (Banken, TCs) Ausgabebank über die Kopplung der beiden Netze. Die Zuordnung von Rollen an einzelne Institutionen wird in der Zeichnung dadurch angezeigt, dass Rechner, Banken usw. neben den Rollen eingezeichnet wurden. Die Zuordnung von Rollen an Personen und Organisationen birgt eine eigenen Problematik in sich, auf die wir uns hier aber nicht näher einlassen wollen.

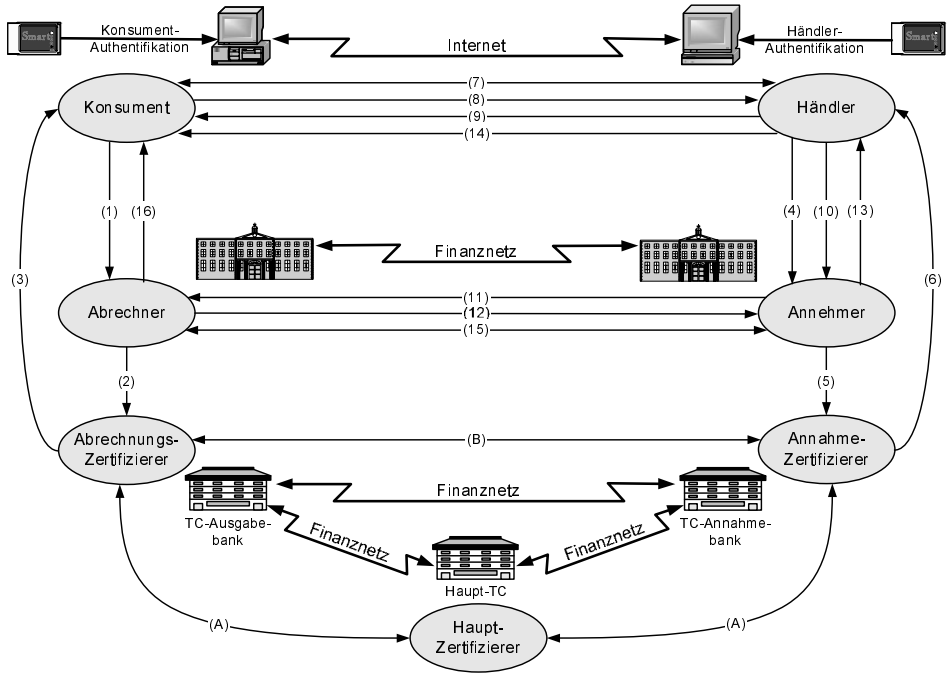

Abbildung 9.10: Einkauf mit Chipkarte, die in diesem Beispiel auch die Funktion einer Kreditkarte (Geldkarte) übernimmt. Rollen sind als Ovale dargestellt

Bevor der geschäftliche Kreislauf beginnt, authentifizieren sich der Konsument und der Händler bei ihren jeweiligen lokalen Rechnern. Der geschäftliche Kreislauf beginnt damit, dass ein Konsument (Client) eine Chipkarte bei dem Abrechner (Server) beantragt (1). Dieser leitet als Client den Antrag weiter an seine Zertifizierungsstelle (Server), die hier als Abrechnungs-Zertifizierer bezeichnet wird (2). Diese Stelle zertifiziert dem Konsumenten mit ihrer digitalen Signatur, dass er ein auf Bonität geprüfter Kontoinhaber bei der Abrechnungsbank mit einem bei ihr hinterlegten öffentlichem Schlüssel und einer bestimmten Kontonummer ist (3). Ferner erhält der Konsument dabei seinen privaten Schlüssel und die Chipkarte ausgehändigt. Analog dazu verfährt der Händler. Er beantragt als Client beim Annehmer (Server) für sich einen Akzeptanzvertrag (4). Der Annehmer wird zum Client und leitet den Antrag an den Annahme-Zertifizierer (Server) weiter (5). Danach wird dem Händler zertifiziert, dass er ein Handelspartner des Annehmers ist (6). Jetzt können der Konsument und der Händler zum Vertragsabschluss kommen. Dafür leiten sie zuerst eine wechselseitige Zwei-Weg-Authentifikation ein (7). Der Konsument (Client) bestellt, gibt den Zahlbetrag an, signiert alles digital und schickt den unverschlüsselten Text (Auftrag) zusammen mit der digitalen Signatur an den Händler (8). Allerdings kann ein nicht verschlüsselter, signierter Text auf dem Übertragungsweg von Dritten gelesen werden, ohne dass der Empfänger es merkt, da durch die digitale Signatur nur die Integrität (Unversehrtheit) des Textes gewährleistet wird. Daher muss der Konsument den Klartext zusätzlich verschlüsseln, damit dieser auch vertraulich bleibt.

Der Händler entnimmt dem Konsumenten-Zertifikat den öffentlichen Schlüssel des Konsumenten und kann den Auftrag als verbindlich betrachten. Was er allerdings nicht weiß, ist, ob das Konsumenten-Zertifikat echt ist und somit der angegebene Abrechner tatsächlich für den Konsumenten ein Konto mit der vorgegebenen Kontonummer eingerichtet hat. Um hier Gewissheit zu erlangen, muss er die digitale Signatur des Abrechnungs-Zertifizierers auf dem Konsumenten-Zertifikat verifizieren. Was der Händler braucht, ist somit ein Zertifikat über den Abrechnungs-Zertifizierer, um an dessen öffentlichen Schlüssel zu gelangen. Dieses Zertifikat muss eine übergeordnete Zertifizierungsstelle ausstellen. In Abbildung 9.10 ist daher eine flache Hierarchie von Zertifizierungsstellen eingezeichnet und der Haupt-Zertifizierer (Wurzel-Zertifizierer) muss diese Aufgabe übernehmen (A). Dafür muss aber eine Vernetzung zwischen ihm und dem Abrechnungs-Zertifizierer über das Finanznetz geschaffen sein. Des Weiteren müssen sich auch der Abrechner mit dem Annehmer gegenseitig als vertrauenswürdige Partner zertifizieren. Dies kann entweder jeweils über die Wurzelinstanz gehen (A), oder die beiden Partner „trauen“ sich und zertifizieren sich direkt gegenseitig (B). Das Interaktionmuster, das zwischen den Zertifizieren auf derselben Ebene angewendet wird, ist das Teammuster. Zwischen den Hierarchieebenen gilt das Client-Server-Modell. Hier wird ganz ähnlich verfahren wie bei der nicht-rekursiven Namennavigation.

Kehren wir zu unseren ursprünglichen geschäftlichen Interaktionsmustern zurück. Entweder vertraut der Händler dem signierten Angebot und bestätigt es gleich, oder er versichert sich vorher direkt über den Haupt-Zertifizierer, wie wir es oben beschrieben haben, oder er wendet sich an seinen Annahme-Verifizierer, damit dieser das Zertifikat über die Abrechnerbank übernimmt. Gleichgültig, welchen Weg der Händler ein-

schlägt, am Ende muss er dem Konsumenten den Auftrag bestätigen (9). Danach autorisiert der Händler (Client) den Annehmer (Server), die Bonität des Kunden prüfen zu lassen (10). Der Annehmer wechselt zum Client und beauftragt die Bonitätsprüfung beim Abrechner (Server) und die Reservierung des Betrags (11). Es folgt die Bestätigung der Reservierung des Abrechners beim Annehmer (12). Der Annehmer wiederum gibt eine Kaufgenehmigung an den Händler (13). Der Händler übergibt die Ware an den Konsument (14). Nach vorangegangener Abrechnung zwischen Händler und Annehmer findet eine gegenseitige Verrechnung zwischen Annehmer (Partner) und Abrechner (Partner) statt (15). Zum Schluss stellt der Abrechner dem Konsumenten eine Rechnung bzw. zieht den Zahlbetrag von seinem Konto ab (16).

Sobald der negative Fall eintritt, dass irgendeine Verifikation in dem gesamten Prozessablauf nicht vorgenommen werden kann, wird der ganze Vorgang abgebrochen.

9.4 Java: verteilte objektorientierte Programmiersprache

Java ist die bedeutendste neue, objektorientierte Programmiersprache, sie wurde 1995 von der Firma Sun entwickelt. Sie nimmt die Konzepte ihrer vier „Mütter“ C++ (z.B. lokale Objektorientierung), Smalltalk (z.B. virtuelle Maschine), Ada (z.B. Klassenpakete zum Programmieren im Großen) und Modula 3 (z.B. entfernter Methodenaufruf) in sich auf, integriert sie in einer neuartigen Kombination in eine eigene Sprache und erweitert sie derart, dass sie sich zur Programmierung verteilter Anwendungen (Systeme) besonders eignet [Arnold, Gosling, Holmes 01], [Hughes, Shoffner, Hamner 99], [Goll, Weiß, Müller 01], [Jobst 02], [http://java.sun.com/]. Uns interessieren in diesem Zusammenhang vor allem zwei wesentliche Fragen: Erstens die Architektur dieser Sprache, d.h. worin bestehen die herausragenden Konzepte (Operationsprinzipien) und wie sieht die daraus abgeleitete Struktur dieser Sprache aus, die sie für die Programmierung verteilter Anwendungen prädestinieren. Und zweitens, welche erstmals in eine Sprache integrierten Sicherheitsvorkehrungen bietet sie?

Es wird sich zeigen, dass durch die Integration vor allem von Betriebssystemdiensten in die Sprache durch die Java-Klassenbibliothek die Bereitstellung von Programmierschnittstellen für Internet-Protokolle, die Fähigkeit zur Strukturverflachung von Daten für die Übertragung (Serialisierung, Deserialisierung), die Realisierung von Aufrufen zur Interprozess-Kommu-nikation und von entfernten Prozeduraufrufen (Interprozess-Kommunikation nach dem Client-Server-Muster) bzw. von entfernten Methodenaufrufen und die Unabhängigkeit der Programme von einer Rechnerplattform (Hardware und Betriebssystem) durch die virtuelle Java-Maschine Java mehr als nur eine Sprache ist. Java realisiert dadurch typische Funktionen von Middleware (Def. 8.1.6) und kann daher auch als ein Werkzeug (Entwicklungs- und Laufzeitumgebung) für verteilte Anwendungen betrachtet werden. In unserer Beschreibung gehen wir von der Version 1.4 aus.