# Sicherheit
# in
# verteilten virtuellen Umgebungen

Vom Promotionsausschuss der
Technischen Universität Hamburg-Harburg
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
genehmigte Dissertation

von

Jan Köhnlein

aus München

2005

| | |
|---|---|
| Gutachter: | Prof. Dr. Helmut Weberpals |
| | Prof. Dr. Dieter Gollmann |
| Tag der mündlichen Prüfung: | 28. September 2005 |

# Vorbemerkungen

## Danksagung

Zuerst möchte ich einigen Menschen danken, die mich auf unterschiedliche Art und Weise bei meiner Promotion unterstützt haben: Prof. Weberpals, mein Doktorvater, hat mich besonders in schwierigen Phasen zum Durchhalten motiviert. Meine Lebensgefährtin Martina Gründler stand mir stets mit viel Geduld und Liebe zur Seite, auch wenn mich die Arbeit manchmal sehr in Anspruch genommen hat. Jens Zemke sorgte für kulinarische Abwechslung in vielen Mittagspausen. Axel Wienberg lieferte wertvolle fachliche Kommentare zur Arbeit, und meine Mutter Edda Köhnlein stöberte in der Urfassung viele orthographische und grammatikalische Fehler auf.

## Formale Konventionen

Die deutschen Übersetzungen englischer Fachtermini aus dem Bereich der Kryptographie habe ich aus [Sch96] übernommen, soweit sie mir präzise genug erschienen. Ansonsten werden die auch in der deutschen Fachsprache mittlerweile üblichen englischen Fachbegriffe benutzt und durch *Kursivschrift* gekennzeichnet.

*Kursivsatz* wird ebenfalls zur Hervorhebung von Begriffen und zur Kennzeichnung von Definitionen eingesetzt. Für Programmfragmente verwende ich `Schreibmaschinenschrift`. Die Namen von Bibliotheken, Programmpaketen, Klassen, Objekten, Attributen und Methoden sowie URLs sind in serifenloser Schrift gesetzt.

# Inhaltsverzeichnis

## Kapitel 1

# Einführung

Seit Einführung des Internets ist mit der Anzahl der vernetzten Rechner auch das Kommunikationsvolumen enorm gewachsen. Breitbandanschlüsse, die vor wenigen Jahren aus Kostengründen nur großen Institutionen und Unternehmen zugänglich waren, sind mittlerweile auch für Privatanwender erschwinglich und erfreuen sich zunehmender Beliebtheit. Gleichzeitig sinken die Preise für immer bessere Rechnerhardware. Die Leistungsfähigkeit von Grafikprozessoren steigt schneller als die der Hauptprozessoren.

Das stetige Wachstum verfügbarer Bandbreite und Graphikleistung begünstigt eine bestimmte Klasse von Anwendungen: Online-Computerspiele, in denen sich die Spieler in einer simulierten Welt aneinander messen können, virtuelle *3D-Chat-Rooms*, für einen zwanglosen Plausch in einer dreidimensionalen Welt, industrielle Konstruktionssysteme, in denen mehrere Entwickler gemeinsam dreidimensionale Entwürfe anfertigen können, medizinische Anwendungen zur Steuerung von Robotern oder für Fernoperationen, militärische Simulationssysteme, mit denen Einsatzszenarien eingeübt und analysiert werden können. Alle diese Anwendungen haben eins gemeinsam: Sie simulieren eine dreidimensionale Szene mithilfe mehrerer vernetzter Rechner. Wir nennen diese Anwendung *verteilte virtuelle Umgebungen*. In ihnen wird die Science-Fiction-Vision des *cyberspace* verwirklicht.

Ein Grund für das massive Wachstum des Internets ist seine offene, dezentral erweiterbare Struktur: Niemand kann global kontrollieren, wer sich mit dem Netz verbindet oder zu welchem Zweck es benutzt wird. Das Internet wurde ursprünglich von Universitäten, von zivilen Forschungseinrichtungen und vom Militär als kooperatives Netzwerk entwickelt, in dem jeder angeschlossene Rechner vertrauenswürdig ist. Daher sind keine Maßnahmen zum Schutz der vernetzten Rechner voreinander direkt in der Internet-Technologie implementiert. Die Folgen davon erleben wir täglich: *Spam-E-Mail*, die wir nur erhalten, weil irgendjemand auf irgendeine Weise an unsere E-Mail-Adresse gekommen ist, Viren und Trojaner, die auf dem eigenen Rechner unbemerkt persönliche Daten ausspionieren oder gleich die vollständige Kontrolle übernehmen, der Ausfall von Web-Servern großer Unternehmen, die gleichzeitig von Tausenden kompromittierten Rechnern mit sinnlosen Nachrichten bombardiert werden, gefälschte Einkaufsportale zum Ausspionieren von Kreditkartennummern, abgehörte E-Mails usf.

Das Internet selbst bietet keinen Mechanismus zum Schutz privater Daten. Es gibt keine Garantie dafür, dass ein vernetzter Rechner verfügbar bleibt, dass Nachrichten korrekt adressiert oder unverändert ausgeliefert werden. Erst die *Sicherheitstechnik* liefert Methoden zur Analyse von Sicherheitslücken und möglichen Angriffen, und stellt Mechanismen zum Schutz bzw. zur Abwehr bereit, die dann in Anwendungen integriert werden können.

Es kann allerdings kein universelles, für alle Anwendungen einsetzbares Sicherheitskonzept ge-

ben, denn neue Sicherheitsmechanismen verschlechtern oft die Benutzbarkeit und Leistung des Systems, z.B. wenn zusätzlich Passwörter eingegeben werden müssen oder zusätzlicher Aufwand für die Verwaltung der Schlüssel kryptographischer Operationen betrieben werden muss. Aus diesem Grund werden oft sogar vorhandene Sicherheitsmechanismen nicht benutzt, beispielsweise die Verschlüsselung von E-Mails. Außerdem sind die Sicherheitsanforderungen und die wahrscheinlichen Angriffe anwendungsspezifisch, und für jeden Einsatz müssen die benötigten Sicherheitsmechanismen und die damit verbundenen Einbußen neu ausbalanciert werden.

Mit dieser Arbeit schlage ich die Brücke zwischen den beiden Themen *Sicherheitstechnik* und *verteilte virtuelle Umgebungen*. Dabei berücksichtige ich sowohl die gemeinsamen Eigenschaften verteilter virtueller Umgebungen als auch die meist anwendungsspezifischen Sicherheitsanforderungen. Nach ausführlicher Analyse entwerfe ich eine speziell angepasste Sicherheitsarchitektur und stelle deren Implementierung vor.

## 1.1 Thesen und Methodik

Als Ansatz dieser Arbeit dienen die folgenden Thesen:

- Die Sicherheit muss bereits in der grundlegenden Architektur einer verteilten virtuellen Umgebung verankert werden.
- Eine effektive und gleichzeitig effiziente Sicherheitsarchitektur muss speziell auf den Anwendungsfall abgestimmt werden.
- Verteilte virtuelle Umgebungen sind mit geringen Einbußen in der Skalierbarkeit und der Echtzeitfähigkeit sicher implementierbar.

Um diese Thesen zu beweisen, wähle ich den folgenden methodischen Weg:

- Zuerst werden die Eigenschaften und Anforderungen verteilter virtueller Umgebungen beschrieben und von denen anderer verteilter Systeme abgegrenzt.
- Darauf basierend wird eine speziell angepasste Kommunikationsbibliothek entwickelt und implementiert, in die unterschiedlich starke anwendungsspezifische Sicherheitsmechanismen für die Nachrichtenübertragung integriert und deren Auswirkung getestet werden können.
- Danach folgt eine Sicherheitsanalyse unter Berücksichtigung verschiedener Anwendungsszenarien und Rollen. Anhand zweier Beispiele werden die anwendungsspezifischen Unterschiede verdeutlicht.
- Aus allen gewonnenen Erkenntnissen werden die Bausteine für eine umfassende anpassbare Sicherheitsarchitektur entworfen, implementiert und anschließend bewertet.

Entwurfs- und Implementierungsdetails werden, wenn möglich, in Diagrammen der *Unified Modeling Language* (UML, [RJB99]) dargestellt. Ansonsten wurde eine selbsterklärende Notation gewählt.

Die Implementierungssprache ist Java [Java]. Die folgenden Eigenschaften machen Java in diesem Fall besonders attraktiv:

- Java ist objektorientiert, was insbesondere einen klaren Entwurf erheblich erleichtert.

- Java ist für fast alle modernen Rechnerplattformen verfügbar. Sogar kompilierte Java-Programme sind portabel.
- Es existiert ein großer Fundus an standardisierten Java-Klassenbibliotheken, insbesondere für kryptographische Algorithmen, Netzwerkkommunikation und die Nutzung von Standarddiensten wie LDAP.
- Java-Programme sind sehr robust.
- Java selbst bietet eine integrierte Sicherheitsarchitektur. So können Programme beispielsweise abgesichert in einer kontrollierbaren *sandbox* mit einem frei wählbaren Satz von Privilegien und Zugangsrechten gestartet werden.
- Java hat sich inzwischen als Implementierungssprache für Spiele auf portablen Geräten wie Mobiltelefonen oder PDAs durchgesetzt.

## 1.2 Gliederung

Die Arbeit gliedert sich wie folgt: In Kapitel 2 definiere ich den Begriff *verteilte virtuelle Umgebungen*, liefere Beispiele und stelle die spezifischen Eigenschaften und Herausforderungen dieser Systeme dar. In Kapitel 3 wird der Kommunikationsaspekt verteilter virtueller Umgebungen genauer analysiert. Davon ausgehend entwerfe und implementiere ich eine neue speziell angepasste Kommunikationsbibliothek. Grundlegende Begriffe und Methoden der Sicherheitstechnik werden in Kapitel 4 behandelt. Kapitel 5 beginnt mit der anwendungsspezifischen Sicherheitsanalyse von verteilten virtuellen Umgebungen, auf deren Basis dann eine maßgeschneiderte Sicherheitsarchitektur entworfen und implementiert wird.

# Kapitel 2

# Verteilte virtuelle Umgebungen

Da die Fachliteratur bisher wenig Allgemeines über verteilte virtuelle Umgebungen bietet, habe ich, basierend auf meiner früheren Veröffentlichung [KW00], in diesem Kapitel die wichtigsten Aspekte verteilter virtueller Umgebungen zusammengetragen. Nach der Definition grundlegender Begriffe in Abschnitt 2.1 und Erläuterung einiger Beispiele in Abschnitt 2.2 werden in Abschnitt 2.3 die wesentlichen Anforderungen einer verteilten virtuellen Umgebung analysiert. Da die Kommunikation den entscheidenden Faktor für die Skalierbarkeit darstellt, werden in Abschnitt 2.4 Techniken zur Verringerung der Netzwerklast gezeigt. In Abschnitt 2.5 werden dann typische Entwurfsmuster vorgestellt. Das Kapitel endet mit einer Zusammenfassung der wichtigsten Ergebnisse.

## 2.1 Definitionen

In der im Wesentlichen englischsprachigen Literatur finden sich für verteilte virtuelle Umgebungen eine Vielzahl von Begriffen: *Distributed Virtual Environment* (DVE), *Networked Virtual Environments* (NetVE), *Distributed Interactive Simulation* (DIS), *cyberspace*, *shared virtual reality*, *distributed virtual world*, *Collaborative Virtual Environment* (CVE) usf. Der Begriff *verteilte virtuelle Umgebung* bedeutet in dieser Arbeit:

> Eine *verteilte virtuelle Umgebung* ist eine dreidimensionale Szene, die von mehreren Rechnern gleichzeitig generiert, dargestellt und verändert wird.

Analog werden in dieser Arbeit auch die Begriffe *Szene* und *Simulation* verwendet. Als Abkürzung verwende ich im Folgenden DVE.

Alternative Definitionen sind möglich. So definieren Singhal und Zyda [SZ99] ein *networked virtual environment* als ein Softwaresystem, das über die folgenden fünf Eigenschaften verfügt:

1. Ein gemeinsamer Eindruck von Raum.
2. Ein gemeinsamer Eindruck von Präsenz.
3. Ein gemeinsamer Eindruck von Zeit.
4. Eine Möglichkeit der Kommunikation.
5. Eine Möglichkeit des Teilens[1].

[1] Gemeint ist eine Möglichkeit der Interaktion mit der Umgebung selbst.

Diese Definition stützt sich stärker auf den Eindruck auf einen menschlichen Benutzer und ist daher weniger technisch, aber auch weniger konkret. In dieser Arbeit wird die erste Definition auch deswegen bevorzugt, weil sie sich nicht auf ein reines Softwaresystem bezieht, und somit die Verwendung von multimedialer Hardware (vgl. Abschnitt 2.3.1) mit einschließt.

Die Szene entsteht also durch die Zusammenarbeit mehrerer vernetzter Rechner. Die Menge aller an einer verteilten virtuellen Umgebung beteiligten Rechner bildet den *Verbund*. Ein Rechner im Verbund heißt *Teilnehmer*. Einen Menschen, der über einen Teilnehmer an der Simulation partizipiert, nennt man *Benutzer*.

Eine Szene enthält diverse virtuelle Gegenstände, deren Zustand sich dynamisch während der Laufzeit der Simulation ändern kann oder deren Zustand statisch bleibt. Ein solcher statischer oder dynamischer Gegenstand in einer verteilten virtuellen Umgebung heißt *Entität*. Ein *Avatar* ist eine dynamische Entität, die einen Benutzer in der virtuellen Szene repräsentiert. Der Benutzer selbst kann seinen Avatar in vielen Fällen nicht sehen; der Avatar bestimmt allerdings die virtuelle Kameraposition, aus der die Welt beobachtet wird.

Eine *Aktion* ist die Änderung des Zustands einer dynamischen Entität, z.B. „Avatar bewegt sich nach Süden“. Aktionen werden durch *Ereignisse* ausgelöst. Ereignisse können direkt von einem Benutzer ausgelöst oder automatisch generiert werden. Die Zeit, die zwischen dem Auslösen eines Ereignisses und der tatsächlich sichtbaren Durchführung der Aktion vergeht, heißt *Antwortzeit*.

## 2.2 Beispiele

Um die theoretischen Definitionen aus dem letzten Abschnitt mit Leben zu füllen, werden in diesem Abschnitt einige Beispiele für verteilte virtuelle Umgebungen vorgestellt.

### 2.2.1 Mehrspieler-Online-Computerspiele

Typische Vertreter von verteilten virtuellen Umgebungen sind netzwerkfähige Computerspiele, bei denen die Spieler in einer virtuellen Welt interagieren. Als Verbindungsnetzwerk fungiert dabei entweder ein *Local Area Network* (LAN), das Internet oder direkte Verbindungen, wie z.B. Null-Modem-Kabel.

Während einige Spiele eher auf Kommunikation und Kooperation zwischen den Benutzern basieren, geht es in anderen um Konkurrenz bis hin zum direkten Kampf der Benutzer gegeneinander, wie in einem so genannten *first-person-shooter*.

In *Massive Multiplayer On-line Role Play Games* (MMORPGs) interagieren viele Benutzer in einer Fantasie-, Märchen- oder Science-Fiction-Welt mit dem Ziel, eine möglichst mächtige Spielfigur zu entwickeln und wertvolle Gegenstände zu sammeln.

Unterschieden wird zwischen *rundenbasierten Spielen* und *Echtzeit-Spielen*. In *rundenbasierten Spielen* wird der Spielablauf in Spielrunden unterteilt, in denen alle Spieler gleichzeitig oder nacheinander ihre Aktionen definieren. Am Ende jeder Runde wird der neue Spielstand aus den gesammelten Aktionen berechnet und zwischen den Teilnehmern synchronisiert. Im Gegensatz dazu werden die Aktionen der Spieler in *Echtzeit-Spielen* sofort verarbeitet. Rundenbasierte Spiele sind in der Regel langsamer und stärker strategisch orientiert, während Echtzeit-Spiele aktionsgeladener und interaktiver sind. Durch die Synchronisation am Ende einer Runde kann der Zustand der Szene bei den rundenbasierten Spielen einfacher konsistent gehalten werden (vgl. Abschnitt 2.3.3). Aus diesem Grund sind viele Spiele, die auf den ersten Blick wie Echtzeit-Spiele aussehen, in Wirklichkeit rundenbasiert, allerdings mit einer extrem kurzen Rundendauer vom Bruchteil einer Sekunde.

Nicht zeitkritische rundenbasierte Spiele werden in dieser Arbeit nicht behandelt, da es sich dabei fast nie um DVEs handelt.

Eines der ersten Mehrspieler-3D-Spiele war SGIs *Flight* (später *Dogfight*), das zwischen 1984 und 1992 als Demonstrationsprogramm mit jeder SGI-Workstation geliefert wurde. Die spätere Freigabe des Quellcodes inspirierte damals viele Programmierer, selbst vernetzte Computerspiele zu entwickeln. Der erste bekannte mehrspielerfähige *first-person-shooter* war Doom [Doom], das besonders durch die kostenlose Verteilung einer Demo-Version rasch weite Verbreitung fand. Der Doom-Nachfolger *Quake* ist inzwischen als Quellcode [Qua] verfügbar, und wird daher oft als Testumgebung verwendet. Bekannte MMORPGs sind *Ultima Online* [UO] und *Everquest* [EQ].

### 2.2.2 Militärische Simulation

#### DIS

Auch das Militär entwickelt verteilte virtuelle Umgebungen, um strategische Szenarien zu analysieren oder Kampfeinsätze zu simulieren. Die US-amerikanische *Defense Advanced Research Projects Agency* (DARPA) entwickelte 1993 den Standard *Distributed Interactive Simulation* (DIS, [DIS]), der Kommunikation, Simulationsumgebung, Wiedergabetreue, Übungskontrolle und Feedback einer verteilten virtuellen Umgebung definiert.

DIS ist eine Erweiterung seines Vorgängers SIMulator NETworking (SIMNET, [MT95]). Die wichtigsten Architekturkonzepte sind [DIS]:

- Es gibt keinen zentralen Rechner, der die gesamte Simulation kontrolliert.
- Autonome Simulationsanwendungen verwalten den Zustand einer oder mehrerer Simulationsentitäten.
- Zur Kommunikation der Entitätszustände wird ein standardisiertes Protokoll verwendet.
- Die Zustandsänderungen werden von den Simulationsanwendungen kommuniziert.
- Die Wahrnehmung von Ereignissen wird von der empfangenden Anwendung bestimmt.
- Um die Verarbeitung der Kommunikationsdaten zu reduzieren, werden die Positionen der Entitäten extrapoliert und nur bei starken Abweichungen korrigiert (vgl. Abschnitt 2.4.1).

DIS schreibt außerdem die Verwendung von *Multicasting* vor und definiert zehn verschiedene Nachrichtentypen, die so genannten *Protocol Data Units* (PDUs), mit denen Ereignisse kommuniziert werden. Der wichtigste Nachrichtentyp ist die *Entity State PDU* (ESPDU), die die wichtigsten Zustandsdaten einer Entität, wie Position, Geschwindigkeit und Typ enthält.

DIS ist für eine Größe bis zu 300 Entitäten pro Szene konzipiert. Diese Skalierbarkeitsgrenze wird im Wesentlichen durch die Bandbreite des Netzwerks gesetzt. Die von jeder Entität mindestens alle 5 Sekunden gesendeten ESPDUs tragen viele redundante Informationen und sind daher relativ groß. Sie machen über 90% des Netzwerkverkehrs aus.

Im Projekt *DIS-Java-VRML* [DJV] wurde ein DIS-Klient auf Basis der *Virtual Reality Markup Language* (VRML, [VRML]) und der Programmiersprache Java [Java] implementiert.

### HLA

Die Entwicklung von Simulationssystemen, die wie DIS auf der Protokollebene standardisiert sind, stellte sich als zu aufwändig und kostspielig heraus. Auch durch die immer populärer werdenden objektorientierten Programmiersprachen entstand die Notwendigkeit eines Standards auf einer höheren, abstrakteren Programmebene. Mit der *High-Level-Architecture* (HLA, [HLA]) wurde eine solche standardisierte Softwarearchitektur für verteilte Simulationen entwickelt. Der Standard beschreibt die Regeln, eine Objektmodell-Schablone[2] für die in der Simulation ausgetauschten Daten und die Schnittstellenspezifikationen der einzelnen Komponenten.

Eine Simulation besteht in HLA aus einzelnen Komponenten, den *federates*, die über eine *Runtime Infrastructure* (RTI) zu einer *federation* verbunden werden. *Federates* können eine oder mehrere Entitäten modellieren, Daten sammeln und/oder anzeigen oder eine Schnittstelle für einen menschlichen Benutzer darstellen. Jeder *federate* implementiert die FederateAmbassador-Schnittstelle, und benutzt die RTI über die RTIambassador-Schnittstelle. Die Kommunikation wird durch ein Verleger-Abonnent-System (vgl. Abschnitt 2.5.2) abstrahiert, in dem sich *federates* als Produzent oder Empfänger spezieller *events* anmelden können.

HLA bietet außerdem Mechanismen zur Synchronisation der *federation* unter Benutzung *logischer* Zeit. Jeder *federate* kann als *Zeit regulierend* (englisch *time-regulating*) eingestuft werden, und damit das Fortschreiten der logischen Zeit für den Rest der *federation* steuern. Darüber hinaus können *federates* auch *zeitbeschränkt* (englisch *time-constrained*) sein, also anderen *federates* die Steuerung der eigenen logischen Zeit erlauben.

Mehr zur Entwicklungsgeschichte, zu den Prinzipien und zu der Verwendung der HLA sowie ein Anwendungsbeispiel findet man in [KWD99]. Im Rahmen des NPSNET-V-Projektes (siehe nächster Abschnitt) wurde die HLA zur *eXtensible Run-Time Infrastructure* (XRTI, [XRTI]) erweitert. XRTI ist eine Implementierung von HLA für zur Laufzeit erweiterbare DVEs mit zusätzlichem Augenmerk auf Zusammensetzbarkeit, Reflexivität, starker Typisierung und automatisch generierten *proxies* zur Abstraktion der Kommunikation.

### NPSNET

In den *NPSNET*-Systemen des MOVES Institute in Monterey [MOVES] wurden einige der beschriebenen Standards implementiert. So ist die Version NPSNET-IV beispielsweise ein DIS-Client in C++ für SGI-Workstations [Bru97].

Die letzte Version, *NPSNET-V* [NPSNET], war als eine Java-basierte Komponentenarchitektur für Anwendungen verteilter virtueller Umgebungen geplant. Dort wurde versucht, über die reine Standard-Implementierung hinaus einen Prototyp zur Einbettung und zum Testen neuer Technologien zu entwickeln. Hauptziele des Systementwurfs waren unter anderem Laufzeit-Erweiterbarkeit, Interoperabilität, Skalierbarkeit und in einem gewissen Maße auch die Sicherheit.

NPSNET-V war immer eher ein Projekt als ein Forschungsprototyp. Die Grundarchitektur wurde im Laufe der Zeit immer wieder von Grund auf verändert, sodass eine konsistente Beschreibung eines Systems nicht möglich ist. Mit der Vollendung von *XRTI* [Kap03] scheint die Arbeit des MOVES Institutes an NPSNET-V eingestellt worden zu sein, obwohl es immer noch als Open-Source-Projekt bei SourceForge registriert ist. Im Folgenden zitierte Komponenten von NPSNET-V stellen demzufolge eher Forschungsvorhaben innerhalb des Projektes dar als Komponenten, die etwa gleichzeitig in das System eingebunden werden könnten.

[2] *object model template*: Eine Schablone, aus der Objektmodelle erzeugt werden können.

### 2.2.3 Virtual-Reality-Systeme

Die *Virtual Reality Markup Language* (*VRML*, [VRML]) ermöglicht eine relativ einfache Verknüpfung von dreidimensionalen Szenen und anderen Web-Inhalten, und bietet Programmschnittstellen für Java und ECMA-Skript. Auf dieser Grundlage wurden daher mehrere Systeme geschaffen, um Teile des World-Wide-Web als virtuelles Universum (englisch *cyberspace*) zu modellieren, in dem sich die „Surfer“ bewegen und auch miteinander kommunizieren (englisch *chatten*) können. Zu diesen Systemen gehören z.B. VNET [VNET] und *Blaxxun Community* [Blx].

Beispiele für Virtual-Reality-Systeme aus dem universitären Bereich sind *Distributed Interactive Virtual Environment* (DIVE, [Dive]) am Swedish Institute of Computer Science, *MAnchester Virtual EnviRonment Interface Kernel* (MAVERIK, [Mav]) der University of Manchester, *Performance ARchitecture for Advanced Distributed Interactive Simulation Environments* (PARADISE, [Para]) der Stanford University.

Nach einer Menge Ärger mit Inkompatibilitäten zwischen verschiedenen *Browser-Plugins* wurde VRML im neuen Standard X3D [X3D] erweitert. Die beiden augenfälligsten Unterschiede sind eine stärkere Standardisierung und eine XML-basierte Syntax. Verteilte virtuelle Umgebungen auf Basis von X3D waren zum Zeitpunkt dieser Arbeit nicht zu finden.

## 2.3 Anforderungen

In diesem Abschnitt werden die Anforderungen an verteilte virtuelle Umgebungen formuliert und analysiert.

### 2.3.1 Immersion

Eine verteilte virtuelle Umgebung heißt *immersiv*[3], wenn ein Benutzer das Gefühl hat darin „einzutauchen“, also den Eindruck hat, sich in einer realen Szene zu befinden. Daraus ergeben sich eine Reihe von Anforderungen an das System.

Hat eine Entität ein reales Vorbild, so sollte sie dessen wesentliche Eigenschaften modellieren. Dazu gehören z.B. realistische Bewegungsmuster von Avataren, oder dass eine Lampe tatsächlich die Szene erhellt und über einen Schalter ein- und ausgeschaltet werden kann.

Physikalische Einflüsse müssen ebenfalls sorgfältig modelliert werden. In den meisten Szenen wirkt eine Gravitation, die Gegenstände nach unten fallen lässt, wenn sie losgelassen werden. Ein Avatar sollte außerdem auf dem Boden stehen, egal, wie hoch dieser an der aktuellen Stelle ist (englisch *terrain-following*). Kollisionen zwischen Entitäten sollten erkannt werden, um unnatürliche Durchdringungen auszuschließen (englisch *collision detection*).

Die in der Szene vorkommenden Entitäten müssen möglichst detailliert dargestellt werden. Zur Standardausstattung eines aktuellen Rechners gehört eine Grafikkarte mit 3D-Beschleunigung, die die Darstellung feiner Texturen, glatter Schattierungen und einer hohen Polygonzahl pro Sekunde ermöglicht. Der Mensch nimmt Einzelbilder erst ab einer Bildrate von ungefähr 25 Bildern pro Sekunde als Bewegung wahr. In einer immersiven Szene sollte die Darstellungsrate daher höher sein. Die reine Grafikleistung moderner Rechner stellt hier in der Regel noch keine Einschränkung dar. Da andere Prozesse nicht in dieser Geschwindigkeit laufen können, hat diese Anforderung gewisse Effekte auf den Entwurf, wie in Abschnitt 2.5.4 gezeigt wird.

[3] lat. immergere: eintauchen

Zu lange Antwortzeiten stören den realistischen Eindruck, den der Benutzer von der Szene haben soll.

Der Entwurf komplexer Entitäten und Szenen kann mit modernen Autoren-Werkzeugen wie Maya [Maya] oder 3D Studio MAX [3DS] unterstützt werden, stellt aber einen ernst zu nehmenden Kosten- und Zeitfaktor bei der Entwicklung einer dreidimensionalen Simulation dar.

Der Markt bietet außerdem eine Vielfalt multimedialer Hardware zur Verbesserung der Immersion. Anzeigegeräte wie *head-mounted displays*, Shutter-Brillen oder gar *caves* verbessern die dreidimensionale Darstellung. Soundkarten und Surround-Systeme können die Szene mit akustischen Elementen bereichern. Als Steuerungs- und Eingabegeräte kommen im Spielebereich z.B. Joysticks, Gamepads und Steuerräder infrage, während professionelle Anwendungen haptische Einheiten oder Datenhandschuhe benutzen.

Nur wenn alle Bereiche ausreichend berücksichtigt werden, bietet die Simulation Immersion. Je realistischer eine Szene ist, desto attraktiver ist sie für die Benutzer. Im Falle von Computerspielen wirkt sich das direkt auf den kommerziellen Erfolg aus.

### 2.3.2 Skalierbarkeit und Stabilität

Die Szene sollte eine große Anzahl komplexer Entitäten umfassen können, ohne dass die Immersion verloren geht. Die verfügbare lokale Grafikleistung ist jedoch immer beschränkt. Eine Vorauswahl der sichtbaren Entitäten sowie eine dynamische Anpassung des *Detaillierungsgrades* (englisch *level-of-detail*) können den Aufwand für die Darstellung konstant halten.

Bei einer großen Anzahl dynamischer Entitäten steigt außerdem die Netzwerklast. Bei vielen Systemen führt das zu einer Begrenzung der Anzahl der Teilnehmer auf eine Handvoll. Besonders ernst ist diese Einschränkung, wenn einige Teilnehmer über Modemverbindungen mit extrem niedriger Bandbreite und hoher Latenz an den Verbund angeschlossen sind. Innerhalb des Bereichs der angestrebten Teilnehmerzahl, der bei einigen DVEs einige hundert bis tausend umfasst, sollte weder die Stabilität noch die Leistung des Systems leiden.

Ein weiterer wichtiger Aspekt ist die *Ausfallsicherheit* (englisch *fail safety*). Es gibt viele Ursachen für den plötzlichen Ausfall eines Teilnehmers, z.B. Systemabsturz, temporäre Überlastung oder Ausfall des Netzwerks. Bei einer großen Teilnehmerzahl muss die Simulation stabil weiterlaufen, auch wenn einige (wenige) Teilnehmer ausfallen oder kurzzeitig nicht erreicht werden können.

Die Simulation sollte generell robust gegenüber Netzwerkproblemen sein. Nicht nur plötzlich unerreichbare Teilnehmer, sondern auch verlorene, beschädigte oder in falscher Reihenfolge eingehende Nachrichten sollten die Stabilität nicht gefährden. Der immense Nachrichtenverkehr bei einer großen Anzahl von Teilnehmern sollte so begrenzt werden, dass das Netzwerk nicht überlastet wird (englisch *congestion control*).

Zentrale Dienste, wie ein einzelner Simulations-Server, stellen oft einen *kritischen Ausfallpunkt* (englisch *single point of failure*) des gesamten Verbunds dar, denn dort konzentrieren sich sowohl der Netzwerkverkehr als auch die Rechenlast. Unvermeidbare zentrale Dienste sollten repliziert werden, um den Schaden eines Ausfalls in Grenzen zu halten. Siehe hierzu auch Abschnitt 3.1.1.

Mehrere hundert Rechner werden mit Sicherheit nicht in einer vorgegebenen Reihenfolge oder gar gleichzeitig starten. Das System sollte daher den *späteren Eintritt* (englisch *late-join*) und das *frühere Verlassen* (englisch *early-leave*) des Rechnerverbunds ermöglichen. Dieser Punkt ist eng verknüpft mit der Erweiterbarkeit zur Laufzeit, der in Abschnitt 2.3.4 behandelt wird.

Die Skalierbarkeit einer verteilten virtuellen Umgebung wird also vorwiegend von den Gegebenheiten des Netzwerks diktiert (vgl. [Bru97]). Wenn das System eine große Anzahl Teilnehmer und

dynamische Entitäten unterstützen soll, muss man den Netzwerkverkehr auf ein Minimum reduzieren.

### 2.3.3 Konsistenz

In einer verteilten virtuellen Umgebung wird der Zustand der Szene auf alle Teilnehmer repliziert. In diesem Abschnitt werden Konsistenzprobleme der verschiedenen Replica erläutert.

#### Kollisionserkennung

Einige Aktionen können sich gegenseitig bedingen oder ausschließen. Beispielsweise dürfen Avatare in vielen Simulationen einander nicht durchdringen, kann derselbe Gegenstand nicht von zwei verschiedenen Avataren gleichzeitig aufgesammelt werden, kann ein toter Spieler in einem *first-person-shooter* nicht mehr schießen oder darf eine von einem Avatar geschlossene Tür nicht von einem anderen durchschritten werden. Die entsprechenden Ereignisse müssen also auf Kollisionen geprüft und eventuell verändert oder gelöscht werden. Die Reihenfolge, in der Ereignisse ausgeführt werden, muss für alle Teilnehmer gleich sein. Außerdem müssen zur Kollisionserkennung alle Ereignisse vorliegen. Zuverlässige Kollisionsprüfung erfordert also Synchronisation (vgl. Abschnitt 2.3.3) und zuverlässigen Transport von Nachrichten (vgl. Abschnitt 3.2.3).

Kollisionsprüfung von Ereignissen kann in einer zentralen Architektur (vgl. Abschnitt 3.1.1) relativ einfach implementiert werden, da der Server die maßgebliche Simulationszeit diktieren kann, indem er eingehende Nachrichten nach Eingangszeitpunkt sortiert, und da nicht beim Server eingegangene Ereignisse einfach ignoriert werden können. Die strenge Client-Server-Architektur kommt jedoch, wie in Abschnitt 2.3.2 argumentiert wurde, für groß angelegte DVEs nicht infrage.

Zur Konfliktauflösung können feste Regeln formuliert werden, z.B. „Wenn mehrere Avatare nach einem Gegenstand greifen, bekommt ihn keiner“, oder Abstimmungsprotokolle benutzt werden. Häufig wird auch eine Konfliktentscheidungsinstanz eingeführt, wie der *referee* in DIS-Java-VRML [DJV]. In einer verteilten Architektur muss ein Konflikt gleichzeitig und eindeutig erkennbar sein, damit er aufgelöst werden kann.

Konfliktvermeidung kann außerdem durch *lockstep*-Protokolle erreicht werden [BL01]: Eine Aktion darf erst dann erfolgen, wenn alle infrage kommenden Teilnehmer eine Erlaubnis erteilt haben und keine kollidierenden Ereignisse vorliegen. Die hierzu notwendige zusätzliche Kommunikation verlängert die Antwortzeit.

#### Synchronisation

Für die Immersion eines DVEs ist eine gemeinsame Zeitskala von entscheidender Bedeutung. Gleichzeitig ausgelöste Ereignisse sollten möglichst auch zu gleichzeitig ausgeführten Aktionen führen, genauso wie hintereinander ausgelöste Ereignisse Aktionen in derselben Reihenfolge bewirken sollten.

In verteilten Architekturen müssen also die Teilnehmeruhren synchronisiert werden, was zusätzliche Kommunikation erfordert, und damit die Antwortzeit verlängert. Zum Uhrenabgleich kann z.B. das NTP-Protokoll [NTP] verwendet werden. Aufgrund der notorischen Ungenauigkeit von PC-Hardware-Uhren müssen die Uhren auch während der Laufzeit der Simulation abgeglichen werden. In NTP kann man zusätzlich die Uhrenabweichungen extrapolieren und somit den Kommunikationsaufwand im Laufe der Simulation minimieren.

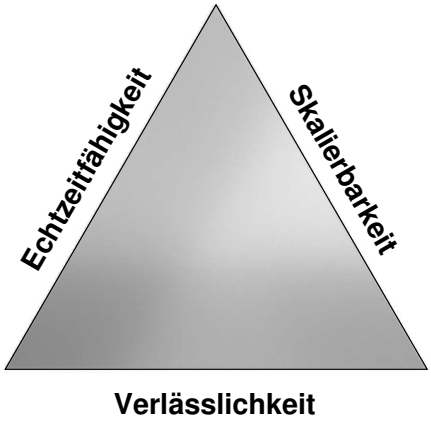



Abbildung 2.1: Kompromiss zwischen Verlässlichkeit, Echtzeitfähigkeit und Skalierbarkeit

### Qualität des Netzwerkdienstes

Die Qualität des Netzwerkdienstes (vgl. Abschnitt 3.2) beeinflusst die Konsistenz erheblich, insbesondere in einem heterogenen Netzwerk wie dem Internet. Beispielsweise stören unterschiedliche Netzwerklatenzen die Gleichzeitigkeit von Ereignissen, oder einzelne Nachrichten erreichen nicht alle Teilnehmer, weil sie aus Geschwindigkeitsgründen mit Protokollen verschickt werden, die keine Auslieferungsgarantie bieten. Die Qualität des Netzwerkdienstes wird daher ausführlich in Abschnitt 3.2 untersucht.

### Verlässlichkeit, Echtzeitfähigkeit und Skalierbarkeit

Das Netzwerkprotokoll zur Aufrechterhaltung der Konsistenz besitzt drei wesentliche Merkmale, von denen leider immer nur zwei gleichzeitig optimiert werden können (siehe Abbildung 2.1). Die Eigenschaften sind *Verlässlichkeit*, *Echtzeitfähigkeit* und *Skalierbarkeit*. Ein verlässliches Protokoll zwischen vielen Teilnehmern benötigt so viele Bestätigungsnachrichten, dass die Echtzeitfähigkeit gefährdet wird, und fordert man kurze Antwortzeiten, muss die Anzahl der Teilnehmer beschränkt werden. Andererseits können echtzeitfähige skalierbare Protokolle keine Garantien über die Auslieferung der Nachrichten an alle liefern.

Die Konsistenz der Simulation, gesichert durch ein verlässliches Protokoll, steht also in einem gewissen Sinne im Widerspruch zu der Immersion, die Echtzeitfähigkeit einschließt, und Skalierbarkeit. Singhal und Zyda nennen dieses Phänomen *performance consistency trade-off* [SZ99]. Maßnahmen zur Stärkung der Konsistenz führen zu Einbußen bei Skalierbarkeit oder Immersion und umgekehrt.

Viele Computerspiele bevorzugen starke Konsistenz und Immersion und beschränken infolgedessen die Anzahl der Spieler oder verwenden schlecht skalierende Client-Server-Architekturen.

Verteilte virtuelle Umgebungen können jedoch in vielen Anwendungen ein relativ hohes Maß an Inkonsistenz tolerieren. Es kommt beispielsweise für den Benutzer nicht darauf an, die exakten Zustände sämtlicher Entitäten zu kennen, solange er nicht mit ihnen interagiert.

Maßnahmen, die auf der einen Seite Konsistenz schwächen, aber auf der anderen Seite die Leistung erhöhen, sind z.B. sämtliche in Abschnitt 2.4 aufgezählte Techniken zur Verringerung der Netzwerklast. Zusätzlich sollte man die Ereignisse in zwei Typen einteilen: solche, die mit ande-

ren kollidieren können, und kollisionsfreie, die lokal sofort in Aktionen umgesetzt werden können. Je größer der Anteil an kollisionsfreien Ereignissen ist, desto weniger Leistungseinbußen sind zu erwarten. Wenn man beispielsweise, wie in vielen Virtual-Reality-Systemen, die gegenseitige Durchdringung von Avataren erlaubt, werden die Bewegungsereignisse der Avatare kollisionsfrei, und Benutzer können ihre Avatare fast verzögerungsfrei steuern.

### 2.3.4 Erweiterbarkeit

Einige verteilte virtuelle Umgebungen können zur Laufzeit erweitert werden (englisch *Runtime Extensible Distributed Virtual Environments*) (REDVEs). Dabei können Entitäten, die zu Beginn der Simulation noch nicht bekannt waren, in die Szene eingebracht werden, oder vorhandene Entitäten durch aktualisierte Versionen ersetzt werden. Dies wird erreicht, indem man die Entitäten als austauschbare Software-Komponenten (vgl. Abschnitt 2.5.5) mit festen Schnittstellen realisiert.

NPSNET definiert mit xfsp sogar ein erweiterbares Protokoll und ermöglicht damit das Einbringen neuer Ereignistypen zur Laufzeit. Andere VR-Systeme bieten den Benutzern die Möglichkeit, eigene Avatare zu generieren.

### 2.3.5 Portabilität

Der Verbund sollte aus heterogenen Teilnehmern bestehen können. Dazu müssen eindeutige Programmschnittstellen und Protokolle entwickelt werden. Will man das DVE nicht für jede Plattform neu implementieren, empfiehlt sich die Implementierung in einer portablen Programmiersprache, beispielsweise *Java* [Java].

Der plattformübergreifende Entwurf wird auch durch die Verwendung portabler Bibliotheken erleichtert. Grafikbibliotheken wie *OpenGL* [OGL], *Java3D* [J3D] oder *Direct3D* [DiX] bieten einheitliche Schnittstellen zu Grafik- und Soundkarten mit unterschiedlichen Fähigkeiten, während multimediale Ein- und Ausgabegeräte z.B. durch *DirectX* [DiX] oder *Delta3D* [Del3D] über eine einheitliche Schnittstelle angesprochen werden können.

Auch die Verwendung des Internets als Kommunikationsnetzwerk erhöht die Portabilität.

### 2.3.6 Persistenz

Der Zustand der Entitäten und der gesamten Szene sollte abgespeichert und zu einem späteren Zeitpunkt wiederhergestellt werden können. Einige der von einem den Verbund verlassenden Rechner kontrollierte Entitäten können von anderen Teilnehmern übernommen werden. Dazu muss der Zustand aller Entitäten serialisierbar sein. Zur Speicherung bieten sich Dateien oder Datenbanken an.

Bei einem Ausfall des Systems sollte der Datenverlust und somit die damit verbundenen Kosten begrenzt sein. Eventuell ist es daher sinnvoll, die Szene in regelmäßigen Abständen oder sogar jede Änderung der Szene abzuspeichern. Auch die Protokollierung von Ereignissen kann die Fehlersuche oder die Angriffsdetektion erleichtern.

## 2.4 Verringerung der Netzwerklast

Da die Netzwerkkommunikation einen wesentlichen Engpass einer verteilten virtuellen Umgebung darstellt, werden in diesem Abschnitt Techniken gezeigt, mit denen die Netzwerklast verringert

und damit Skalierbarkeit und Geschwindigkeit einer verteilten virtuellen Umgebung erhöht werden kann.

### 2.4.1 Positionsvorhersage und -korrektur

Wenn keine äußeren Kräfte auf einen Gegenstand wirken, bewegt er sich nach dem Trägheitsgesetz mit gleichförmiger Geschwindigkeit. Diese Tatsache macht man sich bei *dead reckoning* zu Nutze: Kennt man den Geschwindigkeitsvektor und die Position einer Entität zu einem bestimmten Zeitpunkt, *extrapoliert* man daraus die Position zu einem späteren Zeitpunkt unter der Annahme, dass keine Kräfte wirken. Erst wenn diese Vorhersage deutlich von der tatsächlichen Position der Entität abweicht, wird diese mit einer Netzwerknachricht *korrigiert*.

Wir sprechen in diesem Fall von *dead reckoning erster Ordnung*, da nur die Geschwindigkeit, also die erste Ableitung der Position, zur Extrapolation genutzt wird. Für die *zweite Ordnung* wird auch noch die Beschleunigung hinzugezogen. Das Prinzip lässt sich auch auf Rotation und Rotationsbeschleunigung verallgemeinern, ebenso wie auf Eingabeereignisse, z.B. zur Vorhersage der Position eines Joysticks.

Welche Art und Ordnung von *dead reckoning* sich eignet, hängt von den Eigenschaften einer Entität ab (vgl. [WP02]). Die Positions-Update-Nachrichten in DIS enthalten Informationen über Zeitpunkt, Position, Geschwindigkeit, Beschleunigung und Rotationsgeschwindigkeit, ebenso wie über den verwendeten Extrapolations- und Korrekturalgorithmus. Im PARADISE-System wird die neue Position allein aus den letzten bekannten Positionen berechnet (*position history-based dead reckoning*, [SC94]). Dadurch entfällt die Übertragung der Geschwindigkeit, was ebenfalls die Netzwerklast reduziert.

Wenn die Vorhersage korrigiert werden muss, werden oft Glättungstechniken eingesetzt, um sanft zwischen dem neuen maßgeblichen und dem vorhergesagten Zustand zu interpolieren. Dadurch werden sprunghafte Änderungen der Darstellung vermieden.

*Dead reckoning* kann ein sehr effizientes Mittel zur Nachrichtenreduktion sein, wenn Entitäten nur selten ihre Geschwindigkeit ändern. Es wird oft im Zusammenhang mit dem Master-Ghost-Entwurfsmuster (vgl 2.5.3) zur Vorhersage von Zuständen benutzt.

### 2.4.2 Interessensbereiche und Gebietszerlegung

Eine Jeder-mit-Jedem-Kommunikation zwischen mehreren hundert Teilnehmern ist in Echtzeit nicht möglich. Zum Glück ist aber auch nicht jeder Teilnehmer an allen Nachrichten interessiert. In der Regel genügt es, wenn er nur den für ihn relevanten Teil der Nachrichten empfängt und verarbeitet. In einer verteilten virtuellen Welt sind das die Updates der Entitäten, die sich im Blickfeld der Kamera bzw. des Avatars befinden. Diesen Bereich nennt man den *Interessensbereich* (englisch *area of interest*) des Teilnehmers, ein solches Filterungssystem heißt *Area Of Interest Management* (AOIM).

Da die Berechnung sich ständig ändernder Interessensbereiche schnell sehr aufwändig werden kann, benutzt man stattdessen oft *Gebietszerlegung*. Dabei wird die gesamte Szene in Teilszenen zerlegt, die voneinander unabhängig verarbeitet werden können. Jeder Teilnehmer bekommt dann alle Nachrichten aus dem Teilgebiet, in dem sich sein Avatar gerade befindet, und eventuell noch Informationen aus den direkt benachbarten Bereichen.

Gebietszerlegung wird oft bei der Kollisionsdetektion verwendet: Befinden sich $n$ Entitäten in einem Gebiet, so sind in einem Zeitinterval $n(n+1)/2$ Kollisionen möglich. Zerlegt man das Gebiet

in zwei voneinander unabhängige Teile, so müssen jeweils $(n^2/2+n)/2$ Kollisionen überprüft werden, also insgesamt $n^2/4$ Vergleiche weniger. Viele Szenen lassen sich sehr natürlich in disjunkte Gebiete zerlegen, z.B. in verschiedene Gebäude, Etagen oder Räume.

Weitere Möglichkeiten zeigt Oliveira im *Meta Interest Management* [Oli02]. Wathen stellt in [Wat01] ein dynamisches AOIM für NPSNET-V vor. Die Szene wird dabei zur Laufzeit so unterteilt, dass sich in jedem Teilgebiet zu jeder Zeit etwa gleich viele Entitäten befinden.

AOIM und Gebietszerlegung werden meist als Verleger-Abonnent-System (siehe 2.5.2) implementiert. Um nicht nur Verarbeitungszeit sondern auch Netzwerkbandbreite einzusparen, sollte das AOIM mit der Netzwerktopologie abgestimmt werden, beispielsweise durch ein Multicast-Socket pro Teilgebiet.

### 2.4.3 Verhaltensmuster

Große Zustandsänderungen äußern sich oft in einer Vielzahl vorhersehbarer kleiner Änderungen. Geht beispielsweise ein humanoider Avatar von einem Ort zu einem anderen, so bewegen sich seine Arme und Beine nach einem deterministischen Muster. Kennt ein Teilnehmer dieses Muster, kann aus der Nachricht „Avatar geht von A nach B“ dort lokal das komplette Bewegungsmuster rekonstruiert werden, das aus einer Vielzahl von Teilereignissen besteht. Eine solche Zusammenfassung von Ereignissen nennt man *Verhaltensmuster* (englisch *behaviour*). Typische Verhaltensmuster für Avatare sind beispielsweise „gehen“, „laufen“, „stehen“ oder „angreifen“.

Durch Verhaltensmuster kann die Anzahl der zu übertragenden Ereignisse erheblich reduziert werden. Allerdings steigt dabei der Rechenaufwand für die Teilnehmer.

## 2.5 Objektorientierte Entwurfsmuster

Wie in vielen anderen Bereichen der Computergrafik dominieren bei der Implementierung verteilter virtueller Umgebungen objektorientierte Programmiersprachen. Dabei haben sich auch einige typische *objektorientierte Entwurfsmuster* [GHJV95, BMR+96] bewährt. In diesem Abschnitt werden die wichtigsten davon dargestellt und ihre Anwendung in verteilten virtuellen Umgebungen erläutert.

### 2.5.1 Ereignisse und das Beobachter-Entwurfsmuster

Im *Beobachter-Entwurfsmuster* (englisch *observer pattern*) meldet sich ein Objekt, das über Zustandsänderungen einer Entität informiert werden möchte, als *Beobachter* (englisch *observer*) dieser Entität an. Jede Entität verwaltet eine Liste ihrer Beobachter. Im Falle einer Zustandsänderung erzeugt eine Entität ein *Ereignis-Objekt* (englisch *event*) und übermittelt dieses an alle Beobachter (englisch *event multicasting*). Die Beobachter können dann mithilfe der Informationen aus dem Ereignis-Objekt auf die Zustandsänderung reagieren.

Auf diese Art lässt sich einfach eine 1-zu-*n*-Abhängigkeit zwischen der Entität und den Beobachtern abbilden. Ein Beobachter ist selbst dafür verantwortlich, sich bei der Entität zu registrieren. Die Ereignis-Objekte können von den Beobachtern nicht verändert werden. Durch sie wird oft nur mitgeteilt, dass eine Zustandsänderung stattgefunden hat, aber nicht welche. Wünscht ein Beobachter detailliertere Informationen über die Zustandsänderung, fragt dieser sie anschließend durch einen Methodenaufruf bei der Entität ab.

### 2.5.2 Verleger-Abonnent-Entwurfsmuster

Im *Verleger-Abonnent-Entwurfsmuster* (englisch *publisher subscriber pattern*) informiert ein *Verleger* alle seine *Abonnenten* über Zustandsänderungen. In den meisten Büchern wird es mit dem Beobachter-Muster aus Abschnitt 2.5.1 gleichgesetzt. Die Funktionsweise ist tatsächlich fast identisch, nur dass es bei den Informationen, die der Verleger den Abonnenten anbietet, nicht ausschließlich um seinen eigenen Zustand gehen muss. Eine typische Anwendung dieses Entwurfsmusters ist das *area of interest management* aus Abschnitt 2.4.2: Dort sammelt der Verleger Ereignisse eines Interessenbereichs und verteilt diese an seine Abonnenten.

### 2.5.3 Master-Ghost- und Stellvertreter-Entwurfsmuster

Dynamische Entitäten werden in vielen Fällen in ein *Master-Objekt* und mehrere *Ghost-Objekte* aufgespalten. Das Master-Objekt ist nur auf dem Teilnehmer vorhanden, dem die Entität gehört, während die Entität auf den anderen Teilnehmern durch ein Ghost-Objekt repräsentiert wird. Die Ghost-Objekte sind passive Spiegelbilder des Master-Objekts, und empfangen dessen Zustandsänderungen über das Netzwerk.

Im Zusammenhang mit *dead reckoning* (siehe 2.4.1) besitzen die Ghost-Objekte einen Algorithmus zur Voraussage des Zustandes. Das Master-Objekt kapselt dann zusätzlich ein Ghost-Objekt, und vergleicht dessen vorhergesagten mit dem eigenen maßgeblichen Zustand. Überschreitet die Abweichung einen Schwellwert, wird eine Korrekturnachricht mit dem maßgeblichen Zustand an alle Ghost-Objekte verschickt. Diese passen dann ihren lokalen Zustand entsprechend an.

Die Ghost-Objekte können auch als *Stellvertreter* (englisch *proxy*) des Master-Objekts fungieren. Wird eine Methode, z.B. ein Verhaltensmuster, eines Ghost-Objekts aufgerufen, sendet dieses eine Anfrage an sein Master-Objekt, um die gewünschte Aktion durchzuführen. Dieses prüft zentral, ob die Berechtigung für den Aufruf der Methode vorliegt, bringt von unterschiedlichen Teilnehmern synchron durchgeführte Änderungen in eine einheitliche Reihenfolge und sendet die maßgeblichen Aktualisierungen an alle Ghost-Objekte zurück.

Da das Master-Objekt immer den maßgeblichen Zustand vorgibt, kann mit diesem Muster der Zustand der Entität auf allen Teilnehmern konsistent gehalten, oder im Zweifelsfall zentral über den tatsächlichen Zustand entschieden werden. Daher sollte jede Entität einen eindeutigen Master-Objekt-Besitzer haben, auch wenn sie von mehreren Teilnehmern verändert werden kann.

### 2.5.4 Modell-Sicht-Kontrolleur-Entwurfsmuster

Ein geeignetes Entwurfsmuster zur Modellierung der Entitäten ist das *Modell-Sicht-Kontrolleur-Muster* (englisch *Model-View-Controller pattern*) (MVC). Jede Entität wird in drei Objekte aufgespalten. Das *Modell-Objekt* kapselt den Zustand der Entität, das *Sicht-Objekt* sorgt für ihre Darstellung und das *Kontrolleur-Objekt* ändert ihren Zustand, z.B. aufgrund externer Ereignisse. Eine ausführliche Darstellung von MVC findet sich in [BMR+96]. Abbildung 2.2 zeigt die hier benutzte vereinfachte Variante.

Im einfachsten Fall ist das *Modell-Objekt* ein reines Datenobjekt, das ein Attribut für jede Zustandsvariable der Entität besitzt. Es kann in seinen Methoden aber auch Verhaltensmuster (vgl. Abschnitt 2.4.3) realisieren. Um Inkonsistenzen im Entitätszustand zu vermeiden, besitzt jeder Teilnehmer genau ein Modell-Objekt pro bekannter Entität.

Das *Sicht-Objekt* wird unter Verwendung des Beobachter-Musters über Zustandsänderungen der Entität informiert, und kann dann dessen Darstellung entsprechend anpassen. Da eine Entität meh-

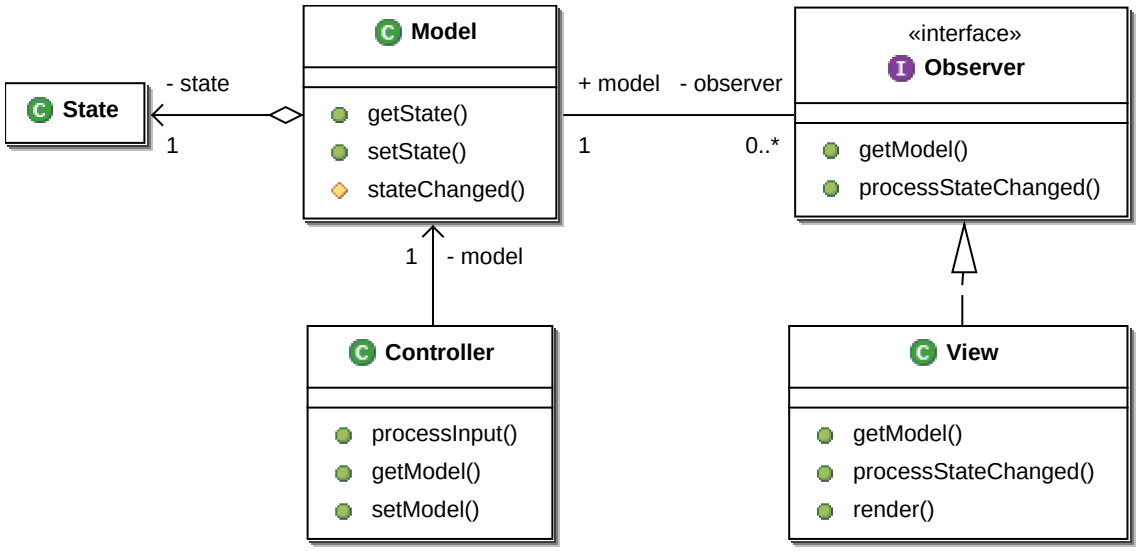



Abbildung 2.2: Klassendiagramm des *Model-View-Controller*-Entwurfsmusters

rere Sicht-Objekte besitzen darf, kann sie durch Austausch dieser Objekte leicht an verschiedene Grafikbibliotheken angepasst werden. Einige Entitäten benutzen auch mehrere verschiedene Sicht-Objekte gleichzeitig, z.B. wenn die aktuelle Geschwindigkeit eines Fahrzeugs auf einem Tachometer angezeigt werden soll.

*Kontrolleur-Objekte* empfangen Eingaben des Benutzers, Netzwerknachrichten oder andere externe Stimuli und setzen sie in Zustandsänderungen der Entität um. In einigen Systemen werden auch die Verhaltensmuster in Kontrolleur-Objekten gekapselt. In NPSNET-V können die Kontrolleur-Objekte einer Entität zur Laufzeit ausgetauscht werden, um das Netzwerkprotokoll dynamisch zu verändern. Eine Entität kann durch unterschiedliche Kontrolleur-Objekte über verschiedene Netzwerk-Protokolle kommunizieren oder durch die Eingaben unterschiedlicher Benutzerschnittstellen manipuliert werden. Falls ein Kontrolleur-Objekt eine lokale Kopie des Entitätszustands besitzt, kann es sich ebenfalls als Beobachter beim Modell-Objekt anmelden, um Änderungen, die beispielsweise von anderen Kontrolleur-Objekten verursacht wurden, zu erhalten. Manchmal werden Kontrolleur-Objekte auch zur Steuerung der Darstellung verwendet.

Neben der Anpassung an unterschiedliche Gegebenheiten des lokalen Rechners bietet die Aufspaltung einer Entität einen weiteren Vorteil: Graphische Darstellung, Eingabe und Zustand können gleichzeitig in unterschiedlichen Threads verarbeitet werden und werden dadurch entkoppelt. Eingabegeräte können nebenläufig abgefragt werden und verzögern nicht die Darstellungsrate. Die Bildrate hängt dann nicht von der Frequenz der eingehenden Netzwerknachrichten ab, was bewirkt, dass in einem heterogenen Verbund jeder Teilnehmer die Qualität der graphischen Darstellung an seine eigenen Kapazitäten anpassen kann. Die Skalierbarkeit des Systems wird also durch MVC verbessert, da der langsamste Teilnehmer die anderen nicht mehr bremst. Außerdem kann durch die Verwendung von Threads die Leistung eines Teilnehmers durch Einbau zusätzlicher Prozessoren verbessert werden.

### 2.5.5 Komponentenarchitektur

Über eine eindeutige Definition des Begriffs *Komponente* (englisch *component*) ist sich die Fachliteratur uneinig. Szyperski stellt einen Vergleich unterschiedlicher Definitionen in [Szy97] auf. Eine

eigene Definition würde auch den Rahmen dieser Arbeit sprengen. Da viele moderne verteilte virtuelle Umgebungen jedoch als *Komponentenarchitektur* entworfen sind, sollen hier die wichtigsten Eigenschaften von Komponenten in diesem Zusammenhang aufgeführt werden:

1. Komponenten sind größere logische Programmeinheiten mit vertraglich festgelegter Schnittstelle.

2. Komponenten sind zusammensetzbar und austauschbar.

3. Instanzen von Komponenten können serialisiert werden.

Der erste Punkt ist als Modularisierungsaspekt wichtig für die Entwicklung eines großen Systems. Der Entwickler einer einzelnen Produktfunktion muss nicht gleich das gesamte System kennen, sondern nur die entsprechende Komponente und deren Schnittstelle. Die Einarbeitungszeit wird dadurch erheblich verkürzt. Außerdem können Komponenten unabhängig vom gesamten System getestet werden.

Damit Komponenten zusammenarbeiten, müssen sie kombinierbar sein. Dazu werden sie über festgelegte Schnittstellen miteinander verbunden. Das System kann durch Austauschen von Komponenten mit der gleichen Schnittstelle verändert oder erweitert werden. So können Teile ausgetauscht werden, ohne die Stabilität des Ganzen infrage zu stellen. Für verteilte virtuelle Umgebungen ist insbesondere das Hinzufügen von Entitäten zur Laufzeit interessant (vgl. Abschnitt 2.3.4), die infolgedessen als Komponenten mit festgelegter Schnittstelle entworfen werden müssen.

Serialisierte Komponenten-Instanzen können persistent gespeichert oder über das Netzwerk verschickt werden. Somit können z.B. Entitäten zwischen den Teilnehmern wandern, bzw. auf einem neuen Teilnehmer installiert und initialisiert werden.

Bekannte Komponenten-Rahmenwerke sind *Java Beans* [JB] und *COM* [COM]. Komponenten-Rahmenwerken für verteilte virtuelle Umgebungen wurden beispielsweise im *Java Adaptive Dynamic Environment* [JADE] und in NPSNET-V [KMC02] propagiert.

## 2.6 Zusammenfassung

Verteilte virtuelle Umgebungen gehören zu den anspruchsvollsten Anwendungen unserer Zeit, da sie hohe Anforderungen an das Verbindungsnetzwerk und an die Rechenleistung der teilnehmenden Rechner stellen.

Verteilte virtuelle Umgebungen produzieren enormen Netzwerk-Nachrichtenverkehr mit vielen Sendern und vielen Empfängern. Die Nachrichten müssen möglichst schnell und effizient übertragen und möglichst in Echtzeit verarbeitet werden. Der Zustand der Szene wird auf alle Teilnehmer repliziert, wobei die Konsistenzanforderungen, verglichen mit anderen verteilten Systemen, abgeschwächt werden können. Lokale und externe Ereignisse müssen möglichst in Echtzeit verarbeitet werden. Daher werden oft Techniken eingesetzt, die die Leistung des Systems steigern, aber gleichzeitig die Konsistenz schwächen.

Der objektorientierte Ansatz im Entwurf von verteilten virtuellen Umgebungen hat sich bewährt. In vielen Implementierungen treten typische Entwurfsmuster wie das Modell-Sicht-Kontrolleur-Muster auf. Für erweiterbare DVEs sollten die Entitäten als einfach austauschbare Komponenten realisiert werden.

# Kapitel 3

# Kommunikation

Die Zustandsänderungen der Entitäten einer verteilten virtuellen Umgebung müssen zwischen den Teilnehmern kommuniziert werden. In diesem Kapitel werden die grundlegenden Aspekte der Kommunikation in DVEs beschrieben. In Abschnitt 3.1 werden die drei wichtigsten Kommunikationsparadigmen erläutert. Im darauf folgenden Abschnitt 3.2 werden die Qualitätsparameter einer Netzwerkverbindung untersucht. Die wichtigsten Internet-Protokolle werden im Abschnitt 3.3 behandelt. Schließlich wird in Abschnitt 3.4 eine neue speziell an die Erfordernisse verteilter virtueller Umgebungen angepasste Serialisierungs- und Kommunikationsbibliothek in Java vorgestellt.

## 3.1 Netzwerkparadigmen und Kommunikationsmodelle

Das *Netzwerkparadigma* einer verteilten virtuellen Umgebung beschreibt ihr Kommunikationsmuster und die Verteilung der Aufgaben und Verantwortlichkeiten unter den Teilnehmern. In diesem Abschnitt werden drei Paradigmen für verteilte virtuelle Umgebungen untersucht: *Client-Server*, *Peer-to-Peer* und *Client-Proxy-Server*.

### 3.1.1 Client-Server

In einer *Client-Server-Umgebung* kommunizieren viele *Clients* mit einem zentralen *Server*. Die Netzwerktopologie ist sternförmig, wobei der Server in der Mitte steht. Die Kommunikationsmuster sind „einer an viele“ für die Server-Client-Richtung, und „viele an einen“ für die Client-Server-Richtung. Der Server verwaltet in diesem Paradigma die Simulationszeit und den maßgebenden Zustand der Szene und prüft eingehende Statusveränderungen auf Gültigkeit und Kollisionen. Durch diese Zentralisierung kann der Zustand der Szene sehr einfach konsistent gehalten und vor unbefugten oder ungültigen Änderungen geschützt werden. Daher und wegen des relativ geringen Implementierungsaufwands dominiert das Client-Server-Paradigma in Anwendungen, die stark von *cheating* bedroht sind (vgl. Abschnitt 5.1.4).

Der Vorteil des Client-Server-Paradigmas, seine zentralisierte Architektur, ist gleichzeitig sein größter Nachteil: Clients können nicht direkt miteinander kommunizieren, sondern nur über den Server. Dieser Umweg führt in der Regel zu einer erheblichen Verlängerung der Latenzzeit. Durch die extreme Arbeitslast auf dem Server wird dieser schnell zum Flaschenhals, sowohl was Rechenleistung als auch Netzwerkverkehr angeht. Reine Client-Server-Architekturen skalieren daher nicht gut.

Erhalten alle Clients dieselben Nachrichten vom Server, sollte nach Möglichkeit *Multicasting*

zur Kommunikation verwendet werden. Je mehr sich die Nachrichten an die einzelnen Clients unterscheiden, umso mehr bietet sich *Unicasting* an. Clients senden ihrerseits ihre Nachrichten per Unicast an den Server, falls keine direkte Client-Client-Kommunikation erwünscht ist, ansonsten kann hier ebenfalls Multicasting zum Einsatz kommen. Stehen genügend Netzwerkbandbreite und Rechenleistung zur Verfügung, kann in beiden Fällen auch eine große Nachricht per Multicast gesendet werden, die mehrere unterschiedlich verschlüsselte Teilnachrichten enthält, die jeweils nur von bestimmten Empfängern entschlüsselt werden können.

### 3.1.2 Peer-to-Peer

Das *Peer-to-Peer-Paradigma* stellt das Gegenteil zum Client-Server-Paradigma dar: Alle Teilnehmer sind *gleichrangig* (englisch *peers*). Es gibt keine zentralisierten Dienste mehr. Jeder Teilnehmer hat dieselben Pflichten und Rechte. Das dominierende Kommunikationsmuster ist „viele an viele“, und Nachrichten werden oft mittels IP-Multicasting übermittelt.

Ein dezentrales Peer-to-Peer-System skaliert zwar besser als ein zentralisiertes System, doch gibt es auch hier klare Grenzen. Da eine zentrale Instanz fehlt, ist die Reihenfolge von Ereignissen schwierig zu bestimmen. Die Synchronisation der Peers, und damit auch die Erkennung und Auflösung von Konflikten, erfordert zusätzliche Netzwerknachrichten. Abstimmungen über maßgebliche Entscheidungen, z.B. den Ausschluss eines Teilnehmers, benötigen ebenfalls zusätzliche Kommunikation. Oft diktiert dabei die langsamste Verbindung die Wartezeit für alle Teilnehmer. Im schlimmsten Fall erzielen die Teilnehmer keinen Konsens über den Zustand der Szene, und die Simulation muss abgebrochen werden.

Um sich bei einem Peer-to-Peer-System anzumelden, benötigt ein neuer Benutzer einen *Zugangspunkt* (englisch *rendezvous point*). Dies kann ein Rechner des Verbunds, ein Verwaltungs-Server oder die Adresse eines Multicast-Sockets sein.

Wegen der Abstimmungs- und Synchronisierungsprobleme findet man das Peer-to-Peer-Paradigma hauptsächlich in kollaborativen Anwendungen oder in Open-Source-Systemen, deren Benutzer oft aus politischen Gründen gegen eine autoritäre Regelung sind. Da die Teilnehmer in Peer-to-Peer-Systemen sogar anonym bleiben können, sind diese sehr populär für Tauschbörsen.

Meist werden reine Peer-to-Peer-Anwendungen durch zentrale Dienste, wie Anmeldungs-Server oder Verzeichnisdienste zum Lokalisieren von Teilnehmern und Ressourcen, erweitert. Um die Skalierbarkeit damit nicht zu verschlechtern, sollten zentrale Dienste replizierbar sein.

### 3.1.3 Client-Proxy-Server und Distributed-Server

Das *Client-Proxy-Server-Paradigma* vereinigt die gute Kontrolle von Client-Server-Architekturen mit der besseren Skalierbarkeit von Peer-to-Peer-Systemen. Clients kommunizieren hier nicht mehr direkt mit dem Server, sondern über einen von vielen *Stellvertretern* (englisch *proxies*). Jeder Proxy verwaltet und kontrolliert mehrere Clients, und leitet gültige und soweit möglich konfliktbereinigte Nachrichten an den Server bzw. an seine Clients weiter.

Meist kommunizieren die Proxies zusätzlich untereinander, um z.B. die Zuständigkeit für einen Client zu delegieren. Wenn keine weitere Synchronisation zwischen den Proxies notwendig ist, kann auf den Server verzichtet werden, und die Proxies werden in den Rang von Servern erhoben. Das entstehende Paradigma nennt man *Distributed-Server-Paradigma*. Dieses Paradigma findet man oft in Verbindung mit Interessenbereichs-Management, wobei jeder Server einen Bereich maßgeblich verwaltet, und Clients, die diesen Bereich verlassen, an den verwaltenden Server des

neuen Bereichs delegiert werden.

Das Client-Proxy-Server-Paradigma für Computerspiele wird in [MFW02] und in weiteren Beiträgen aus demselben Konferenzband beschrieben.

## 3.2 Qualität des Netzwerkdienstes

Die *Qualität des Netzwerkdienstes* (englisch *Quality of Service*) (QoS) einer Netzwerkverbindung legt fest, für welche Zwecke diese Verbindung eingesetzt werden kann. In diesem Abschnitt wird der Einfluss der einzelnen Eigenschaften auf die Funktion von DVEs untersucht.

### 3.2.1 Latenz

*Latenz* ist der allgemeine Begriff für die Zeitdifferenz zwischen dem Auslösen eines Ereignisses und der Ausführung der zugehörigen Aktion. Werden die Ereignisse als Netzwerknachrichten verschickt, so berechnet sich die Gesamtlatenz als Summe der Einzellatenzen der folgenden Prozesse:

**Detektion:** Die Auslösung eines Ereignisses wird erkannt, z.B. wird ein Mausklick oder eine Kollision registriert. Die Detektionszeit ist besonders lang, wenn Eingabegeräte im Polling-Verfahren abgefragt werden.

**Senderseitige Verarbeitung:** Das Eingabeereignis wird auf dem Computer, auf dem das Ereignis ausgelöst wurde, instanziiert und verarbeitet.

**Serialisierung:** Das Ereignis wird in binäre Form gebracht und als Netzwerknachricht verpackt.

**Übertragung:** Die Nachricht wird über das Netzwerk vom Sender zum Empfänger transportiert.

**Deserialisierung:** Das Ereignis wird auf der Empfängerseite aus der Netzwerknachricht wiederhergestellt.

**Empfänger-Verarbeitung:** Das Ereignis wird auf der Empfängerseite weiterverarbeitet.

Ein DVE-Entwickler kann viele dieser Prozesse und damit die entstehenden Latenzen optimieren. Die Übertragungslatenz hängt allerdings von äußeren Gegebenheiten des Netzwerks ab, die im Folgenden näher untersucht werden.

Das physikalische Minimum der Übertragungszeit einer Information wird durch die Lichtgeschwindigkeit definiert. Die Überlandverbindung zwischen zwei maximal entfernten Punkten auf der Erde hat bei Lichtübertragung im Vakuum somit bereits eine Mindestlatenz von ca. 70ms. In einem Glasfaserkabel sind es bereits etwa 100ms, im Kupferkabel ca. 200ms. Diese theoretischen Minimalwerte können niemals unterschritten werden.

Weitere Latenzen entstehen durch Verstärker und *Repeater* die in regelmäßigen vom Übertragungsmedium abhängigen Abständen das Signal verstärken bzw. weiterleiten müssen. Auch Internet-Router bearbeiten jedes Paket und erhöhen dadurch die Latenzzeit. Besonders stark fallen Signalwandlungen, beispielsweise analog/digital in einem Modem, sowie Zwischenspeicherung von Informationen ins Gewicht.

Um ein Gefühl für die Größenordnung der einzelnen Anteile der zu erwartenden Übertragungslatenzen zu bekommen, wurde der WWW-Server der TUHH stichprobenartig von verschiedenen Rechnern aus mit dem Unix-Dienstprogramm traceroute angewählt. Dieses listet alle Router, die

Tabelle 3.1: *Round trip times* und Bandbreite für typische Internetanbindungen

| *Verbindungstyp* | *RTT zum Provider*[*ms*] | *Bandbreite* [*kbit/s*] *(upstream/downstream)* |
|---|---|---|
| *Analogmodem* | $> 110$ | 33.6 / 56.6 |
| *ADSL* | 40-70 | 128 / 768 |
| *ISDN* | 21-34 | 64 |
| *ADSL Fastpath* | 5-10 | 128 / 768 |
| *Kabelmodem* | 5-8 | 128 / 1500 |
| *SDSL QSC 2* | 5-11 | 3000 |
| *WLAN* | 5 | 11 000 |
| *Local Ethernet* | $< 1$ | 100 000 |

ein Datenpaket auf dem Weg zu seinem Zielort passiert, inklusive der *Round Trip Times* (RTT), also der Übertragungszeit zum jeweiligen Router und wieder zurück, auf. Einige Router-Namen gaben einen deutlichen Hinweis auf ihren Standort, sodass sich das Verhältnis von gemessener Latenz und geographischer Distanz berechnen ließ. Dies war im Internet-Backbone erstaunlich gut. Ein deutschlandweiter Bogen von Hamburg über Berlin, Dresden, Leipzig, Nürnberg, Frankfurt am Main zurück nach Hamburg (Luftlinie ca. 1300km) dauerte beispielsweise nur 20ms. Auch ansonsten lagen die Werte zwischen 30 und 45% der Lichtgeschwindigkeit im Vakuum. Deutlich schlechter ist das Verhältnis auf dem Teilstück vom Startrechner zu seinem lokalen Internet-Provider. Die Latenzen dieser Verbindung werden durch die eingesetzte Technologie diktiert.

Die verbindungsabhängigen Ergebnisse sind, ergänzt mit Resultaten aus [Che], in Tabelle 3.1 zusammengefasst. Die hohen Latenzen bei der Übertragung per Analogmodem entstehen in der Hauptsache bei der Wandlung des Signals von digital nach analog der Senderseite und der Rückwandlung von analog nach digital beim Empfänger. Diese Wandlung entfällt beim digitalen ISDN. ADSL Fastpath, Kabelmodem und SDSL haben in etwa die gleichen niedrigen Latenzen. Noch besser ist auch wegen der Luftübertragung WLAN, welches nur noch von der direkten Ethernet-Verbindung unterboten wird.

ADSL schneidet in der Standardkonfiguration erstaunlicherweise schlechter ab als das einige Jahre ältere ISDN. Der Grund liegt in der Verwendung von *interleaving* zur Fehlerkorrektur. Dabei werden mehrere hintereinander gesendete Pakete zur Übertragung ineinander geschachtelt. Da dazu mehrere Pakete vorliegen müssen, entstehen zusätzliche Verzögerungen. Die Anzahl der ineinander geschachtelten Pakete nennt man die *Interleaving-Tiefe*. Dieser Parameter ist im DSL-Protokoll konfigurierbar, wird aber meist vom Internet-Provider vorgegeben. Die Übertragung ohne Interleaving nennt man *Fastpath*. Viele Provider bieten Fastpath gegen Aufpreis als Option speziell für Online-Spieler an, die besonderen Wert auf geringe Latenzen legen.

### 3.2.2 Bandbreite und Verstopfung

Die Bandbreite einer festen Kommunikationsverbindung ist immer nur maximal so groß wie die der kürzesten Teilverbindung. Da die meisten Verbindungen im Internet nicht exklusiv nutzbar sind, schwankt die tatsächliche Bandbreite mit der weiteren Auslastung der Verbindung. Eine feste Bandbreite kann nur in einer exklusiv genutzten Verbindung garantiert werden, oder wenn ein bestimmter Anteil reserviert worden ist, z.B. mittels DiffServ [DS]. Zur Drucklegung dieser Arbeit waren solche Reservierungsdienste jedoch für Endverbraucher noch nicht verbreitet. Die Bandbrei-

ten der typischen Verbindungsendstücke sind ebenfalls in Tabelle 3.1 aufgeführt.

Wird versucht, über eine Datenleitung mehr zu senden als die maximale Bandbreite zulässt, tritt *Verstopfung* (englisch *congestion*) auf. Wenn ein Netzwerkrouter überlastet ist, nimmt er keine weiteren Nachrichten an, und oft gehen dadurch Nachrichten verloren. Da solche Fehler häufig dann auftreten, wenn große Datenmengen übertragen werden, gehen dann oft viele Nachrichten auf einmal verloren (englisch *burst drop*).

### 3.2.3 Verlässlichkeit

Durch Ausfälle und Störungen auf dem Übertragungsweg kann eine Nachricht verfälscht werden oder gar verloren gehen. Viele Protokolle besitzen daher die Möglichkeit, defekte oder verloren gegangene Teile einer Nachricht zu detektieren oder gar kleinere Fehler zu korrigieren. Im Folgenden werden kurz die dabei eingesetzten Verlässlichkeitsmechanismen umrissen.

Zur Detektion verfälschter Nachrichten dienen beispielsweise *Cyclic Redundancy Codes* (CRCs), Hashwerte (vgl. Abschnitt A.2), *Message Authentication Codes* (A.6.1) oder digitale Signaturen (A.6.2). Alle Detektionsmechanismen erhöhen das Kommunikationsvolumen, da zusätzliche Informationen übertragen werden müssen. Auch die Übertragungslatenz steigt an, denn auf beiden Seiten ist eine zusätzliche Bearbeitung jeder Nachricht notwendig.

Kleinere Übertragungsfehler können durch *Vorwärts-Fehlerkorrektur* (englisch *Forward Error Correction*) (FEC) auf der Empfängerseite korrigiert werden. Dabei werden in jeder Nachricht redundante Informationen mitgesendet, aus denen durch Verwendung festgelegter Algorithmen falsche Bits detektiert und korrigiert werden können.

Nicht wiederherstellbare oder als verloren detektierte Nachrichten kann der Empfänger vom Sender neu anfordern. Dazu sind zusätzliche Nachrichten erforderlich. Man unterscheidet grundsätzlich zwischen *positiver Bestätigung* (englisch *ACKnowledgement*) (ACK) und *negativer Bestätigung* (englisch *Negative ACKnowledgement*) (NACK). Der Empfänger bestätigt also entweder den Eingang jeder Nachricht (ACK), oder fordert die erneute Sendung einer verloren gegangenen Nachricht (NACK). Beide Ansätze haben ihre Vor- und Nachteile. ACKs steigern das Kommunikationsvolumen und sind in Broadcast- oder Multicast-Anwendungen, in denen der Sender die Empfänger nicht explizit kennt, nicht einsetzbar. NACKs produzieren in einem in der Regel fehlerfreien Netzwerk ein viel geringeres zusätzliches Nachrichtenvolumen und funktionieren auch mit unbekannten Empfängern. Sie geben dem Sender allerdings keine Garantie, dass alle Empfänger die Nachricht erhalten haben, denn diese könnten den eventuellen Verlust gar nicht bemerkt haben. Des Weiteren besteht, speziell im Broadcast- oder Multicast-Fall, die Gefahr der so genannten *NACK-Implosion*: Bleibt eine Nachricht in einer überlasteten Verbindung hängen, senden alle dahinter liegenden Empfänger gleichzeitig NACK-Nachrichten, sodass sich die Verbindung nicht erholen kann oder der Sender überflutet wird. Verlässliche Multicast-Protokolle bedienen sich daher ausgefeilterer Methoden zur Bestätigung.

### 3.2.4 Paket- und verbindungsbasierte Kommunikation

Kommunikationsverbindungen können *paketbasiert* oder *verbindungsbasiert* sein. Paketbasierte Verbindungen übermitteln einzelne Pakete, auch *Datagramme* genannt, die die Adresse des Empfängers tragen, und sind damit vergleichbar mit der konventionellen Kommunikation per Post. Verbindungsbasierte Verbindungen etablieren eine feste Verbindung zwischen den Kommunizierenden, auf der Daten in einem Datenstrom ausgetauscht werden können, vergleichbar mit dem

konventionellen Telefon. Da die hier behandelten Protokolle alle auf dem paketbasierten IP-Protokoll aufsetzen, stellen die verbindungsbasierten Protokolle immer einen Mehraufwand dar.

In der paketbasierten Kommunikation ist weiterhin eine *maximale Paketgröße* (englisch *Maximum Transfer Unit*) (MTU) zu beachten. Wird sie überschritten, müssen Nachrichten *fragmentiert*, also in mehrere Pakete aufgeteilt werden. Soll Fragmentierung vermieden werden, so dürfen die Pakete nie größer als das Minimum der MTUs aller Verbindungsteilstücke sein. Fragmentierung kann dazu führen, dass Fragmente alter Nachrichten den Eingangspuffer einer Verbindung blockieren und erst nach einem Timeout entfernt werden. Ist die Verbindung unzuverlässig, können verloren gegangene Fragmente nicht neu angefordert werden und bewirken den Verlust der gesamten Nachricht.

### 3.2.5 Blockierende und nicht-blockierende Kommunikation

In dieser Arbeit unterscheiden wir grob zwischen *blockierender* und *nicht-blockierender* Kommunikation.[1] In der blockierenden Kommunikation synchronisieren sich Sender und Empfänger bei jedem Nachrichtenaustausch, und setzen den Programmablauf erst nach abgeschlossener Übertragung fort. Aufgrund dieses einfachen Kommunikationsmodells kann im darauf folgende Programm davon ausgegangen werden, dass die vollständige Nachricht fehlerfrei übermittelt wurde. Werden mehrere Kommunikationskanäle auf einmal bedient, oder müssen bei der Verarbeitung weitere aufwändige Operationen durchgeführt werden, führt dieses Muster allerdings schnell zu Leistungsengpässen, da beide Kommunikationsteilnehmer viel Zeit mit Warten verbringen. Um dies zu vermeiden, kann pro Kanal ein eigener Thread eingesetzt werden, der, solange es nichts zu senden oder empfangen gibt, automatisch ausgesetzt wird. Leider skalieren auch Anwendungen mit beliebig vielen Threads nicht, da das Umschalten zwischen verschiedenen Threads eine konstante Zeit benötigt.

In der nicht-blockierenden Kommunikation läuft die eigentliche Übertragung im Hintergrund ab. Oft werden Nachrichten, beispielsweise vom Betriebssystem, in Zwischenspeichern abgelegt und können zu gegebener Zeit übertragen bzw. empfangen werden. Ein Empfänger kann dann abfragen, ob in einem der Kommunikationskanäle eine Nachricht eingegangen ist (englisch *polling*), und falls nicht die Abarbeitung anderer Operationen vorziehen. In einer anderen Variante können mit *Selektoren* mehrere Kanäle gebündelt werden. Der Sender oder Empfänger erhält auf Anfrage vom Selektor eine Liste aller registrierten und zur Zeit lesbaren bzw. beschreibbaren Kanäle und kann diese entsprechend bearbeiten. Auf dieser Grundlage können hochskalierbare Server-Anwendungen implementiert werden, da mit wenigen Threads viele Kommunikationskanäle auf einmal bedient werden können. Die Programmierung mit nicht-blockierender Kommunikation ist allerdings komplexer als mit blockierender Kommunikation.

## 3.3 Internet-Protokoll-Kommunikation

Das *Internet Protocol* (IP, [IP]) bildet die Grundlage für alle hier behandelten Protokolle. Es abstrahiert die tatsächliche physikalische Übertragung und bildet so eine gemeinsame Schnittstelle für völlig verschiedene Netzwerktechnologien. IP ist *paketorientiert* und gibt keine Garantien über die tatsächliche Auslieferung von Datagrammen (englisch *best effort delivery*). Da jedes Datagramm über einen unterschiedlichen Weg übertragen werden kann, wird auch nicht garantiert, dass die Pa-

[1] Feinere Abstufungen, wie man sie beispielsweise im MPI-Standard [MPI2] findet, sind hier nicht notwendig.

kete in derselben Reihenfolge empfangen werden, in der sie gesendet wurden (englisch *unordered delivery*).

Ein Rechner, der mittels IP kommunizieren möchte, benötigt eine IP-Adresse. Adressen können über den *Domain Name Service* (DNS) an für Menschen besser handhabbare Namen gebunden werden. Ein Datagramm wird an eine IP-Adresse und einen *Port* adressiert. Dadurch können Rechner verschiedene aktive Netzwerkanwendungen mit unterschiedlichen Ports gleichzeitig betreiben. Die Kombination von IP-Adresse und Port bezeichnet man auch als *Socket-Adresse*.

IP-Netzwerke sind hierarchisch aufgebaut. Das *Local Area Network* (LAN) umfasst alle Rechner, die, ohne dass ein Gateway passiert wird, direkt erreicht werden können.

Große Datagramme werden fragmentiert, also in kleine Stücke zerlegt, die einzeln versendet werden. Die MTU eines Übertragungspfads über mehrere Router ist das Minimum der Teil-MTUs. Die MTU hängt von der verwendeten Verbindungstechnologie ab. Für reine Ethernetverbindungen ist die MTU 1500 Byte, während ADSL (PPPoE) sie wegen zusätzlicher Header auf 1492 Byte reduziert. Verbindungen über GPRS bieten eine wesentlich kleinere MTU von 512 Byte. Mit dem UNIX-Programm tracepath kann man die MTU einer Verbindung ermitteln, falls die benutzten Router dies zulassen.

Obwohl die neue Version 6 des IP-Protokolls seit Jahren in den Startlöchern steht, operieren die meisten Netzwerke immer noch mit der Version 4.

### 3.3.1 IP-Broadcasting

Als *IP-Broadcasting* bezeichnet man den Versand von IP-Nachrichten an alle Rechner des LAN. Broadcasting findet generell über die Adresse 255.255.255.255 statt. Router blocken alle Nachrichten an diese Adresse ab. Broadcasting ermöglicht effiziente Einer-an-Alle-Kommunikation im LAN, da eine Nachricht nur einmal gesendet werden muss, um von allen Rechnern empfangen zu werden. Es wird oft zum Auffinden von anderen Rechnern oder Druckern im Netzwerk benutzt. In der Regel werden alle empfangenen Broadcast-Nachrichten vom Betriebssystemkern der angeschlossenen Rechner verarbeitet. Wenn Nachrichten nur an einen Teil der Rechner im LAN gesendet werden sollen, sollte man daher andere Kommunikationsmöglichkeiten benutzen.

### 3.3.2 IP-Multicasting

Der Adressraum von 224.0.0.0 bis 239.255.255.255 ist für *IP-Multicasting* reserviert. Multicasting ist eine sehr effiziente Methode zur Viele-an-Viele-Kommunikation.

Um IP-Multicast-Nachrichten zu empfangen, muss sich der Empfänger bei seinem Gateway an ein IP-Multicast-Socket anmelden. Dieser registriert den Empfänger bei allen direkt erreichbaren Routern, die ihrerseits wieder alle erreichbaren Router kontaktieren. Um eine Überschwemmung des gesamten Internets zu vermeiden, wird bei der Registrierung ein *Time-To-Live-Argument* (TTL) übergeben, das die maximale Anzahl der Weitergabe an die Router begrenzt. Am Ende der Anmeldung ist ein Multicast-Übertragungsbaum etabliert. Nachrichten, die an das entsprechende IP-Multicast-Socket gesendet werden, werden von den Routern an alle am Socket angemeldeten Teilnehmer weitergeleitet, und dabei nur an lokalen Verzweigungen des Baums kopiert.

Zur Etablierung des Übertragungsbaums existieren mehrere Protokolle, z.B. *Core Based Tree* (CBT), *Distance Vector Multicast Routing Protocol* (DVMRP), *Protocol Independent Multicast* (PIM), *Multicast extensions to OSPF* (MOSPF) oder *Simple Multicast Routing Protocol* (SMRP). Eine genauere Beschreibung der unterschiedlichen Protokolle kann man beispielsweise in [Kos98]

oder [WZ99] nachlesen.

Eine Multicast-Nachricht muss also nur einmal gesendet werden und minimiert die für die Übertragung benötigte Bandbreite. Im Gegensatz zum Broadcasting dringen IP-Multicast-Nachrichten an nicht-registrierte Ports nicht bis in den Betriebssystemkern vor, und, wenn der Gateway das zulässt, können auch Rechner außerhalb des LAN erreicht werden. IP-Multicasting ist außerdem empfängerbasiert, das heißt, ein Sender besitzt keine Informationen über die am Socket registrierten Empfänger. Dadurch wird der Sender von der Verwaltung der Multicast-Gruppe befreit, was die Skalierbarkeit weiter erhöht.

Leider ist die Unterstützung von IP-Multicasting nicht sehr weit verbreitet. Verantwortlich für die schlechte Verbreitung ist laut Diot et al. [DLL+00] unter anderem, dass IP-Multicasting nicht auf konkrete kommerzielle Anwendungen hin entworfen wurde. Ihre wesentlichen Kritikpunkte sind:

**Gruppenverwaltung:** Es sind keine Autorisierungsmechanismen für die Erzeugung von Multicast-Gruppen, für den Sender oder für die Empfänger vorhanden.

**Multicast-Adressen-Vergabe:** Es sind keine Mechanismen zur Koordination der Adressen-Vergabe vorgesehen, sodass Kollisionen mit anderen Anwendungen auftreten können.

**Sicherheit:** Sowohl das Routing-Protokoll als auch die Multicast-Session ist einfach angreifbar. Es gibt keine Integritätsgarantie für Nachrichten.

**Netzwerk-Management:** Werkzeuge, die einem Netzwerk-Provider die Verwaltung von Multicast-Verbindungen erleichtern, sind angesichts der vielen Routing-Protokolle nicht sehr verbreitet.

Des Weiteren bietet IP-Multicasting keine Verstopfungskontrolle (vgl. Abschnitt 3.2.2). Viele Internet-Provider stellen ihren Kunden den IP-Multicast-Dienst daher gar nicht erst zur Verfügung.

Diese Defizite und die daraus resultierende geringe Verfügbarkeit von IP-Multicasting haben zur Entwicklung einer Vielzahl unterschiedlicher Multicast-Implementierungen auf der Anwendungsschicht geführt, die in Abschnitt 3.3.5 behandelt werden.

### 3.3.3 User Datagram Protocol

Das *User Datagram Protocol* (UDP, [UDP]) bietet Anwendungen einen beinahe direkten Zugang zum IP-Protokoll. Wie IP ist UDP *paketorientiert* und offeriert keine Auslieferungsgarantie oder Ordnung der Datagramme. UDP-Datagramme werden außerdem beim IP-Multicasting eingesetzt.

Da die Adresse in jedem Datagramm enthalten ist, können über dasselbe UDP-Socket Nachrichten an verschiedene Rechner gesendet bzw. von diesen empfangen werden. Ein einziges Socket reicht damit für eine Anwendung meist aus.

UDP eignet sich besonders für den Versand von Nachrichten, bei denen niedrige Latenz erforderlich ist. Dazu gehören auch die Zustands-Updates von DVE-Entitäten. Unter abgeschwächten Konsistenzforderungen stört auch das Ausbleiben eines Updates nur wenig, wenn das nächste Update schnell eintrifft und das vorherige überflüssig macht. Die Verwaltung von UDP-Sockets erfordert sehr wenig Aufwand. Fragmentierung sollte allerdings vermieden werden, da verloren gegangene Fragmente nicht automatisch erneut gesendet werden.

### 3.3.4 Transmission Control Protocol

Das *Transmission Control Protocol/Internet Protocol* (TCP/IP) [TCP] setzt wie UDP auf dem IP-Protokoll auf. Allerdings sind TCP/IP-Sockets *verbindungsorientiert*, und bieten eine *Datenstromabstraktion* der Verbindung. Für jeden Kommunikationspartner muss also ein eigenes Socket geöffnet werden. Da Netzwerkdienste meist feste Portnummern haben, ist hierzu ein *TCP handshake* notwendig, in dem individuelle Sockets auf Sender- bzw. Empfängerseite verhandelt werden und deren Verbindung etabliert wird.

Fragmentierung und Wiederzusammensetzung der Nachrichten erfolgt automatisch. TCP/IP ist zudem verlässlich, d.h. jedes empfangene Fragment wird vom Empfänger bestätigt (ACK). Nicht bestätigte Fragmente werden nach Ablauf einer Wartezeit neu gesendet. Um hier Latenzen zu vermeiden, benutzt TCP/IP ein *sliding window* und überdeckt damit die Wartezeit auf Bestätigungen mit der Übertragung weiterer Fragmente. Das TCP/IP-Protokoll ist also zustandsbehaftet. Weiterhin garantiert TCP/IP den Empfang der Nachrichten in derselben Reihenfolge, in der sie gesendet wurden.

Um Verstopfung zu vermeiden, benutzt TCP/IP einen *langsamen Start* (englisch *slow start*): Die Rate, mit der die Pakete gesendet werden, wird so lange erhöht, bis die ersten Paketverluste auftreten. So passt sich TCP/IP sehr gut an die aktuellen Netzwerkbedingungen an.

Die Übertragungslatenz von TCP/IP ist wegen der Bestätigungsnachrichten ein wenig höher als im UDP-Protokoll. TCP/IP eignet sich daher hervorragend zur Übertragung größerer Nachrichten, bei denen die Latenz keine Rolle spielt, also z.B. für die Übertragung von Dateien, oder wenn eine verlässliche bzw. geordnete Übertragung unverzichtbar ist. Die Verwaltung der Zustände einer großen Anzahl von Verbindungen wird schnell zum Engpass. Verloren gegangene Fragmente können dazu führen, dass bereits empfangene Fragmente derselben Nachricht den Eingangspuffer des Empfängers verstopfen.

Die im Gegensatz zu UDP zusätzlich benötigten Nachrichten zur Kontrolle der Verbindung erhöhen zwar theoretisch die benötigte Bandbreite, jedoch wird dieses Manko meist dadurch wettgemacht, dass TCP/IP die IP-Pakete bis zur MTU auffüllt, und dadurch weniger Header-Informationen gesendet werden.

### 3.3.5 Application-Layer Multicasting

Die Funktionalität von IP-Multicasting kann auch auf der Anwendungsschicht implementiert werden (englisch *Application-Layer Multicasting*) (ALM). Dabei entstehen so genannte *Overlay-Netzwerke*, also virtuelle Netzwerke auf vorhandenen Netzen. Die Aufgabe der Multicast-Router wird auf Server umverteilt.

Multicasting auf der Anwendungsschicht ist wesentlich langsamer als IP-Multicasting, da die Kommunikationswege länger sind und jede Nachricht an jedem virtuellen Router den gesamten Netzwerkprotokollstapel durchlaufen muss. Trotzdem wird ALM manchmal selbst dann eingesetzt, wenn IP-Multicasting verfügbar wäre; denn auf der Anwendungsschicht können Gruppen- oder Verstopfungskontrolle sowie Verlässlichkeits- und Sicherheitsmechanismen einfacher implementiert und besser an spezielle Anwendungsfälle angepasst werden.

Multicasting auf der Anwendungsschicht eignet sich daher gut für Gruppenkommunikation mit hoher Bandbreite, bei der verlässliche Übertragung und konsistentes Gruppenmanagement wichtiger sind als niedrige Übertragungslatenz. Wegen des hohen Verarbeitungsaufwands beschränkt sich der Einsatz allerdings auf relativ kleine Gruppen.

Ein typisches Beispiel für ALM ist das *Multicast Backbone* (MBone, [Kum96]). Da in den frühen

90er Jahren nur wenige Router Multicast-fähig waren, wurde ein Overlay-Netzwerk geschaffen, das einige Server überwiegend wissenschaftlicher Einrichtungen miteinander verband. Diese übernahmen die Rolle der Multicast-Router, das heißt, sie speisten die Multicast-Nachrichten in die lokalen Netzwerke ein, leiteten sie an die anderen MBone-Server weiter und kopierten sie falls nötig.

Ein weiteres ALM-Protokoll ist *Narada* [CRZ00]. In diesem voll verteilten Peer-to-Peer-Protokoll organisieren sich die Gruppenmitglieder dynamisch in einem optimalen Netzwerk. *Overcast* [JGJ+00] konzentriert sich auf Multicasting mit einem einzigen Sender, und passt das Overlay-Netzwerk ebenfalls adaptiv an die Bedingungen der Teilverbindungen an. *ROMA* [KB03] implementiert verlässliches ALM über TCP/IP-Verbindungen und löst das Problem der unterschiedlichen Bandbreiten in einem heterogenen Netzwerk mithilfe von fehlerresistenten Codes und dem *forward-when-feasible-Paradigma*. *ALMI* [PSVW01] setzt einen ausgezeichneten Rechner als Gruppenverwalter ein und organisiert das Netzwerk mit einer anwendungsspezifischen Metrik in einem minimalen aufspannenden Baum. Der Gruppenverwalter tritt allerdings nur bei Veränderungen der Gruppenstruktur in Aktion, sodass der Verbund auch ohne ihn funktioniert.

### 3.3.6 HyperText Transfer Protocol

Das *HyperText Transfer Protocol* (HTTP) ist ein Stützpfeiler des WWW. HTTP ist ein Client-Server-Protokoll und setzt auf TCP/IP auf. Im Zusammenhang mit DVEs eignet sich HTTP besonders zur Bekanntmachung von Zugangspunkten eines DVEs, z.B. Server-Adressen, und zur Übertragung großer Dateien, wie dem Ausgangszustand der Szene beim Login oder benötigten Programmbibliotheken.

In Java kann man über HTTP zur Laufzeit Klassen- oder Ressourcen-Bibliotheken laden und sofort benutzen. Client-seitige Java-Programme, die *Applets*, können ebenfalls über HTTP vom Server auf den Client übertragen werden. Auf Server-Seite können über HTTP angesprochene *Servlets* [JS] mithilfe von *Java Server Pages* [JSP] dynamische Web-Inhalte generieren.

HTTP bietet zusätzlich Client-Authentisierung auf Basis von Passwörtern. Die authentisierte und verschlüsselte Variante *HTTPS* entspricht HTTP über TLS (vgl. Abschnitt 4.6.3).

### 3.3.7 Lightweight Directory Access Protocol

*Verzeichnisdienste* (englisch *directory services*) dienen zur Speicherung und Abfrage von Daten mit relativ langer Gültigkeitsdauer, z.B. Zertifikaten oder anderen Zugangsdaten. Wegen der langen Lebensdauer können Verzeichnisse einfach auf mehrere Server repliziert werden, wodurch Lastverteilung und verbesserte Verfügbarkeit erreicht werden können. Clients können einen Verzeichnisdienst mithilfe des *Lightweight Directory Access Protocol* (LDAP, [LDAP]) nutzen.

Die Einträge eines Verzeichnisses sind hierarchisch in einem Baum organisiert. Die einzelnen Einträge bzw. Knoten des Baums entsprechen dabei beispielsweise Ländern, Organisationen oder Personen, wobei Knoten auf derselben Ebene denselben Typ haben sollten. Der *Distinguished Name* (DN) eines Eintrags ist sein eindeutiger Name und besteht aus einem *Relative Distinguished Name* (RDN) und seinem *Kontext*, beschrieben durch die RDNs aller übergeordneten Einträge. Jedem RDN wird ein Kürzel für seinen Typ vorangestellt. Der DN für einen Eintrag mit meinen Daten könnte beispielsweise

DN: CN=Jan Köhnlein,OU=Informatik,O=TUHH,C=DE

lauten, wobei CN für *Common Name* steht, OU für *Organisational Unit*, O für *Organisation* und C für *Country*. Der RDN ist CN=Jan Köhnlein und der Kontext OU=Informatik,O=TUHH,C=DE.

Im Verzeichnis können außerdem an jeden Eintrag *Attribute* gebunden werden, beispielsweise ein Zertifikat oder eine Telefonnummer. Eine *Objektklasse* beschreibt einen Satz von möglichen bzw. notwendigen Attributtypen. Das Attribut objectClass eines Eintrags legt fest, welchen Objektklassen ein Eintrag genügt – hier beißt sich die Katze ein bisschen in den eigenen Schwanz. Die Beschreibung der gültigen Objektklassen und Attributtypen nennt man *Schema*.

Die in einem Verzeichnis abgelegten Daten sind oft öffentlich, und es ist keine Authentisierung notwendig. Ist dennoch Authentisierung gewünscht, bietet LDAP diese über das *Simple Authentication and Security Layer* (SASL, [SASL]) an. Des Weiteren kann die Kommunikation mithilfe von TLS (vgl. Abschnitt 4.6.3) geschützt werden.

Beispiele für Verzeichnisdienste sind *ActiveDirectory* von Microsoft oder *NIS* für UNIX-Systeme. Ein freier und als Quelltext verfügbarer LDAP-Server ist *OpenLDAP*.

In DVEs kommt LDAP insbesondere als Verzeichnis für Zertifikate zur Anwendung, damit diese nicht über den ohnehin ausgelasteten DVE-Kommunikationskanal gesendet werden müssen, sondern bei Bedarf über das Verzeichnis bezogen werden können. Außerdem kann ein LDAP-Verzeichnis Daten zum Auffinden von Server-Diensten oder anderen Teilnehmern enthalten.

## 3.4 Eine Kommunikationsbibliothek für DVEs

In diesem Abschnitt wird eine neue Kommunikationsbibliothek vorgestellt, die speziell für den Einsatz in DVEs optimiert ist. Sie ermöglicht unter anderem eine einfache und effiziente Integration kryptographischer Operationen, die im Weiteren einen wesentlichen Teil der Sicherheitsarchitektur für DVEs bilden.

### 3.4.1 Kategorisierung der Nachrichten

Die Nachrichten, die ein DVE produziert, lassen sich grob in zwei Kategorien unterteilen: Die einen müssen mit einer möglichst niedrigen Latenz übertragen werden, und die anderen erfordern eher eine verlässliche Übertragung. Da sich die Anforderungen der beiden Nachrichtenkategorien im Kern unterscheiden, ist es sinnvoll, zwei getrennte Kommunikationskanäle mit unterschiedlichen Protokollen zu verwenden.

In die erste Kategorie fallen die *Aktualisierungen der Entitätenzustände* (englisch *entity state updates*): Niedrige Latenz ist Trumpf, denn sie sichert hohe Interaktivität. Diese Nachrichten haben eine relativ geringe Gültigkeitsdauer. Wenn *absolute Updates* verwendet werden, also jede neue Nachricht die vorherigen überflüssig macht, kann sogar auf Empfangsbestätigung verzichtet werden. Die Ordnung der Updates wird durch darin enthaltene Zeitstempel definiert und muss nicht vom Übertragungsprotokoll übernommen werden. Sprachübertragung fällt in die gleiche Kategorie, da Interaktivität im Gespräch wichtiger ist als die fehlerfreie Übertragung aller Details. Für die Übertragung von Nachrichten dieser Kategorie eignen sich besonders paketbasierte Protokolle. Sie minimieren die Latenz, da ausgehende Nachrichten sofort gesendet werden können, und nicht, wie oft bei datenstrombasierten Protokollen, zunächst gepuffert werden müssen. In der Tat stellt die Datenstromabstraktion, insbesondere die Ordnung der Nachrichten, hier einen unvorteilhaften Overhead dar: Um an ein aktuelles Update zu kommen, müssen zunächst alle noch im Datenstrom benötigten Updates ausgelesen werden, wodurch eine wahrnehmbare Verzögerung entstehen kann. Paketbasierte Protokolle erfordern außerdem weniger Verarbeitungsaufwand, benötigen meist weniger Bandbreite und skalieren besser. Für Nachrichten dieser ersten Kategorie eignen sich also besonders Protokolle wie IP-Multicasting oder UDP.

Zur zweiten Kategorie gehören alle Nachrichten der Konfliktauflösung, der Registrierung und der Zugangskontrolle: Verzögerungsfreie Übertragung ist zwar erwünscht, aber nicht so wichtig wie die verlässliche Übertragung, da maßgebliche Entscheidungen im System auf diesen Nachrichten beruhen. Wird der Verlust einer Nachricht nicht erkannt, könnten falsche oder inkonsistente Entscheidungen mit weit reichenden Konsequenzen getroffen werden. Das gleiche gilt für fehlerhafte Nachrichten oder in der Reihenfolge vertauschte Nachrichten. Für Nachrichten dieser Kategorie muss man also verlässliche, geordnete Protokolle wie TCP/IP verwenden oder die erforderlichen Mechanismen selbst auf Anwendungsebene implementieren. Für verlässliche Gruppenkommunikation stehen die in Abschnitt 3.3.5 beschriebenen Multicast-Protokolle auf der Anwendungsschicht zur Verfügung.

### 3.4.2 Serialisierung

Um ein DVE-Nachrichtenobjekt zu senden, muss es zunächst in einen Byte-Strom umgewandelt werden. Dafür stellt Java innerhalb des *datenstrombasierten* Pakets java.io die *Java Serialization API* (Java-IO) bereit. Zum Serialisieren von Objekten dient die Klasse java.io.ObjectOutputStream, während die Deserialisierung von der Klasse java.io.ObjectInputStream bewerkstelligt wird. Ein serialisierbares Java-Objekt implementiert das Interface java.io.Serializable. Dieses umfasst zwar keine expliziten Methoden; ein serialisierbares Objekt kann allerdings, falls die Serialisierung von der Standardmethode abweichen soll, die privaten Methoden void writeObject(java.io.ObjectOutputStream out) und void readObject(java.io.ObjectInputStream in) implementieren. Attribute, die nicht serialisiert werden sollen, werden durch das Schlüsselwort transient in der Deklaration markiert.

Die Java-IO bietet eine Reihe von Vorteilen:

- Durch die Integration in die Java-VM kann direkt auf der internen binären Struktur der Objekte gearbeitet werden, die von außen aus Sicherheitsgründen nicht zugänglich ist.
- Aggregierte oder referenzierte Objekte können automatisch mit übertragen werden.
- Innerhalb einer Übertragung mehrfach referenzierte Objekte werden nur einmal serialisiert und ansonsten per Objekt-ID referenziert. Die Objekte einer Übertragung behalten dadurch ihre Identität.
- Typsicherheit ist gewährleistet, da eine Klassen-ID für jedes Objekt mitgesendet wird. Kann die Klasse eines empfangenen Objektes vom einem java.lang.ClassLoader des Empfängers nicht in derselben Version geladen werden, meldet die Serialisierung eine Ausnahme.
- Die Serialisierung ist polymorph. Alle Klassen, die das Interface Serializable implementieren, können über denselben Kommunikationskanal versendet werden. Mithilfe von *reflection* kann der Empfänger die Klasse empfangener Objekte bestimmen und entsprechende Typumwandlungen vornehmen.

Als Alternative zur Java-IO ermöglicht die seit JDK Version 1.4 enthaltene *New I/O API* (Java-NIO) die direkte Verwendung von Pufferspeichern als java.nio.ByteBuffer-Objekte. Dadurch kann der durch die Datenstromabstraktion entstehende Overhead umgangen werden. Insbesondere wenn statt Objekten nur Datenblöcke ohne Identität versendet werden sollen, kann das erhebliche Leistungsgewinne bringen. Allerdings fällt damit auch der Komfort der Standardserialisierung der Java-IO weg, und der Programmierer muss die Serialisierung für jede Nachrichtenklasse selbst explizit

Tabelle 3.2: Vergleich der Standard-Java-Serialisierung mit dem neu entwickelten DveSec-Paket

| | *Serialisierung* [*ms*] | *Deserialisierung* [*ms*] | *Summe* [*ms*] | *Zeit / Objekt* [*ms*] | *Paketgröße* [*Byte*] |
|---|---|---|---|---|---|
| *Java-IO* | 3302,4 | 1915,2 | 5217,6 | 0,010435 | 230 |
| *Java-NIO* | 614,2 | 951,6 | 1565,8 | 0,003131 | 64 |
| *DveSec stream* | 1660,4 | 1433,4 | 3093,8 | 0,006187 | 66 |
| *DveSec buffer* | 707,4 | 1019,8 | 1727,2 | 0,003454 | 66 |

festlegen. Polymorphie empfangener Nachrichten muss ebenfalls von Hand implementiert werden. Im Gegensatz zur datenstrombasierten Java-IO muss außerdem die Maximalgröße von Puffern bei deren Erzeugung festgelegt werden.

Um den Durchsatz der beiden APIs zu vergleichen, wurden Tests in der Testumgebung (vgl. Abschnitt B.7) durchgeführt. Dazu wurde eine serialisierbare Testklasse entworfen, die mehrere numerische Attribute mit einer Gesamtgröße von 64 Byte enthält. Die Klasse enthält außerdem die Methoden void encode(ByteBuffer) und void decode(ByteBuffer), die die Attribute direkt in den gegebenen ByteBuffer schreiben bzw. aus dem Puffer lesen. Zunächst wurde ein Feld von 500.000 Objekten dieser Klasse generiert. Dieses wurde einmal mit der Java-IO und einmal mit der Java-NIO serialisiert und wieder deserialisiert. Die über fünf Versuche gemittelten Ergebnisse sind in Tabelle 3.2 zusammengefasst.

Die datenstrombasierte Java-IO ist also um den Faktor 3,3 langsamer als die pufferbasierte Java-NIO und generiert obendrein 3,6 mal größere Pakete. Für groß angelegte DVEs mit extrem vielen kurzen Nachrichten kommt die Java-IO also nicht infrage. Ist die Kommunikation ohnehin paketbasiert, so stellt schon die Datenstromabstraktion einen unnötigen Overhead dar.

Für diese Arbeit habe ich daher ein neues Serialisierungskonzept auf Basis der Java-NIO-Klasse ByteBuffer entworfen. Die Ziele waren:

- Fest programmierter Serialisierungsablauf für jeden Nachrichtentyp in Methoden der Klasse selbst (Lokalität).
- Einfache Abbildung von Aggregation und Vererbung auf die Serialisierung.
- Effiziente Nutzung der Java-NIO-Klasse ByteBuffer für Nachrichten fester oder begrenzter Länge.
- Verwendung von Java-IO Datenströmen für Nachrichten unbekannter Länge.
- Ein einheitliches Interface für puffer- und datenstrombasierte Serialisierung.
- Vermeidung der Java-Reflection-API für polymorphe Nachrichten, da diese sehr langsam ist. Stattdessen Einsatz fester Nachrichtentyp-IDs (encodableIds).
- Übertragung der encodableId kann bei vereinbarten Nachrichtentypen unterdrückt werden.
- Typsicherheit auf der Empfängerseite: Alle deserialisierbaren Nachrichtenklassen müssen zur Laufzeit registriert werden.
- Bündelung mehrerer Nachrichtentypen zur systematischen Implementierung von Protokollen.

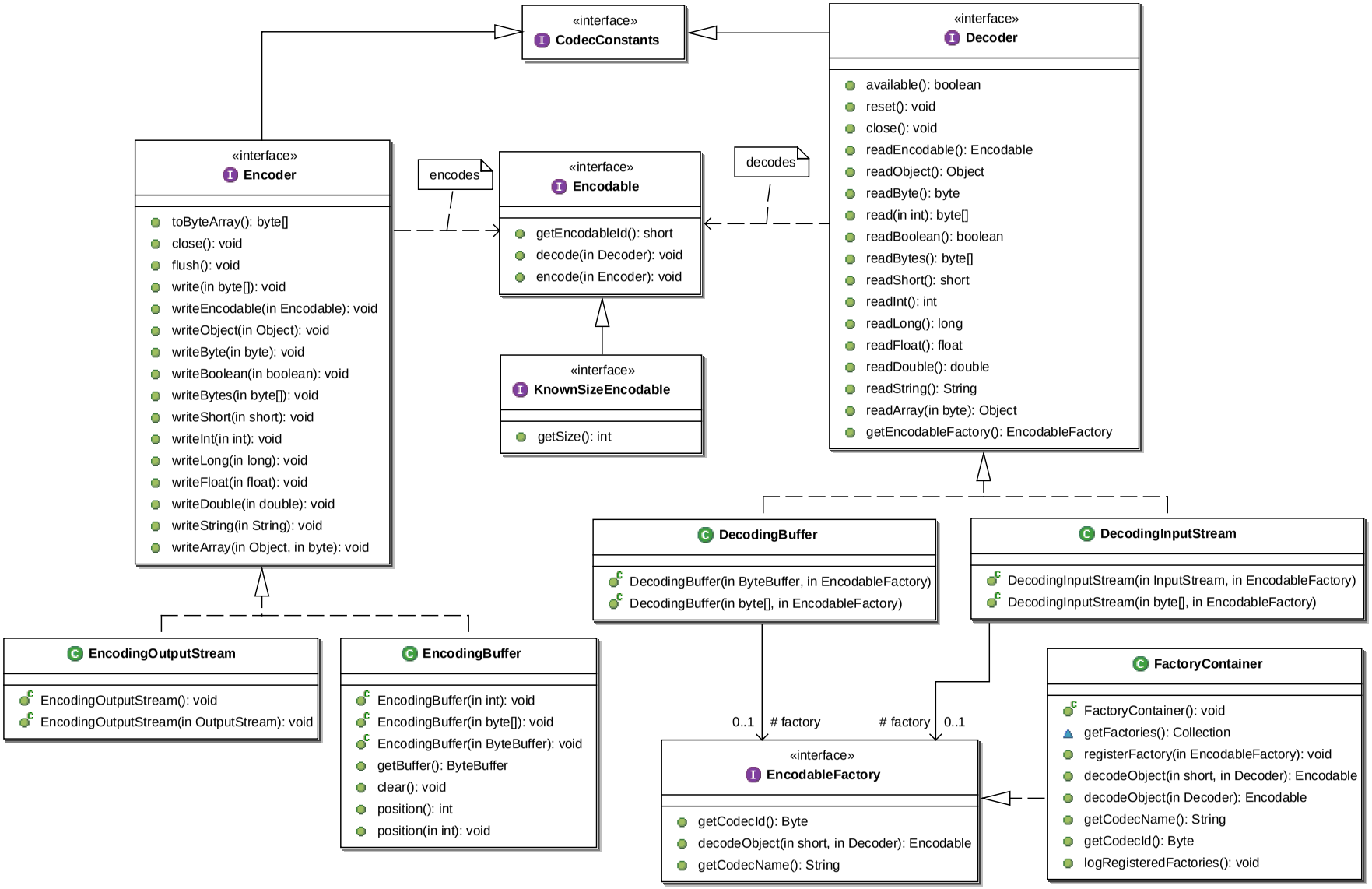



Abbildung 3.3: Klassendiagramm des dvesec.codec Pakets

```
1  public ClassA implements Encodable {
2    private int n;
3    private ClassB b;
4    private Encodable c;
5
6    ...
7
8    public void encode(Encoder enc) {
9      enc.writeInt(n);
10     b.encode(enc);           // serialisiert b direkt ohne encodableId
11     enc.writeEncodable(c);   // serialisiert c mit encodableId
12   }
13
14   public void decode(Decoder dec) {
15     n = dec.readInt();
16     b = new ClassB(enc);     // deserialisiere b (vereinbarter Typ)
17     c = dec.readEncodable(); // deserialisiere c (Polymorphie)
18   }
19 }
```

Abbildung 3.4: Serialisierungsmethoden der Beispielklasse ClassA

- Direkte und effiziente Einbindung von kryptographischen Operationen, die die serialisierte Form der Nachricht benötigen.

Die Klassen des Programmpakets dvesec.codec sind im Klassendiagramm 3.3 dargestellt.

Das Interface Encodable muss analog zum Serializable-Interface der Java-IO von jeder serialisierbaren Klasse implementiert werden. Die Methode encode(Encoder) serialisiert ein Objekt während decode(Decoder) die Deserialisierung übernimmt. In diesen beiden Methoden wird für jedes zu serialisierende Attribut die dem Typ entsprechende Schreib- bzw. Lesemethode des übergebenen Encoder- bzw. Decoder-Objekts aufgerufen. Für die Interfaces Encoder und Decoder liegen jeweils eine puffer- und eine datenstrombasierte Implementierung vor, wobei letztere auf beliebigen Java-IO-Datenströmen aufsetzen kann.

Um Encodable-Objekte zu (de-)serialisieren, deren Typ a priori nicht bekannt ist, werden die Methoden writeEncodable(Encodable) des Encoders bzw. decodeEncodable() des Decoders verwendet. Dabei wird der Serialisierung zur Identifizierung des Typs die encodableId der Nachrichtenklasse vorangestellt. Auf der Decoder-Seite werden die EncodableFactory-Objekte der gewünschten Nachrichtenklassen registriert. Diese haben jeweils eine eindeutige codecId und kennen die classId von zugehörigen Encodable-Klassen. In der Methode Decoder.readEncodable() wird zunächst die codecId gelesen, anhand derer der Decoder die zugehörige EncodableFactory identifiziert. Dieser liest anschließend die classId und ruft die Methode decodeObject(Decoder, classId) der EncodableFactory auf. Dort wird schließlich ein Konstruktor der mit der classId bezeichneten Klasse aufgerufen, der das Objekt mittels decode(Decoder) deserialisiert. Das resultierende Objekt wird von der Methode Encodable Decoder.decodeEncodable() als Ergebnis zurückgeliefert. Die codecId und die classId einer Encodable-Klasse sind in der encodableId zusammengefasst.

Ein großer Vorteil des dvesec.codec-Pakets besteht darin, dass sowohl vereinbarter als auch polymorpher Nachrichtenaustausch mit optimalen Paketgrößen realisierbar ist. Im vereinbarten Nachrichtenaustausch erwartet der Empfänger genau einen Nachrichtentyp und benötigt daher die encodableId des Typs nicht. Dieser Fall tritt insbesondere bei der Deserialisierung von Nachrichten-

attributen auf, deren Typ in der Klasse der Nachricht festgelegt ist. Abbildung 3.4 zeigt dazu ein kurzes Beispiel: Die Encodable-Klasse ClassA deklariert ein Integer-Attribut n, ein Attribut b der Encodable-Klasse ClassB und eine Attribut c einer beliebigen Encodable-Klasse.

Die Ergebnisse des Durchsatztests der neuen DveSec-API sind ebenfalls in Tabelle 3.2 auf Seite 31 dargestellt. Die datenstrombasierte Implementierung ist immer noch deutlich schneller als die konventionelle Java-IO. Die pufferbasierte Implementierung reicht beinahe an die Java-NIO heran. Der leichte Verlust ist zum einen durch die zusätzliche Indirektion durch das Encoder-Interface und zum anderen durch die Mechanismen für polymorphe Serialisierung zu erklären.

### 3.4.3 Kommunikationskanäle

Die Java-IO-Bibliothek bietet unterschiedliche Schnittstellen für verschiedene Kommunikationskanäle an. Für die datenstrombasierte Kommunikation, z.B. für Dateien oder TCP/IP-Sockets, werden die Klassen java.io.OutputStream und java.io.InputStream bereitgestellt. Die paketbasierte Kommunikation hat dagegen eine völlig andere Schnittstelle: Um ein Datenpaket per UDP zu verschicken, muss zunächst der Inhalt als byte-Feld vorliegen. In einem java.io.DatagramPacket-Objekt eingepackt, kann dieses dann über ein java.io.DatagramSocket verschickt werden. Auf der Empfängerseite wird dann das byte-Feld wieder aus dem DatagramPacket extrahiert und verarbeitet.

Mit der Java-NIO-Bibliothek kann man ByteBuffer-Objekte direkt an java.nio.channels.Channel senden und hat damit eine für UDP- und TCP/IP-Sockets sehr ähnliche und hochleistungsfähige Schnittstelle. Zusätzlich unterstützt die Java-NIO nicht-blockierende Kommunikation mithilfe von Selektoren (vgl. Abschnitt 3.2.5).

Aufbauend auf dem oben beschriebenen Serialisierungspaket dvesec.codec habe ich für diese Arbeit das Programmpaket dvesec.channel entwickelt, das speziell an die besonderen Erfordernisse von DVEs angepasst ist, nämlich:

- Möglichst hohe Leistung speziell bei kurzen Nachrichten, wie Zustands-Updates von DVE-Entitäten.
- Minimale Latenz.
- Minimale Netzwerkauslastung.
- Möglichst geringer Verarbeitungsaufwand.
- Maximale Skalierbarkeit.
- Nach außen einfache Benutzbarkeit.

Im Kommunikationspaket dvesec.channel wurden vier wesentliche Konzepte umgesetzt: Die Abstraktion des Kommunikationskanals, die Verwendung eines Serialisierungspuffers, hintereinander schaltbare Nachrichtenfilter und die Sitzungsverwaltung. Jedes dieser Konzepte stelle ich im Folgenden in einem eigenen Absatz vor.

#### Abstraktion des Kommunikationskanals

Die Übertragung von Nachrichten kann über sehr verschiedene Kanäle erfolgen. Bereits beschrieben wurden TCP/IP-, UDP- und Multicast-Sockets. Nachrichten können aber auch aus Dateien oder

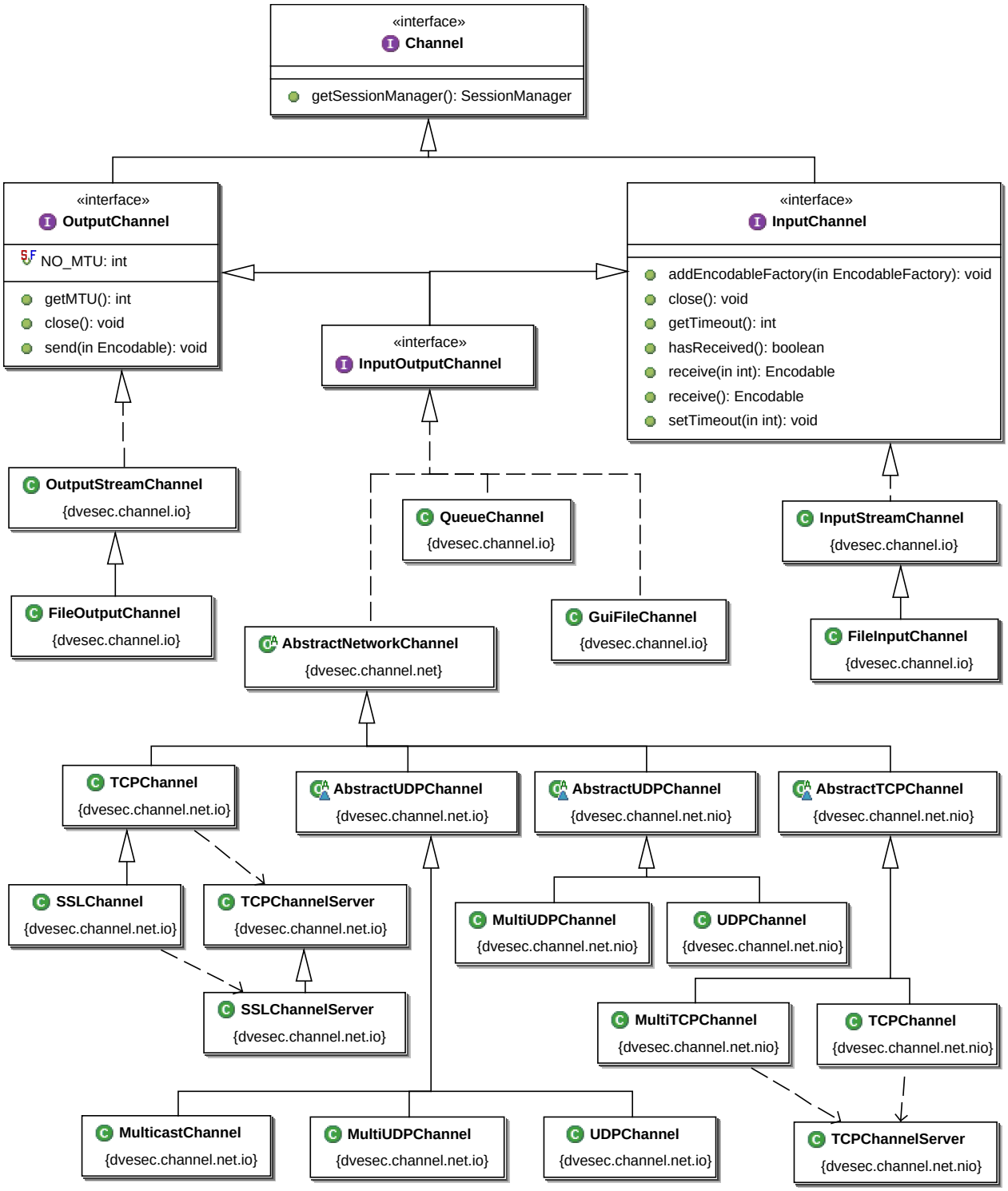



Abbildung 3.5: Klassendiagramm des dvesec.channel Pakets

Warteschlangen gelesen bzw. dorthin geschrieben werden oder sogar per E-Mail übertragen werden. Um unterschiedliche Kommunikationskanäle einheitlich nutzbar und vergleichbar zu machen, wurde eine Abstraktion des Kommunikationskanals entworfen und implementiert.

Da in DVEs besonders viele kurze Nachrichten auftreten und um alle Kommunikationskanäle auf einen gemeinsamen Nenner zu bringen, ist dvesec.channel paketbasiert. Natürlich können die

Pakete trotzdem über datenstrombasierte Kanäle wie TCP/IP-Sockets transportiert werden.

Die Basisklassen sind im Klassendiagramm 3.5 dargestellt. Jeder Kommunikationskanal implementiert das Interface Channel. Die Schnittstellen OutputChannel und InputChannel definieren die wesentlichen Methoden für Ein- und Ausgabe von dvesec.codec.Encodable-Objekten. Die Methode OutputChannel.getMTU() liefert die maximale Paketgröße, und damit die Größenbeschränkung für serialisierte Encodable-Objekte. Kanäle ohne Größenbeschränkung liefern hier den Wert NO_MTU zurück. Die Methode send(Encodable) versendet ein Encodable-Objekt über diesen Kanal, sodass es auf der Empfängerseite vom passenden InputChannel mit einer der Methoden receive() bzw. receive(timeout) empfangen werden kann, falls dort die notwendige EncodableFactory registriert wurde. Um die Verwendung der API möglichst einfach zu halten, sind die Schnittstellen blockierend. Unterstützt der zu Grunde liegende Kommunikationskanal nicht-blockierende Kommunikation, kann die Methode InputChannel.hasReceived() so implementiert werden, dass ein Blockieren beim Warten auf den Empfang einer Nachricht vermieden werden kann. Ansonsten können, wie bei blockierender Kommunikation üblich, Timeout-Werte in Millisekunden angegeben werden. Das Interface InputOutputChannel fasst Ein- und Ausgabe zusammen, wie es für Sockets sinnvoll ist.

AbstractNetworkChannel ist die abstrakte Basisklasse aller netzwerkbasierten Kommunikationskanäle. Im unteren Bereich befinden sich auf der linken Seite die Implementierungen mit Java-NIO-Kanälen (Paket dvesec.channel.net.nio) und auf der rechten Seite die mit Java-IO-Sockets (Paket dvesec.channel.net.io). Leider unterstützt die zum Zeitpunkt dieser Arbeit aktuelle Version der Java-NIO-Bibliothek weder Multicast- noch TLS-Sockets.[2]

TCP/IP-basierte Kanäle erhält man, wie in Java üblich, auf der Server-Seite über eine Socket-Fabrik TCPChannelServer (bzw. SSLChannelServer) und auf der Client-Seite direkt über die Klasse TCPChannel.

Die Klassen MultiTCPChannel, MultiUDPChannel und MulticastChannel implementieren Kanäle, über die mit mehreren unterschiedlichen Kommunikationspartnern gleichzeitig kommuniziert werden kann. Genaueres dazu gibt es in Abschnitt 3.4.5.

Fast alle in Kapitel 5 vorgestellten Softwarekomponenten kommunizieren über solche Kanäle. Wenn sich zwei kommunizierende Komponenten auf demselben Rechner befinden, können diese über einen QueueChannel kommunizieren. Dieser verwaltet eine Warteschlange für jede Kommunikationsrichtung. Zur Übertragung durch einen QueueChannel werden Nachrichten nicht serialisiert bzw. deserialisiert.

InputStreamChannel und OutputStreamChannel lesen bzw. schreiben Java-IO-Datenströme. Davon abgeleitet sind FileInputChannel und FileOutputChannel, mit denen Dateien zur Kommunikation verwendet werden können. Die Klasse GuiFileChannel ist ein besonderer dateibasierter Ein- und Ausgabekanal, der eine graphische Benutzerschnittstelle besitzt, mit der der Name der Eingabe- bzw. Ausgabedatei interaktiv gesetzt werden kann. Ein solcher Kanal kommt beispielsweise bei *Certification Authorities* zum Einsatz, die nicht direkt mit dem Netzwerk verbunden sind (vgl. Abschnitt 5.5.6).

### Serialisierungspuffer

Alle Kommunikationskanäle sollen mit einem möglichst hohen Nachrichtendurchsatz arbeiten. Um das zu erreichen, wurde Wert darauf gelegt, dass jede Nachricht nur einmal serialisiert wird, und Umkopieren von serialisierten Nachrichten wenn möglich vermieden wird.

[2] TLS-Unterstützung ist in Java Version 1.5 vorgesehen.

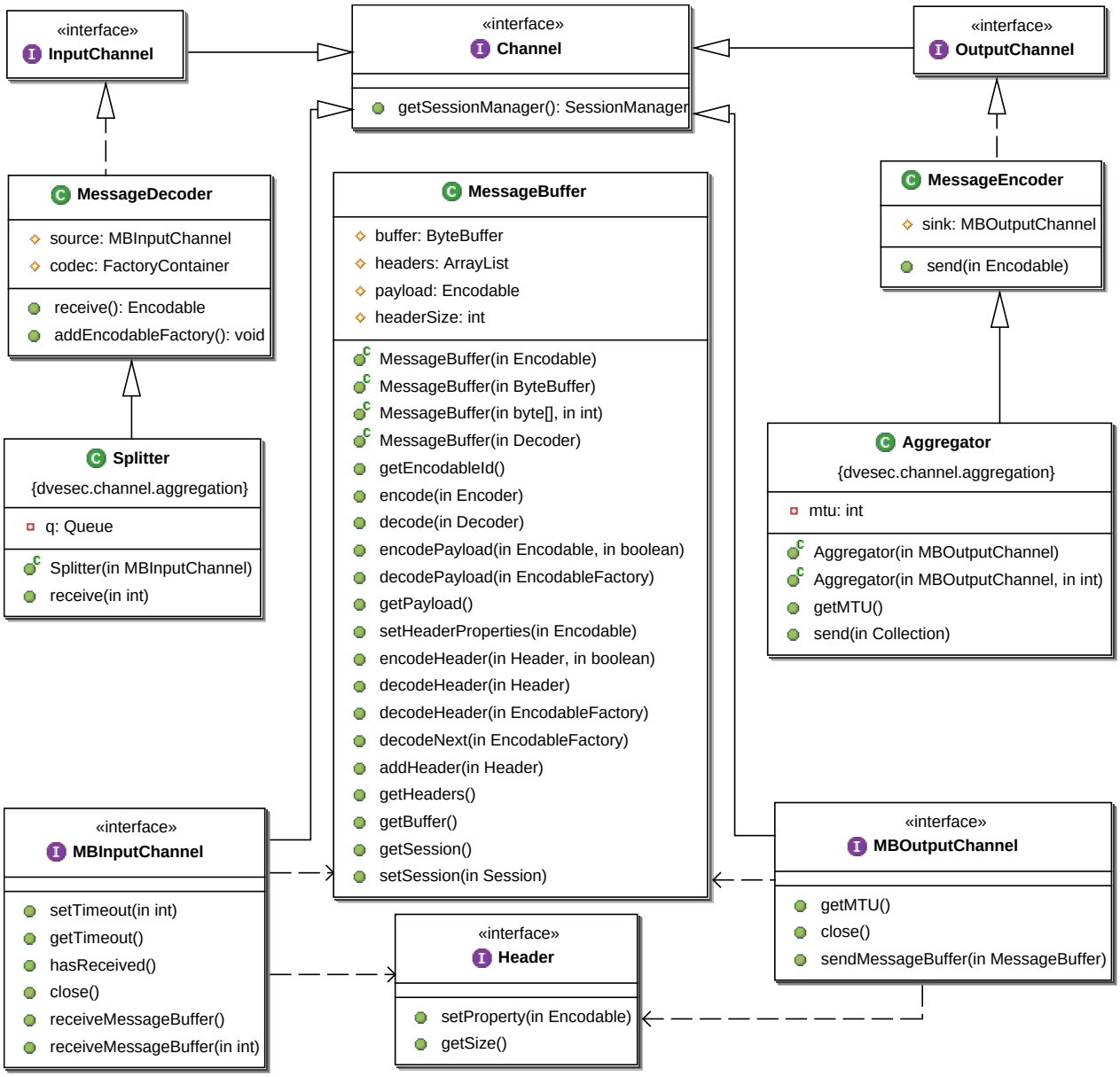



Abbildung 3.6: Klassendiagramm der Pufferserialisierung

Daher wurde die Klasse MessageBuffer ins Leben gerufen. Abbildung 3.6 zeigt ein Klassendiagramm rund um diese Klasse. Jede Encodable-Nachricht wird genau einmal als Nutzlast payload in einen MessageBuffer serialisiert. Dieser kapselt einen java.nio.ByteBuffer, der die serialisierte Nachricht enthält. Operationen auf der serialisierten Nachricht, wie die Berechnung eines MAC oder einer digitalen Signatur, können direkt auf diesem ByteBuffer durchgeführt werden, wodurch eine Kopie oder eine zweite Serialisierung vermieden werden. Der ByteBuffer kann direkt von Java-NIO-Kanälen verarbeitet werden.

Mithilfe eines MessageBuffer-Objekts können einer Nachricht außerdem ein oder mehrere Header hinzugefügt werden. Da Header im Puffer vor der serialisierten Nachricht stehen, muss ihre Größe im Voraus bekannt sein. Die Methode Header.getSize() liefert diese Größe. Ein Beispiel für die Verwendung von Header-Objekten findet sich im nächsten Abschnitt.

Die Klasse MessageBuffer implementiert zwar das Interface Encodable, jedoch ist es für einen

Ausgabekanal besser zu wissen, ob eine Nachricht bereits als MessageBuffer vorliegt, und ein Eingabekanal kann auch einen MessageBuffer liefern, wenn noch Operationen auf der serialisierten Nachricht durchzuführen sind, z.B. die Überprüfung einer digitalen Signatur. Daher gibt es neben den Interfaces InputChannel und OutputChannel die Schnittstellen MBInputChannel und MBOutputChannel, mit denen man statt Encodable-Objekten MessageBuffer-Objekte senden bzw. empfangen kann. Da der MBInputChannel keine Nachrichten deserialisiert, benötigt er auch keine EncodableFactory. Die Klassen MessageEncoder kapselt einen MBOutputChannel, implementiert alle Methoden der Schnittstelle OutputChannel und agiert als eine Art Konverter zwischen Encodable-basierten OutputChannel-Objekten und MessageBuffer-basierten MBOutputChannel-Objekten. Analoges gilt für die Klasse MessageDecoder. Zum Deserialisieren der Nachricht wird die entsprechende EncodableFactory benötigt.

Tatsächlich arbeitet jede Kanalimplementierung aus dem letzten Abschnitt intern zunächst nur als MessageBuffer-Kanal. Damit die Anwendung nun nicht jedem Kanal einen eigenen MessageEncoder bzw. MessageDecoder vorschalten muss, sind diese bereits als Attribute in den Kanalimplementierungen vorgesehen, die bei Bedarf instanziiert werden und nach dem Fassaden-Entwurfsmuster (vgl. [GHJV95]) über die Schnittstelle des Kanals angesteuert werden. Die Instanziierung erfolgt automatisch beim ersten Aufruf der Methode send(Encodable) bzw. wenn die erste EncodableFactory mittels addEncodableFactory(EncodableFactory) registriert wird.

Auf den ersten Blick mögen zwei so ähnliche Interface-Typen wie MBOutputChannel und OutputChannel redundant erscheinen. Insbesondere für das im nächsten Abschnitt behandelte Filterkonzept ist diese Trennung allerdings sinnvoll. Innerhalb der Filterpipeline operieren alle Kanäle mit den effizienten MessageBuffer-Objekten, während die Anwendung außerhalb nur mit Encodable-Nachrichten arbeitet. Somit werden alle ByteBuffer-Operationen gekapselt und die Anwendung kümmert sich nur um Objekte und nicht um deren binäre Repräsentation.

### Aggregation von Nachrichten

Um mehrere Nachrichten in einem Sammelpaket zusammenzufassen, kann ein Aggregator eingesetzt werden. Dieser fasst die ihm übergebenen Nachrichten in möglichst wenige MessageBuffer zusammen, deren Größe maximal der MTU des bedienten MBOutputChannels entspricht. Auf der Empfängerseite dekodiert ein Splitter die einzelnen Teilnachrichten und reiht sie zum Empfang für die Anwendung in einer Warteschlange ein. Die wichtigsten Eigenschaften der Nachrichtenaggregation sind:

- Bessere Nutzung des Netzwerks durch Aggregation von Nachrichten. Da nicht mehr jede Kleinstnachricht einzeln geroutet werden muss, wird das Netzwerk wesentlich effizienter genutzt.

- Header müssen nur einmal pro Sammelpaket übertragen und bearbeitet werden. Der Netzwerkverkehr und Bearbeitungsaufwand werden somit reduziert.

- Keine Fragmentierung von Nachrichten (vgl. Abschnitt 3.2.4). Dadurch enthalten alle Sammelpakete nur vollständige Nachrichten.

- Kein Warten auf weitere Nachrichten bis ein Sammelpaket voll ist, da sonst die Latenz steigen würde. Stattdessen sendet die Anwendung zusammengehörige Nachrichten auf einmal mittels send(Collection).

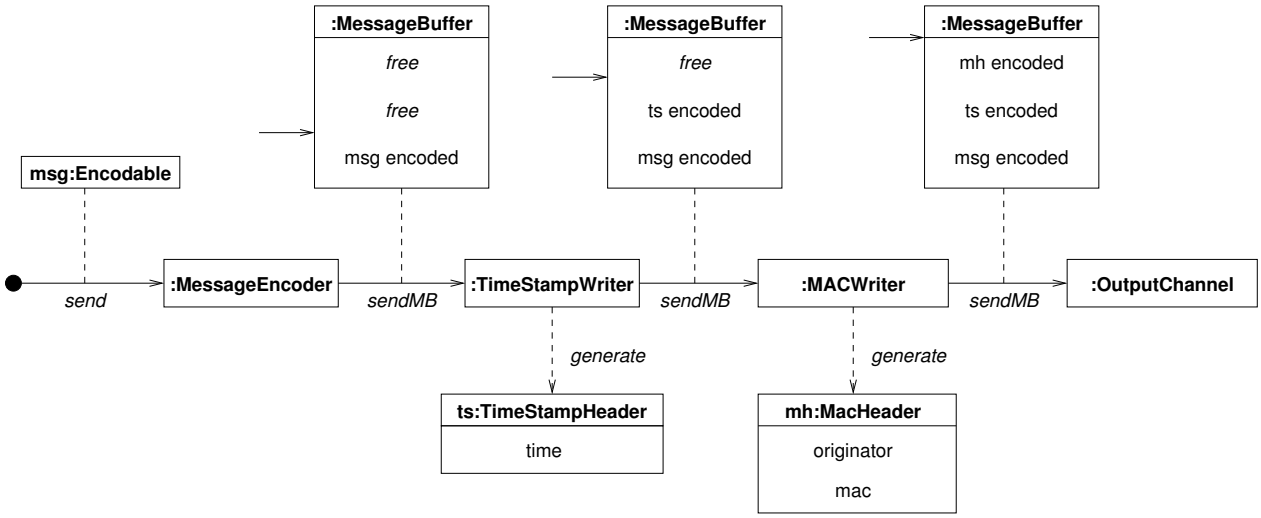



Abbildung 3.7: Funktionsweise von OutputFilter-Objekten

- Transparenz auf der Empfängerseite: Die Nachrichten werden dort genauso verarbeitet als wären sie einzeln geschickt worden.

### 3.4.4 Nachrichtenfilter

Die abstrakten Klassen InputFilter und OutputFilter bilden die Basis für *Nachrichtenfilter* (siehe Abbildung 3.5 auf Seite 35). Ein *Filter* (vgl. [BMR+96]) stellt keinen eigenen Kommunikationskanal dar, sondern erweitert Nachrichten, bereitet sie auf oder filtert sie aus, bevor sie versendet werden bzw. nachdem sie empfangen wurden. Da ein Filter die gleiche Schnittstelle wie ein Kommunikationskanal hat, kann er ohne aufwändige Änderungen zwischen eine vorhandene Anwendung und deren Kommunikationskanal geschaltet werden. Ebenso können mehrere Filter hintereinander geschaltet werden. Dadurch entsteht eine *Filterpipeline*, deren einzelne Stufen effizient über die MBInputChannel- bzw. MBOutputChannel-Schnittstellen kommunizieren. Viele Filter erweitern die eigentliche Nachricht um einen Header, in dem zusätzliche Informationen wie ein Zeitstempel, ein MAC (vgl. Abschnitt A.6.1) oder eine digitale Signatur (vgl. Abschnitt A.6.2) enthalten sind.

Das Zusammenspiel von OutputFilter, OutputChannel und MessageBuffer ist in Abbildung 3.7 skizziert. Die Filterpipeline besteht hier aus einem MessageEncoder, der die Nachricht serialisiert, einem TimeStampWriter, der Zeitstempel erzeugt, einem MacWriter, der einen MAC berechnet und einem beliebigen OutputFilter. Der MessageEncoder erzeugt einen neuen MessageBuffer, in den die serialisierte Nachricht und alle Header hineinpassen. Die Nachricht wird dann am unteren Ende des Puffers serialisiert. Der Positionsanzeiger im Puffer steht danach auf dem Beginn der Nachricht. Der MessageBuffer wird an den TimeStampWriter weitergereicht. Dieser erzeugt den TimeStampHeader und schreibt dessen serialisierte Form direkt vor die serialisierte Nachricht in den Puffer. Der Positionsanzeiger wandert nach oben auf den Beginn des Zeitstempels. Der MacWriter generiert einen neuen MacHeader, der neben dem eigentlichen MAC einen Bezeichner für den Urheber der Nachricht enthält. Wie alle Filter schreibt er seinen Header direkt vor die aktuelle Pufferposition. Der OutputChannel sendet den empfangenen Puffer an den Empfänger.

Die Empfängerseite ist in Abbildung 3.8 skizziert. Die Filterpipeline besteht aus den Gegenspielern der Sender-Pipeline in umgekehrter Reihenfolge. Der InputChannel empfängt die Nachricht

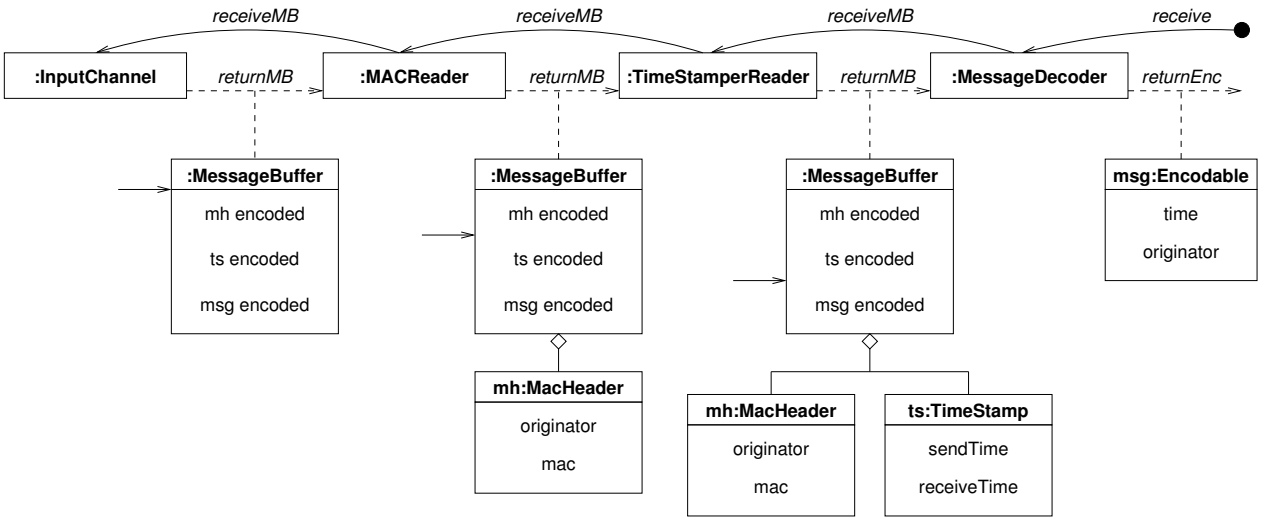



Abbildung 3.8: Funktionsweise von InputFilter-Objekten

als Nachrichtenpuffer und leitet sie an den MacReader weiter. Dieser extrahiert den MacHeader und prüft ihn. Ist der MacHeader ungültig, meldet der MacReader eine Ausnahme, die von den übrigen Filtern sofort zur Anwendung durchgereicht wird. Ist der MacHeader gültig, hängt ihn der MacReader als Objekt an den Puffer an. Der Positionsanzeiger steht dann direkt hinter dem dekodierten Header. Der TimeStampReader dekodiert den TimeStampHeader und hängt ihn ebenfalls an den Puffer an. Der MessageDecoder deserialisiert anschließend die eigentliche Nachricht. Dabei wird bei jedem angehängten Header die Methode setProperty(msg) aufgerufen, die, falls die Klasse von msg die entsprechenden Schnittstellen – in diesem Falle wären das die Interfaces TimeStamped und Authenticated – implementiert, die jeweilige Header-Eigenschaft setzt.

Der Filtermechanismus hat folgende Eigenschaften:

- Auf der Basis von Filtern werden Header sofort nach der Deserialisierung ausgewertet. Nachrichten mit ungültigen Headern werden früh erkannt und müssen nicht weiter bearbeitet werden.
- Durch Filter können Nachrichtenkanäle auf für die Anwendung transparente Weise verbessert werden. So können Kanälen beispielsweise Integritätsprüfung, Authentisierung oder Verschlüsselung ohne Änderung im Rest der Anwendung hinzugefügt werden.
- Der Anwendungsprogrammierer wählt selbst für jeden Nachrichtentyp aus, ob aus der Bearbeitung der Nachrichten-Header gewonnene Informationen in der Anwendung benötigt werden. Die Nachrichtenklasse muss dazu einfach das zum Header gehörige Interface implementieren.
- Filteroperationen werden abstrahiert, indem Methode und Ergebnis getrennt werden. Ein Authentisierungsfilter speichert beispielsweise das Ergebnis, also den authentisierten Prinzipal, über das Authenticated-Interface in der Nachricht, und nicht etwa den verwendeten MAC selbst.
- Alle Nachrichtenteile werden nur einmal serialisiert. Header, die bereits serialisierte Teile benötigen, können direkt auf dem MessageBuffer operieren, wie der MacWriter im Beispiel, der sogar noch den Zeitstempel einschließen kann.

- payload und headers eines Nachrichtenpuffers bleiben auch nach der Serialisierung erhalten und ermöglichen so, dass Filter auf die Aktionen vorangegangener Filter ohne Deserialisierung der Nachricht reagieren können.
- Mit Filtern können geschichtete Protokolle entworfen werden. Jeder Filter entspricht dabei einer Protokollschicht und kommuniziert ausschließlich mit der nächsten Stufe der Filterpipeline.
- Mithilfe eines Aggregator-Objektes wird die Effizienz von Filtern (vgl. Abschnitt 3.4.3) erheblich gesteigert. Selbst auf datenstrombasierten Kanälen kann der Aggregator sinnvoll zur Regelung der Granularität bestimmter Operationen eingesetzt werden, beispielsweise um festzulegen, wie oft ein MAC gesendet werden muss.
- Die Filterarchitektur befreit die Anwendung von den Einschränkungen monolithischer Sicherheitsprotokolle wie TLS. So können die Reihenfolge kryptographischer Operationen verändert und die Algorithmen einfach ausgetauscht werden.

Abbildung 3.9 zeigt ein Klassendiagramm der für diese Arbeit implementierten Filter. Im oberen Bereich befinden sich die Eingabefilter, in der Mitte die zugehörigen Header und unten die Ausgabefilter.

Der TimeStampWriter erzeugt nur den sendTime-Zeitstempel eines TimeStamp-Objektes. Die receiveTime wird auf der Empfängerseite vom TimeStampReader gesetzt. Der Zeitstempel wird an Encodable-Objekte weitergegeben, die das Interface TimeStamped (ohne Abbildung) implementieren. Jeder Zeitstempel ist wie in Standard-Java ein 64-Bit-Integer-Wert, der die Anzahl der vergangenen Millisekunden seit dem 1. Januar 1970 um 0.00 Uhr beschreibt.

Die kryptographischen Daten einer Verbindung, wie Schlüssel zur Berechnung von MACs und zur Chiffrierung, Initialisierungsvektoren der Chiffre und die beteiligten Prinzipale, werden jeweils in einem SecurityAssociation-Objekt gespeichert. Der MAC einer Nachricht wird vom MacWriter im Attribut mac des MacHeaders abgelegt. Der MacReader ermittelt die richtige SecurityAssociation und verifiziert den MAC mit der Methode MacHeader.verify(). Bei Erfolg wird der remotePrincipal der SecurityAssociation als originator an Nachrichten weitergegeben, die das Interface Authenticated (ohne Abbildung) implementieren. Der hier eingesetzte MAC-Algorithmus SHA-1 liefert 160-Bit-Hashwerte.

Der EncryptionHeader enthält keine Informationen, die an das entschlüsselte Nachrichtenobjekt weitergegeben werden müssen. Der Encryptor benutzt in der Standardkonfiguration AES mit einer Schlüssellänge von 128 Bit. Der Chiffretext einer Nachricht darf nicht von vorangegangenen Nachrichten abhängen, da bei der Verwendung von UDP eventuell Pakete verloren gehen. Wenn der Betriebsmodus der Chiffre kein Padding benötigt, wird der Chiffretext nicht verlängert. Ich habe mich daher für den Zähler-Modus (vgl. Abschnitt A.4.1) als Voreinstellung entschieden, aber es können selbstverständlich auch alle anderen von der JCE angebotenen Modi verwendet werden. Der Encryptor ermittelt die vom Betriebsmodus abhängige maximale Nachrichtenexpansion und passt seine MTU entsprechend an.

Im Zähler-Modus müssen alle Nachrichten durchnummeriert werden, damit der Empfänger die richtige Position im Schlüsselstrom finden kann. Ein 32-Bit-Integer-Wert reicht dazu völlig aus: Ausgehend von der Ethernet-MTU von 1500 Byte bleiben neben der Seriennummer für jedes Paket 1496 Byte, also könnten insgesamt 6.425.271.074.816 Byte mit demselben Schlüssel verschlüsselt werden, ohne dass der Zähler überläuft. Das ist wesentlich mehr als die Menge Daten, die mit einem 128-Bit-Schlüssel chiffriert werden sollte (vgl. Abschnitt 4.2.2). Der Zähler wird durch

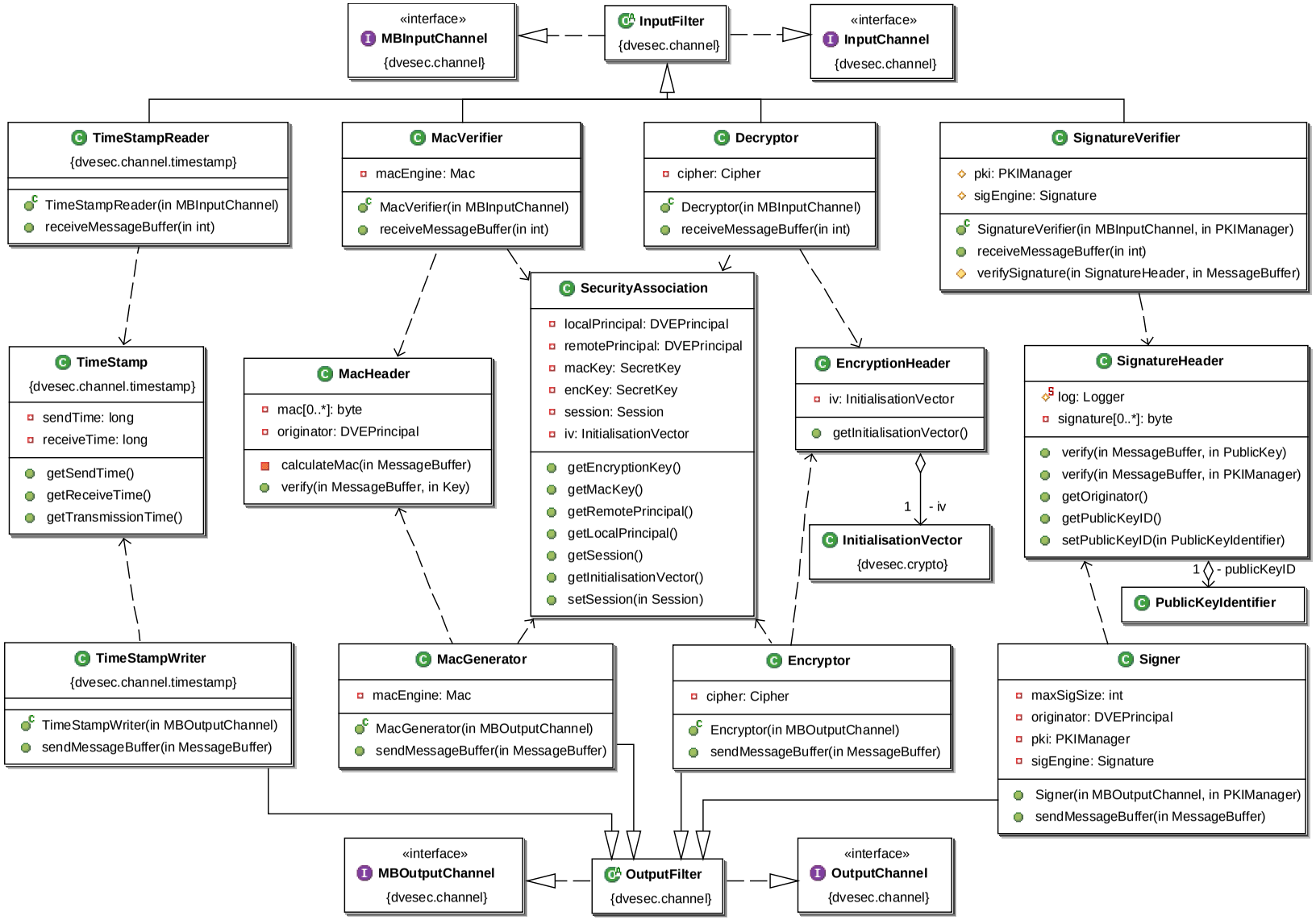



Abbildung 3.9: Klassendiagramm der implementierten Filter

Tabelle 3.10: Nachrichtenexpansion durch Filter in der Standardkonfiguration

| *Filterklasse* | TimeStamp | MacHeader | EncryptionHeader | SignatureHeader |
|---|---|---|---|---|
| *Größe [Byte]* | 8 | 20 | 4 | 158 |

ein Objekt der Klasse InitialisationVector repräsentiert, das in anderen Strom-Chiffrier-Modi den Initialisierungsvektor enthält.

Signer benutzt in der Standardkonfiguration RSASSA-PKCS1 [PKCS1] mit SHA-1 und einer Schlüssellänge von 1024 Bit. Jede Signatur ist dadurch 128 Byte lang. Die Klassen DVEPrincipal und PKIManager werden im Abschnitt 5.5.4 behandelt. Mit der Klasse PublicKeyIdentifier wird der öffentliche Schlüssel identifiziert, der zur Verifikation der Signatur signature eines SignatureHeader-Objektes notwendig ist, z.B. durch den Prinzipal und den SubjectPublicKeyIdentifier seines Zertifikats. Im serialisierten Zustand benötigt dieser Bezeichner 30 Byte.

Tabelle 3.10 zeigt die Größen der serialisierten Header-Objekte in der Standardkonfiguration.

### Aktive Filter

Alle bisher beschriebenen Eingabefilter arbeiten synchron, d.h. erst in dem Moment, in dem die Anwendung versucht, eine Nachricht zu empfangen, wird am anderen Ende der Filterpipeline der Kommunikationskanal abgefragt, und danach die eventuell empfangene Nachricht durch die Filterpipeline bearbeitet (vgl. Abbildung 3.8). Fallen dabei mehrere aufwändige Operationen an, steigt die Bearbeitungslatenz. Als Gegenmaßnahme wurde der CachingInputFilter entwickelt, der den Empfang und die Bearbeitung von Nachrichten im Hintergrund ermöglicht. Dazu benutzt er einen eigenen Thread, den ChannelListener, der permanent am Eingabekanal horcht und eventuell empfangene Nachrichten in eine Warteschlange einreiht.

Die Funktionsweise ist im Aktivitätsdiagramm 3.11 dargestellt. Da der ChannelListener mit einem blockierenden receive() empfängt, können andere Threads die Wartezeit ausfüllen. Für den asynchronen, nicht-blockierenden Empfang genügt ein einziger CachingInputFilter am Ende der Filterpipeline, und damit nur ein einziger zusätzlicher Thread. Für alle Ausnahmen, die der ChannelListener melden kann, besitzt der CachingInputFilter außerdem eine Callback-Methode, die erbende Unterklassen überschreiben können.

Sollen empfangene Nachrichten sofort verarbeitet werden, bietet sich die Vererbung der Klasse ActiveMessageProcessor (ohne Abbildung) an. Wie der CachingInputFilter benutzt er einen eigenen Thread, allerdings werden die empfangenen Nachrichten sofort in demselben Thread von der abstrakten Methode processMessage verarbeitet, anstatt in eine Warteschlange eingereiht zu werden. Außerdem arbeitet der ActiveMessageProcessor hinter dem MessageDecoder und verarbeitet daher Encodable-Objekte anstelle von MessageBuffer-Objekten. ActiveMessageProcessor erbt direkt von java.lang.Thread, sodass die Priorität von außen gesetzt werden kann.

## 3.4.5 Sitzungsverwaltung

UDP- und Multicast-Sockets können prinzipiell mit mehreren Partnern auf einmal kommunizieren(vgl. Abschnitt 3.3.3). Wir nennen diese Eigenschaft im Folgenden *Multi-Session-Fähigkeit*.

In Java können UDP-Sockets mit einer Socket-Adresse eines festen Zielrechners verbunden werden. In diesem Fall überwacht der SecurityManager, dass Datagramme nur an diese Adresse gesendet bzw. von dieser Adresse empfangen werden. Ansonsten meldet er eine Ausnahme. Dieses Konzept wurde speziell für *applets* eingeführt, die aus Sicherheitsgründen nur mit dem Rechner

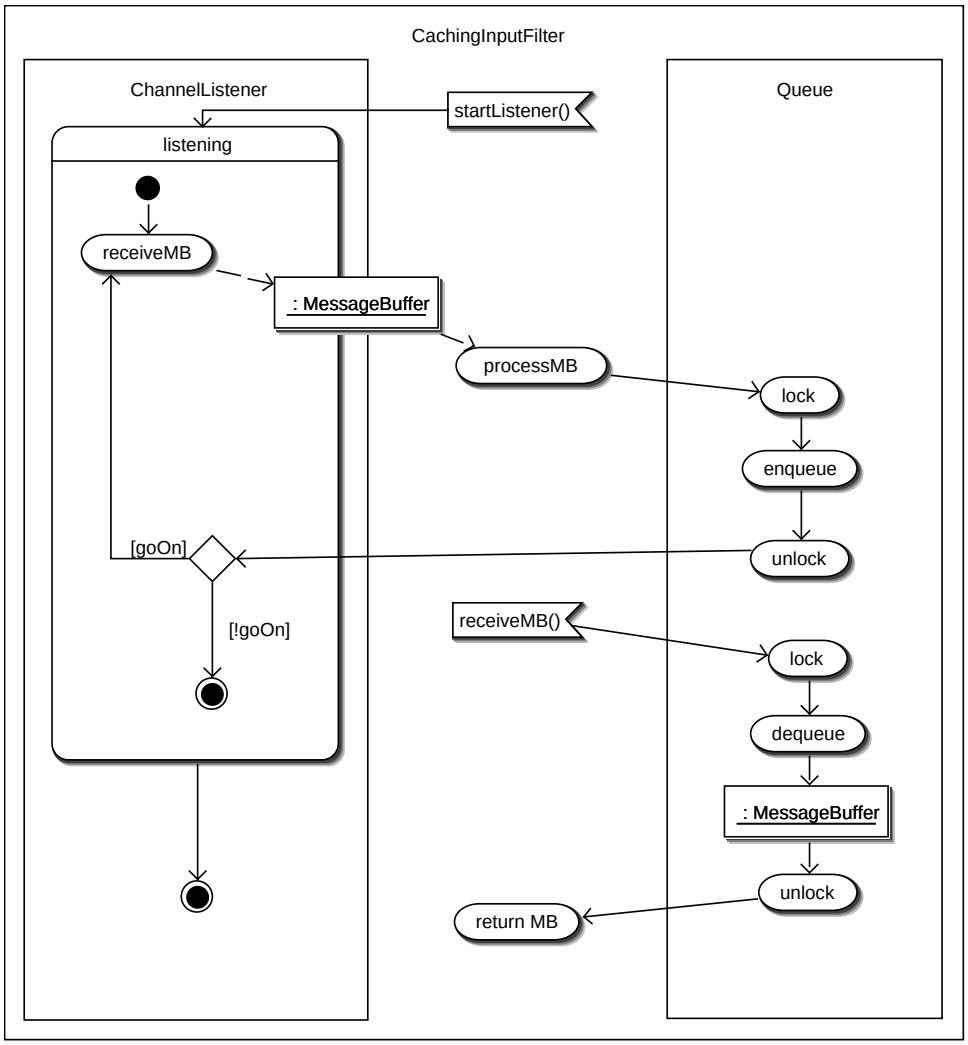



Abbildung 3.11: Aktivitätsdiagramm eines CachingInputFilter-Objekts

kommunizieren dürfen, von dem sie heruntergeladen wurden.

Die Sicherheit der „verbundenen Sockets“ und die Flexibilität der Multi-Session-Fähigkeit wurden im Kommunikationspaket dvesec.channel vereint. Das entscheidende Konzept ist die *Sitzungsverwaltung*.

Eine *Sitzung* (englisch *session*) speichert alle Daten zur Kommunikation mit einem bestimmten Rechner. Die Klassen der Sitzungsverwaltung sind in Abbildung 3.12 dargestellt. Zunächst einmal besitzt jeder Kommunikationskanal einen SessionManager. Dieser verwaltet alle aktuellen Session-Objekte. Jede Session hat mindestens einen Schlüssel, mit dem sie eindeutig in einer Hash-Tabelle referenziert werden kann. Für alle InetSessions ist der Schlüssel die Socket-Adresse des Kommunikationspartners[3], für die NioTCPSession kommt noch der Java-NIO-Kanal der Verbindung hinzu. Die Klasse GroupSession fasst mehrere Sitzungen zusammen. Ein Session-Objekt kann zusätzliche Parameter speichern, wie hier die SecurityAssociation (vgl. Abschnitt 4.6.1).

Vor dem Versand eines MessageBuffers wird dessen Session mit MessageBuffer.getSession() ermittelt und mit SessionManager.checkResolveSession(Session) überprüft. Ist keine Sitzung gesetzt, wird das Paket der Standardsitzung SessionManager.getDefaultSession() zugeordnet. Ist die

[3] genauer: die lokal gültige Adresse des Partners, um Probleme mit NAT (vgl. Abschnitt 4.5) auszuschließen.

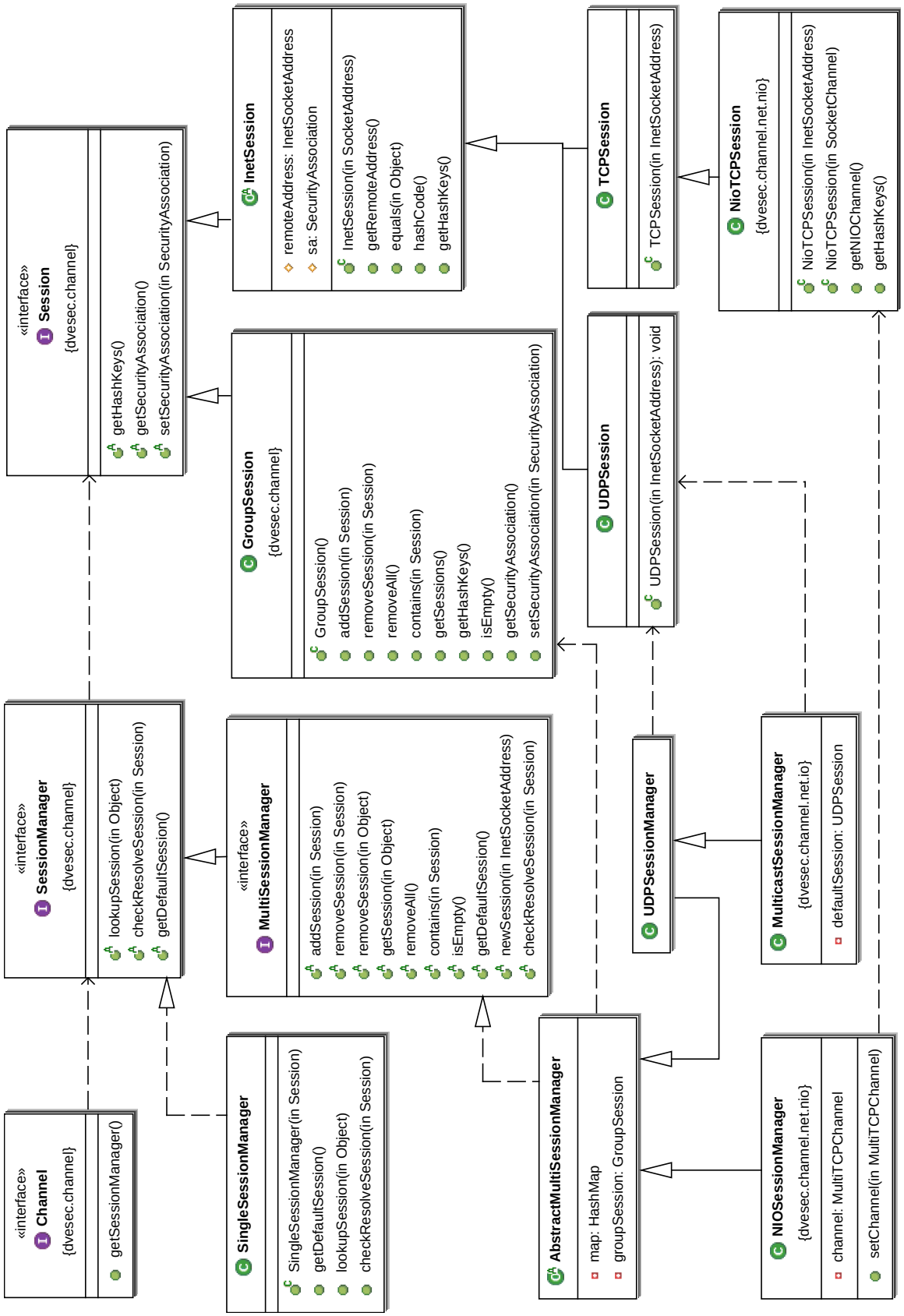



Abbildung 3.12: Klassendiagramm der Sitzungsverwaltung

Sitzung ungültig, wird eine Ausnahme gemeldet. Eine Anwendung kann alle Nachrichten adressieren, die das Interface Addressed implementieren, indem sie mit Addressed.setSession(Session) die Session setzt. Diese wird dann beim Serialisieren automatisch von der MessageBuffer-Klasse übernommen.

Auf der Empfängerseite wird mit SessionManager.lookupSession(Object) das zu einer Absenderadresse gehörige Session-Objekt nachgeschlagen. Wie bei Header-Objekten wird auf der Empfängerseite die Session auf deserialisierte Addressed-Nachrichten übertragen.

Es gibt zwei grundsätzlich verschiedene SessionManager: Den SingleSessionManager für fest verbundene Kanäle und den MultiSessionManager für Multi-Session-fähige Kanäle. Der AbstractMultiSessionManager ist eine Hilfsklasse, die die meisten Methoden des MultiSessionManager-Interfaces implementiert. Davon abgeleitet ist der UDPSessionManager für UDP-Sockets mit mehreren Partnern, der MulticastSessionManager für Multicast-Sockets und der NIOSessionManager. Der zuletzt Genannte bündelt mehrere Java-NIO-basierte TCP/IP-Sockets mithilfe eines Selektors (vgl. Abschnitt 3.4.3). Damit wird dieselbe Multi-Session-Funktionalität wie auf UDP-Sockets über TCP/IP-Sockets implementiert (vgl. Abschnitt 3.4.5). Eine dem Typ des Kommunikationskanals entsprechende Session kann mit MultiSessionManager.newSession(InetSocketAddress) erzeugt werden.

Ein UDPSessionManager, der noch keine Sitzungen speichert, befindet sich im *promiskuitiven Modus*. Er erlaubt den Empfang von Nachrichten von beliebigen Kommunikationspartnern. In diesem Modus erlaubt allerdings nur der MulticastSessionManager das Senden nicht adressierter Nachrichten. Sobald Sitzungen registriert sind, lehnen alle MultiSessionManager Nachrichten von nicht registrierten Partnern ab. Von der Anwendung nicht adressierte ausgehende Nachrichten werden dann immer der Standardsitzung zugeordnet, die der Gruppe aller registrierten Sitzungen entspricht.

Damit wurden im Sitzungskonzept folgende Punkte realisiert:

- Kommunikationsverbindungen können identifiziert und zugehörige Daten in Session-Objekten gespeichert werden. Dazu gehören auch die Daten der Sicherheitsassoziation (vgl. Abschnitt 4.6.1).

- Multi-Session-Kanäle können mehrere Kommunikationspartner auf einmal verwalten. Dadurch benötigt man auch nur eine Filterpipeline für alle Partner. Insbesondere benötigt man auch nur einen Thread für den asynchronen Empfang.

- MultiSessionManager überwachen und bestimmen, mit wem kommuniziert werden kann.

- Transparente Multi-Session-Kanäle für unterschiedliche Protokolle.

- Multi-Session-Kanäle im promiskuitiven Modus ermöglichen Registrierungs-Server.

- Nachrichten zum Versand können, müssen aber nicht adressiert werden. Ebenso können die Adressen empfangener Nachrichten an die Anwendung weitergegeben werden, müssen aber nicht.

- Der SessionManager eines Kanals wird von den nachfolgenden Filtern weitergereicht, und steht somit auch der Anwendung zur Verfügung.

### Multicast-Abstraktion

Es wurde bereits gezeigt, dass einerseits in DVEs oft am effizientesten über Multicasting kommuniziert werden kann, andererseits IP-Multicasting oft nicht verfügbar ist (vgl. Abschnitt 3.3.2).

Mit den Multi-Session-Kanälen wird das Multicasting-Prinzip daher abstrahiert. Die hier dargestellten Realisierungen basieren auf IP-Multicast-, UDP- und Java-NIO-TCP/IP-Sockets. Durch die Abstraktion können Anwendungen die geeignete Implementierung je nach Verfügbarkeit und unter Berücksichtigung zusätzlicher Anforderungen mit wenig Aufwand ändern. Über den assoziierten MultiSessionManager können neue Mitglieder zur Multicast-Gruppe hinzugefügt oder alte entfernt werden. Auf welche Weise sich neue Gruppenmitglieder anmelden, ist in der Abstraktion offen gelassen.

Benutzen alle Gruppenmitglieder einen Multi-Session-Kanal mit denselben registrierten Teilnehmern, entsteht ein Kanal für *Viele-an-Viele-Kommunikation*. Benutzt nur ein Teilnehmer einen Multi-Session-Kanal und die anderen Gruppenmitglieder den zugehörigen Unicast-Kanal, entsteht die Infrastruktur für *Einer-an-Viele-Kommunikation*.

Basiert ein Multi-Session-Kanal auf Unicast-Verbindungen und dem lokalen Versand aller Nachrichten an alle Gruppenmitglieder, sprechen wir von *emuliertem Multicasting*. Emuliertes Multicasting ist zwar keine sehr effiziente Art der Kommunikation, ist aber für kleine Gruppen in vielen Fällen ausreichend. Für den Fall, dass mehrere Mitglieder einen Multi-Session-Kanal benutzen, muss die Anwendung sicherstellen, dass alle dieselben Gruppenmitglieder registriert haben.

Die Integration eines der in Abschnitt 3.3.5 vorgestellten *application-layer-multicast-Protokolle* kann ebenfalls über die *Multi-Session-Kanäle* erfolgen. Im Gegensatz zum emulierten Multicasting müssen die Gruppenmitglieder hier nur ihre Nachbarn im Multicast-Baum registrieren, wodurch eine wesentlich effizientere Kommunikation entsteht.

## 3.4.6 Geschützte Nachrichtenteile

Die Kanalfilter sind dann besonders effektiv, wenn alle Nachrichten auf dieselbe Weise bearbeitet werden müssen. In vielen Anwendungen sind aber nicht alle über einen Kanal übertragenen Nachrichten, sondern nur Teile bestimmter Nachrichten mit kryptographischen Mitteln zu schützen. Bei vielen Schlüsselverhandlungsprotokollen wird beispielsweise in der ersten Phase ein Chiffrieralgorithmus verhandelt, mit dem die Nachrichten späterer Phasen chiffriert werden.

Daher habe ich die Klassen des Pakets dvesec.crypto entwickelt, mit denen sich einzelne Teile von Nachrichten kryptographisch schützen lassen. Abbildung 3.13 zeigt das dazugehörige Klassendiagramm. Die Entschlüsselung von EncryptedPart- oder PublicKeyEncryptedPart-Objekten und die Authentisierung von MacProtectedPart- oder SignedPart-Objekten muss durch die Anwendung erfolgen, sobald die geschützten Daten verwendet werden sollen. Die Daten werden dabei über den Encoder, den eine Methode getEncoder() liefert, serialisiert und geschützt, und mit einem Decoder, der mit getDecoder() erzeugt wird, geprüft und deserialisiert.

Abbildung 3.14 auf Seite 49 zeigt ein kleines Code-Beispiel zur Verdeutlichung: Ein Nachricht der Klasse ClassB enthält neben anderen Attributen einen MAC-geschützten String. Der String s ist kein explizites Attribut von ClassB. Vielmehr wird er vom Konstruktor im MacProtectedPart mpp abgelegt. Wie die Kanalfilter de-/serialisiert der MacProtectedPart die geschützten Attribute nur einmal. Die Überprüfung des MAC erfolgt erst durch die Anwendung, also auch nachdem alle anderen Attribute von ClassB deserialisiert worden sind. Auf diese Weise kann auch die Reihenfolge, in der beispielsweise ein Zeitstempel und eine MAC überprüft werden, von der Anwendung festgelegt werden.

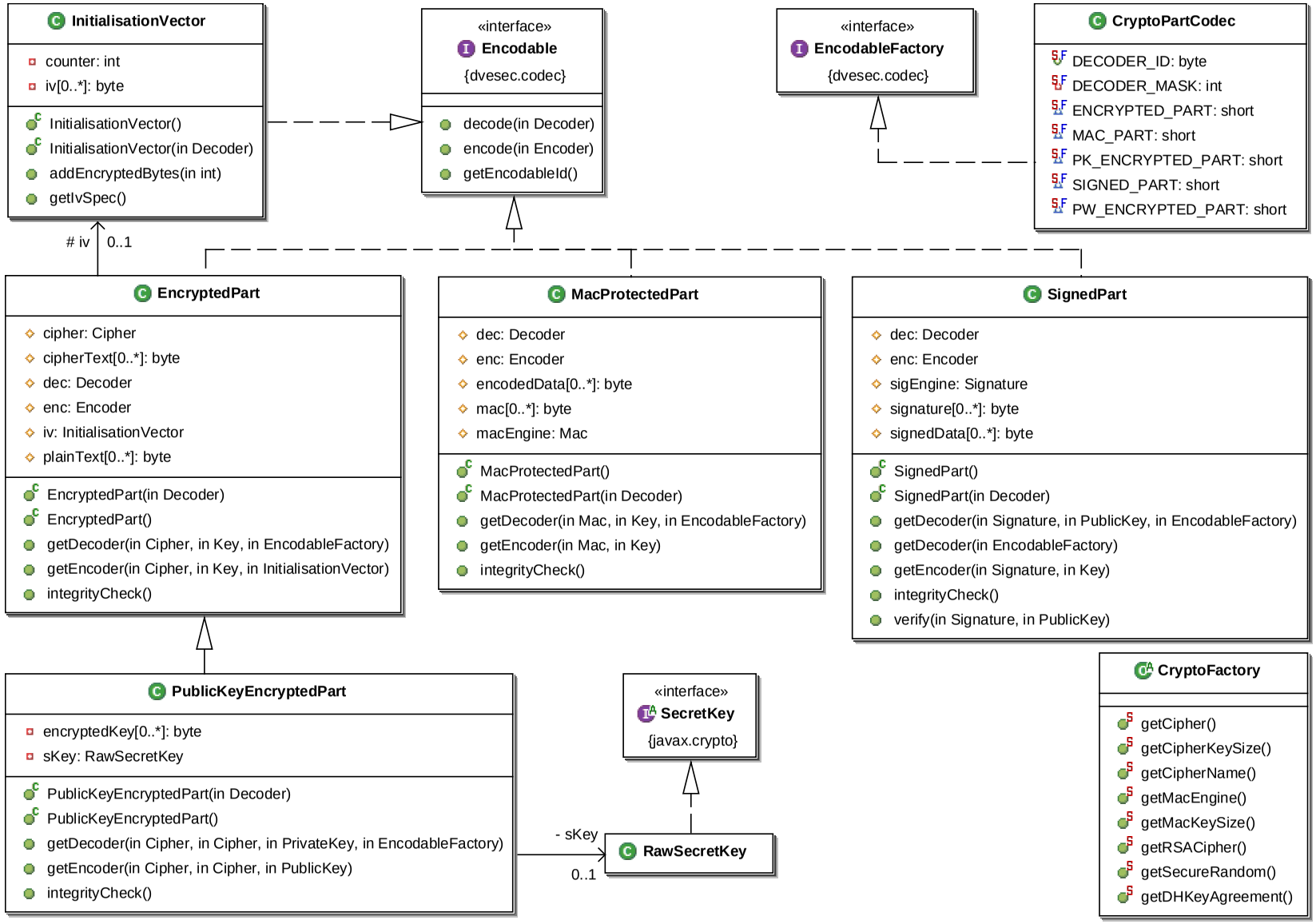



Abbildung 3.13: Klassendiagramm des dvesec.crypto-Pakets

```
1  public ClassB implements Encodable {
2    ...
3    private MacProtectedPart mpp;
4
5    public ClassB(int n, String s, Key key) {
6      this.n = n;
7      // initialisiere mpp
8      this.mpp = new MacProtectedPart();
9
10     // schreibe s in mpp
11     Encoder enc = mpp.getEncoder(CryptoFactory.getMacEngine(), key);
12     enc.writeString(s);
13     enc.close();
14   }
15
16   ...
17
18   public String getProtectedString(Key key) {
19      try {
20       // verifiziere mpp
21       Decoder dec = mpp.getDecoder(CryptoFactory.getMacEngine(), key, null);
22
23       // lies s aus mpp
24       return dec.readString();
25     } catch (AuthenticationException exc) {
26       System.out.println("Invalid MAC");
27       return null;
28     }
29
30   }
31
32   public void encode(Encoder enc) {
33     ...
34     mpp.encode(enc);
35   }
36
37   public void decode(Decoder dec) {
38     ...
39     mpp = new MacProtectedPart(dec);
40   }
41 }
```

Abbildung 3.14: MAC-geschütztes Attribut der Beispielklasse ClassB

Kryptographische Algorithmen, Schlüssel und eventuelle zusätzliche Parameter werden normalerweise beim Aufruf vom getEncoder() bzw. getDecoder() gesetzt. Als einzige Ausnahme ermöglicht SignedPart.getDecoder(EncodableFactory) das Lesen der geschützten Attribute bevor die Signatur mit verify(Signature, PublicKey) überprüft wird. Dadurch können geschützte Attribute zur Identifikation des öffentlichen Schlüssels, der zur Überprüfung benötigt wird, verwendet werden.

Die Methode integrityCheck() prüft jeweils, ob alle erforderlichen Attribute gesetzt sind, und meldet ansonsten eine Ausnahme. PublicKeyEncryptedPart verschlüsselt die geschützten Daten mit einer symmetrischen Chiffre, deren zufälliger Schlüssel mit einem asymmetrischen Verfahren und

dem öffentlichen Schlüssel des Empfängers verschlüsselt wird. Auf diese Weise können größere Datenmengen effizient chiffriert werden, ohne dass zuvor ein gemeinsamer symmetrischer Schlüssel ausgehandelt wurde. Diese Funktion wird bei vielen Schlüsseletablierungsprotokollen (vgl. Abschnitt 4.6.2) und Zertifizierungsprotokollen (vgl. Abschnitt 5.4.1) benötigt.

CryptoPartCodec ist die EncodableFactory für alle Encodable-Klassen des Pakets. CryptoFactory stellt einheitliche symmetrische kryptographische Algorithmen und deren Parameter für alle Klassen im Projekt *DveSec* zur Verfügung. Austausch der voreingestellten Algorithmen und Änderungen der Parameter werden hier vorgenommen.

# Kapitel 4

# Sicherheitstechnik

Die wissenschaftliche Disziplin, die sich mit der Sicherung von IT-Systemen beschäftigt, nennt man *Sicherheitstechnik* (englisch *security engineering*). Anderson beschreibt Sicherheitstechnik in [And01] wie folgt:

> *Security engineering is about building systems to remain dependable in the face of malice, error, or mischance. As a discipline, it focuses on the tools, processes, and methods needed to design, implement, and test complete systems, and to adapt existing systems as their environment evolves.*

Sicherheitstechnik umfasst unter anderem die Beschreibung von Sicherheitsanforderungen, die Mechanismen zur Durchsetzung dieser Anforderungen, die Detektion und Abwehr von Angriffen und die Wiederherstellung des Systems und seiner Sicherheitsanforderungen nach einem geglückten Angriff.

Diese Arbeit konzentriert sich auf die *Anwendung* von Sicherheitstechnik in verteilten virtuellen Umgebungen. Sie enthält daher keine neuen kryptographischen Algorithmen. Vielmehr wird versucht, vorhandene kryptographische Bausteine zum Aufbau einer Sicherheitsarchitektur für DVEs einzusetzen.

Zunächst wird in Abschnitt 4.1 der Begriff *Sicherheitstechnik* definiert. Abschnitt 4.2 beschreibt dann die Verbindung zwischen Sicherheitstechnik und Kryptographie. In Abschnitt 4.3 werden weitere Grundbegriffe der Sicherheitstechnik dargestellt, gefolgt von typischen Sicherheitsmechanismen in Abschnitt 4.4. Abschnitt 4.5 ist den Firewalls gewidmet. In Abschnitt 4.6 werden schließlich die für diese Arbeit relevanten Sicherheitsprotokolle vorgestellt.

## 4.1 Definition

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) definiert in [BSI03a] die *Grundwerte der IT-Sicherheit* als:

**Vertraulichkeit:** Vertrauliche Informationen müssen vor unbefugter Preisgabe geschützt werden.

**Verfügbarkeit:** Dem Benutzer stehen Dienstleistungen, Funktionen eines IT-Systems oder auch Informationen zum geforderten Zeitpunkt zur Verfügung.

**Integrität:** Die Daten sind vollständig und unverändert.

Die Aufgabe der Sicherheitstechnik ist, diese Grundwerte in einem System zu schützen. Die Gewichtung dieser drei Grundwerte sowie angemessene Methoden zu deren Schutz unterscheiden sich dabei von Anwendung zu Anwendung.

## 4.2 Kryptographie

Viele Sicherheitsmechanismen, die bei der Implementierung von Sicherheitsanforderungen eingesetzt werden, beruhen auf *Kryptographie*. Menezes et al. definieren Kryptographie in [MvOV97]:

> *Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication.*

Als fundamentale Dienste der Kryptographie nennen sie:

**Vertraulichkeit:** Der Inhalt einer Information darf nur denen zugänglich sein, die dazu berechtigt sind.

**Datenintegrität:** Der Schutz vor unerlaubter Datenmanipulation.

**Authentisierung:** Die Identifizierung von Entitäten und Datenherkunft.

**Nicht-Abstreitbarkeit:** Mechanismen zur Verhinderung, dass Entitäten frühere Handlungen abstreiten können.

Von diesen Diensten können weitere Dienste abgeleitet werden, z.B. Schutz der Privatsphäre, Geheimhaltung, Nachrichtenauthentisierung, digitale Signaturen, Autorisierung und Zugangskontrolle, Zertifizierung, Erzeugung von Zeitstempeln, Bezeugung, Empfangsbestätigung, Bestätigung, Eigentumssicherung, Anonymität und Widerruf von Authentisierung oder Zertifizierung.

Zwei der drei Grundwerte der IT-Sicherheit können also durch kryptographische Methoden geschützt werden. Bei der Sicherung der Verfügbarkeit ist man allerdings auf andere Methoden angewiesen, wobei kryptographische Mechanismen, z.B. die Authentisierung, auch eine Nebenrolle spielen können.

### 4.2.1 Verschleierung, Sicherheit und Komplexität

Zunächst soll kurz der Unterschied zwischen Sicherheit und Verschleierung erklärt werden.

*Verschleierung* bedeutet, dass die Sicherheit eines kryptographischen Algorithmus' allein auf der Geheimhaltung seiner Funktionsweise beruht. Sobald der Algorithmus im Detail bekannt wird, bricht die Sicherheit des gesamten Systems zusammen [MN03]. Kerckhoffs hat dieses Problem bereits im Jahr 1883 erkannt (Prinzip von Kerckhoffs, [Ker83]). Je größer das Interesse an den geschützten Informationen ist, umso größer ist die zu erwartende Zahl der Hacker, die das Geheimnis zu lüften versuchen, und über das Internet lassen sich Informationen oder Hilfswerkzeuge, z.B. Lizenznummer-Generatoren, sehr schnell verbreiten.

Daher lautet einer der Leitsätze der Sicherheitstechnik: Die Sicherheit eines Systems darf nicht allein auf Verschleierung beruhen. Stattdessen spielt die *Berechenbarkeit* eine entscheidende Rolle.

Ein kryptographischer Algorithmus ist sicher, wenn der beste erfolgreiche Angriffs-Algorithmus so komplex ist, dass er nicht mit einem akzeptablen Aufwand an Ressourcen, z.B. Zeit, Speicherplatz oder Kommunikationsbandbreite, durchgeführt werden kann. Man spricht in diesem Fall von

einem *praktisch unmöglichen Angriff* (englisch *computationally unfeasible attack*), was also nicht bedeutet, dass er absolut unmöglich ist, sondern vielmehr zu aufwändig.

Eine übliche Messgröße für die Sicherheit eines kryptographischen Algorithmus ist daher eine Kostenschätzung für den besten Angriff, z.B. Kosten für den Bau eines speziell angepassten Rechners, für gemietete Rechenzeit oder für Speicherkapazitäten. Sind diese Kosten niedriger als der Wert der geschützten Informationen für den Angreifer, eignet sich der Algorithmus nicht.

Was heute noch als praktisch unberechenbar gilt, könnte in naher Zukunft aufgrund technologischer Weiterentwicklungen mit vertretbarem Aufwand berechenbar werden. Dabei spielen sowohl Fortschritte auf theoretischem Gebiet, z.B. neue kryptanalytische Methoden, als auch die ständig wachsende Leistung moderner Rechnersysteme und die sinkenden Preise vorhandener Hardware eine Rolle. Insbesondere bei Informationen, die über einen langen Zeitraum vertraulich bleiben sollen, muss der technologische Fortschritt berücksichtigt werden.

### 4.2.2 Kryptographische Schlüssel

Wie im vorangegangenen Abschnitt dargelegt wurde, sollte die Vertraulichkeit einer Nachrichtenübertragung nicht allein auf der Geheimhaltung des Algorithmus begründet werden. Die meisten kryptographischen Operationen benutzen deshalb einen oder mehrere geheime *Schlüssel* als zusätzliche Parameter. Die resultierende Sicherheit hängt von der Geheimhaltung mindestens eines Teils des Schlüsselmaterials ab. Ein geheimer Schlüssel, der einem unberechtigten Dritten in die Hände fällt, garantiert keine Sicherheit mehr.

Eine obere Schranke für die Komplexität eines Angriffs auf einen kryptographischen Schlüssel ist die Anzahl der möglichen Schlüssel: In einem *brute-force-Angriff* probiert ein Angreifer einfach alle möglichen Schlüssel aus, bis er den richtigen darunter findet. Komplexität und Kosten dieses Angriffs müssen den Wert des Schlüssels für den Angreifer bei weitem übertreffen (vgl. Abschnitt 4.2.1). Die Komplexität eines *brute-force-Angriffs* hängt vorwiegend von der Größe des Suchraums und vom Aufwand einer einzelnen Anwendung des Algorithmus ab. Die Größe des Suchraums wird wesentlich von der *Schlüssellänge* bestimmt. Leitlinien zur Bestimmung der benötigten Schlüssellängen finden sich in [LV01]. Die zugehörige Website bietet auch ein Berechnungsformular, mit dessen Hilfe empfohlene Schlüssellängen für verschiedene Algorithmen in Abhängigkeit von Lebensdauer, Sicherheitsniveau und anderen Parametern ermittelt werden können.

Zur Schlüsselgenerierung werden (Pseudo-)Zufallszahlgeneratoren eingesetzt, damit Angreifer die neuen Schlüssel nicht vorhersehen können. Siehe hierzu auch Abschnitt A.3.

Schon allein wegen des technologischen Fortschritts (vgl. Abschnitt 4.2.1) müssen kryptographische Schlüssel eine begrenzte Gültigkeitsdauer haben. Dabei muss berücksichtigt werden, dass der Wert eines Schlüssels mit der Menge der durch ihn geschützten Daten steigt. Viele kryptographische Protokolle ermöglichen die Änderung des Schlüssels zur Laufzeit (englisch *re-key*). Nicht mehr benötigtes Schlüsselmaterial muss sofort zerstört werden, um zu verhindern, dass es Dritten in die Hände fällt.

Da nicht auszuschließen ist, dass verschiedene Angriffe auf unterschiedliche kryptographische Algorithmen effizient kombiniert werden können, sollte derselbe Schlüssel nie in zwei unterschiedlichen kryptographischen Operationen benutzt werden.

Für einige kryptographische Algorithmen existieren so genannte *schwache Schlüssel* (englisch *weak keys*) [CN94]. Die Verwendung dieser Schlüssel ermöglicht spezielle besonders einfache Angriffe. Diese Schlüssel dürfen nicht verwendet werden. Ein Sicherheitssystem muss alle intern und extern generierten Schlüssel daraufhin prüfen und gegebenenfalls ablehnen.

Nicht mehr benötigtes Schlüsselmaterial muss gelöscht werden, bevor es Unberechtigten in die Hände fallen kann. Außerdem sollten Schlüssel nie im Klartext in permanentem Speicher, z.B. auf einer Festplatte, abgelegt werden. Langzeitschlüssel können beispielsweise mit einem Passwort geschützt werden, das ein Benutzer dann zur Verwendung des Schlüssels eingeben muss.

## 4.3 Grundbegriffe der Sicherheitstechnik

In diesem Abschnitt werden die wichtigsten Begriffe der Sicherheitstechnik eingeführt.

### 4.3.1 Prinzipale, Rollen und Gruppen

Ähnlich wie Anderson [And01] definieren wir die Begriffe für die Akteure eines Sicherheitssystems wie folgt:

**Subjekt:** Eine reale Person, die in einem Sicherheitssystem agiert.

**Prinzipal:** (englisch *principal*) Eine Entität, die an einem Sicherheitssystem teilnimmt. Das kann ein Subjekt, eine Rolle, ein Kommunikationskanal, ein Teil der Ausrüstung, z.B. ein PC, eine Smartcard oder ein Kartenlesegerät, sein, genauso wie eine Gruppe, eine Verbindung, eine zusammengesetzte Rolle oder die Delegation eines Prinzipals.

**Rolle:** (englisch *role*) Eine Funktion im System, die von verschiedenen Prinzipalen ausgeübt werden kann, z.B. die des Gruppenkoordinators. Die Zuordnung, welche Prinzipale welche Rolle einnehmen, kann sich über die Zeit ändern.

**Gruppe:** (englisch *group*) Eine Menge von Prinzipalen. Auch die Zusammensetzung von Gruppen kann sich zur Laufzeit verändern.

Andere Definitionen sind möglich. In der *Java Cryptographic Architecture* (JCA) umfasst ein Subject-Objekt ein oder mehrere Principal-Objekte und dazugehörige Informationen wie kryptographische *Berechtigungsnachweise* (englisch *credentials*). Group-Objekte sind Gruppen von Prinzipalen in einer Zugangskontrollliste. *Rollen* existieren in dieser Form nicht.

### 4.3.2 Vertrauen

Sicherheit existiert nicht ohne *Vertrauen*: Nur jemand oder etwas, dem ich vertraue, kann mir Sicherheit geben. Sicherheit ist also immer im Vertrauen auf etwas oder jemanden verankert; gleichzeitig setzt die Sicherheitstechnik dort an, wo das Vertrauen aufhört.

Denn auch IT-Sicherheit setzt Vertrauen voraus: Der Sender einer E-Mail vertraut dem Hersteller des Mail-Programms, dass die Nachricht in unveränderter Form nur an den Empfänger gesendet wird. Ebenso traut er der Hardware seines Rechners und dem Betriebssystem, dem Internet-Provider und dem Internet selbst. Schneier schreibt in [Sch00]: „*There is no security system that is devoid of trust; even the person who writes his own security software has to trust his compiler and computer.*“ Der erste Schritt in einer Sicherheitsanalyse muss daher eine Vertrauensanalyse sein: Welche Vertrauensbeziehungen werden zwischen den verschiedenen Prinzipalen, Rollen oder Gruppen angenommen?

Im Kontext der sicheren Betriebssysteme spricht man von der *trusted computing base*, die als die Menge der Komponenten eines Systems (Hardware, Software, Menschen etc.) definiert ist, deren korrektes Funktionieren die Durchsetzung der Sicherheitsrichtlinie garantiert [Dep85].

### 4.3.3 Angriffe

Dort wo Vertrauen aufhört, beginnen die *Angriffsszenarien*. Jedes System hat eine charakteristische Familie von Angriffen. Mit einem Angriff verbinden sich dabei folgenden Begriffe:

**Angreifer:** Ein an der aktiven Durchführung des Angriffs beteiligtes Subjekt.

**Intention:** Die Absicht des Angreifers.

**Angriffsmethode:** Die Methode, die zur Durchführung des Angriffs eingesetzt wird.

**Angriffspunkt:** Die Stelle im System oder die Eigenschaft des Systems, die angegriffen wird.

**Wirkungsbereich:** Die Menge der vom Angriff betroffenen Teilnehmer.

Gegenmaßnahmen zur Sicherung des Systems gegen Angriffe können *präventiv* (englisch *proactive*) oder *reaktiv* (englisch *reactive*) sein. Präventive Maßnahmen machen den Angriff bereits im Voraus undurchführbar, während reaktive Maßnahmen auf einen entdeckten Angriff reagieren. Im letzten Fall benötigt man zusätzliche Mechanismen zur *Detektion* eines Angriffs. Nicht alle Angriffe sind vorhersehbar. Daher enthalten viele Systeme Maßnahmen zur *Wiederherstellung* (englisch *recovery*) nach einem Angriff. Die Reaktion kann über den eigentlichen Angriff hinausgehen, z.B. durch den Ausschluss eines Teilnehmers aus dem Verbund oder durch die Sperrung eines Zugangs.

Der Schwerpunkt der traditionellen Sicherheitsanalyse liegt auf der *Prävention* von Angriffen. In den meisten Fällen entsteht dadurch ein für reale Anwendungen unbrauchbares Alles-oder-Nichts-Sicherheitsmodell. Schneier führt in [Sch00] die Verschlüsselung einer Verbindung als Beispiel an: Sie verhindert zwar den Angriff, nämlich das Abhören der Verbindung; knackt ein Angreifer aber die Chiffrierung, gibt es für die Kommunikationspartner weder die Möglichkeit der Detektion noch der Reaktion. Entwickler von Sicherheitssystemen können nie alle möglichen Eventualitäten und Angriffsszenarien im Voraus kennen, und selbst wenn es einen Schutz gegen *alle* möglichen Angriffe gäbe, wäre er vermutlich viel zu teuer oder würde das System unbenutzbar machen. Schneier schlägt daher vor, Sicherheitstechnik eher als *Risikomanagement* zu verstehen. Sichere Systeme müssen sowohl Mechanismen zur *Prävention* als auch zur *Detektion*, *Reaktion*, sowie zur *Wiederherstellung* nach einem Angriff haben.

Die unsicherste Stelle eines IT-Systems ist oft der menschliche Anwender. Die Angriffe mittels *sozialer Manipulation* (englisch *social engineering*) sind in Systemen aller Art verbreitet. Die stärksten Sicherheitsmechanismen nützen nichts, wenn die Benutzer nicht mitspielen, und beispielsweise Passphrasen offen neben dem Arbeitsplatzrechner deponieren oder gar bestechlich sind. Die Gesamtsicherheit eines Systems kann nie allein durch eine Software- oder Hardwarelösung gewährleistet werden.

### 4.3.4 Sicherheitsrichtlinie

Die *Sicherheitsrichtlinie* (englisch *security policy*) eines Systems beschreibt, welche Sicherheitsfunktionen ein System haben soll. Die Sicherheitsrichtlinie stellt für die Sicherheitsfunktionen

des Systems das dar, was die Systemspezifikation für die generelle Funktionalität ist. Oft enthält die Sicherheitsrichtlinie nur die Beschreibung, welche Benutzer welche Funktionen des Systems nutzen dürfen. In einem *top-down*-Systementwurf wird die Sicherheitsrichtlinie als Ergebnis einer Bedrohungsanalyse aufgestellt, und regelt ihrerseits den Entwurf von Sicherheitsmechanismen (vgl. [And01]).

Die Sicherheitsrichtlinie schafft Verbindlichkeit zwischen Entwicklern, Betreibern und Benutzern von DVEs. Schreibt man die Rechte und Pflichten der Prinzipale vertraglich bindend in der Sicherheitsrichtlinie fest, so kann das Risiko von Insider-Angriffen im Schadensfall auch jenseits von IT-Mechanismen, z.B. durch ein gerichtliches Verfahren, kontrolliert werden. Mögliche Sanktionen bei Verstößen gegen die Sicherheitsrichtlinie werden ebenfalls in der Richtlinie aufgeführt.

## 4.4 Sicherheitsmechanismen

Die in der Sicherheitsrichtlinie (vgl. Abschnitt 4.3.4) festgelegten Sicherheitsfunktionen eines Systems werden unter anderem durch die Implementierung von *Sicherheitsmechanismen* durchgesetzt, die das Thema dieses Abschnitts sind.

### 4.4.1 Eindeutige Bezeichner

Die im Abschnitt 4.3.1 beschriebenen Prinzipale, Rollen und Gruppen müssen in einem Sicherheitssystem eindeutig bezeichnet werden. Personennamen sind in der Regel nicht global eindeutig; beispielsweise tragen bereits in meinem Bekanntenkreis mehrere Personen den Namen „Jan“ oder „Köhnlein“. Datenbanken, in denen die Daten vieler Personen gespeichert werden, benutzen daher computergenerierte *eindeutige Schlüssel* (Sozialversicherungsnummer, Personalausweis-ID usf.). Ein solcher Schlüssel ist lokal eindeutig und als Paar (Schlüssel, Namensraumbezeichner) global eindeutig, wenn der *Namensraumbezeichner*, z.B. „Personalausweisnummer der Bundesrepublik Deutschland“, global eindeutig ist.

Die Vergabe von eindeutigen Bezeichnern wird in vielen Systemen zentral vorgenommen, um versehentliche Kollisionen (Nicht-Eindeutigkeiten) zu vermeiden. In NPSNET-V war dafür ein zentraler ID-Server zuständig, obwohl das System ansonsten nach dem *Peer-to-Peer*-Paradigma entworfen war.

Die vielerorts verwendeten *global eindeutigen Bezeichner* (englisch *Globally Unique IDentifier*) (GUID) beruhen auf Hashfunktionen (vgl. Abschnitt A.2). Als GUID für einen Rechner dient z.B. ein Hashwert einer eindeutigen Beschreibung seiner Hardwarekonfiguration. Dieser Hashwert identifiziert den Rechner, ohne die Details der Konfiguration preiszugeben. Die *globale Eindeutigkeit* ist hier trotz des Namens nicht garantiert, sondern nur höchst wahrscheinlich.

Alternativ kann der Namensraum im Voraus aufgeteilt werden. Eine zentrale Stelle delegiert dann jeden einzelnen Bereich an genau einen Bezeichnerverteiler. Ein solches System wird bei der Vergabe der global eindeutigen MAC-Adressen für Ethernetadapter durch die *IEEE Registration Authority* benutzt: Die ersten 24 Bit identifizieren eindeutig den Hersteller, der die restlichen 24 Bit eigenständig und eindeutig den hergestellten Geräten zuordnet. Globale Eindeutigkeit der MAC-Adressen ist so garantiert. Allerdings muss die Größe der lokalen Bereiche im Voraus auf einen möglichst günstigen Wert festgelegt werden: Der Namensraum wäre schlecht genutzt, wenn die meisten Hersteller nur 10 Adressen vergeben würden. Einigen Herstellern ist übrigens der zugewiesene Namensraum von $2^{24} \approx 16,8$Mio Adressen zu klein, und sie besitzen daher mehrere Hersteller-IDs.

### 4.4.2 Authentifizierung und Authentisierung

Das englische Wort *authentication* hat im Deutschen unglücklicherweise zwei Bedeutungen: *Authentisierung* und *Authentifizierung*. Dabei bedeutet *Authentisierung „die Vorlage eines Nachweises eines Kommunikationspartners, in dem bestätigt wird, dass er tatsächlich derjenige ist, der er vorgibt zu sein“*, während *Authentifizierung „die Prüfung einer Authentisierung, d.h. die Überprüfung, dass ein Kommunikationspartner tatsächlich derjenige ist, der er vorgibt zu sein“* bezeichnet [BSI03b]. Ein Benutzer *authentisiert* sich also gegenüber einem System, das ihn *authentifiziert*. Oft sind diese beiden Begriffe allerdings schwer zu trennen. Im Folgenden wird daher der Terminus *Authentisierung* bevorzugt, wenn es sich nicht explizit um die beschriebene Funktion des Systems handelt.

Die Authentisierung ist die erste Phase einer sicheren Kommunikation. Der vertrauliche Austausch von Daten ist meist nicht sinnvoll, solange man nicht weiß, mit wem man kommuniziert. Durch die Authentifizierung werden Benutzer, Software- oder Hardwarekomponenten zu *Prinzipalen* (vgl. Abschnitt 4.3.1). Während der Authentifizierung wird oft auch die *Sicherheitsassoziation* einer Sitzung verhandelt, was Thema des Abschnitts 4.6.1 sein wird.

#### Authentisierung von Benutzern

Ein IT-System kann Menschen nicht in der Art erkennen, wie Menschen das untereinander tun. Authentifizierung von Benutzern durch ein IT-System erfolgt daher mittels

**Wissensnachweis:** Der Benutzer kennt ein Geheimnis, z.B. ein Passwort oder eine Geheimzahl.

**Besitznachweis:** Der Benutzer besitzt einen speziellen Gegenstand, z.B. eine Smartcard oder einen Ausweis.

**Biometrische Eigenschaften:** Für den Benutzer eindeutige biologische Merkmale, z.B. ein Fingerabdruck oder das Muster seiner Retina.

Oft wird auch eine Kombination aus diesen Methoden benutzt.

Menschen können sich im Allgemeinen geheime kryptographische Schlüssel, also lange komplexe Zeichenketten oder große Zahlen, nur sehr schlecht merken. Hängt der Zugang zu einem System von der Kenntnis eines kryptographischen Schlüssels ab, muss daher eine von den Benutzern bedienbare Schnittstelle gefunden werden. *Passwörter* sind wohl die bekannteste Authentisierungsmethode: Der Benutzer gibt seinen Benutzernamen und ein geheimes Passwort ein, das System überprüft dieses Paar und erlaubt den Zugriff bei erfolgreicher Verifikation. Fast jedes Buch über Sicherheit beschreibt ausführlich die Probleme bei der Verwendung von Passwörtern [And01, Sch96, Sch00]. Die wichtigsten Konsequenzen lassen sich wie folgt zusammenfassen:

- Passwörter dürfen nie im Klartext gespeichert oder über eine unsichere Verbindung übertragen werden.
- Das System muss leicht vorhersagbare oder zu kurze Passwörter ablehnen, da sonst *Wörterbuchangriffe* möglich werden. Passwörter sollten in keinem Wörterbuch zu finden sein und nicht nur aus Buchstaben, sondern auch aus Zahlen und Sonderzeichen bestehen.
- Benutzern muss klargemacht werden, dass Passwörter sicherheitsrelevante geheime Daten sind. Da Menschen sich sichere Passwörter nur schwer merken können, schreiben sie sie häufig auf und deponieren sie an für Unberechtigte leicht zugänglichen Orten. Das muss genauso vermieden werden wie die Weitergabe von Passwörtern.

- Werden nur Hashwerte von Passwörtern gespeichert, sollte man *salt-Werte* zum Erschweren eines Wörterbuchangriffs einsetzen.
- Auch verschlüsselte Dateien, die Passwörter enthalten, sind vertraulich.

Der Trend geht inzwischen zu *Passphrasen* (englisch *passphrases*), also zu ganzen Sätzen statt kurzer Zeichenketten. Geschriebene Sprache birgt allerdings eine relativ geringe Entropie (vgl. Abschnitt A.3). Es wird angenommen, dass ein englischer Text ca. 1,5 Bit Entropie pro Buchstabe [TC96] enthält. Um also mit einer Einwegfunktion aus einer englischen Passphrase einen möglichst unvorhersagbaren 128-Bit-Schlüssel, also einen mit maximaler Entropie, zu erzeugen, muss die Passphrase mindestens 85 Buchstaben haben (vgl. [Sch00]).

*Zugangsausweise* (englisch *access tokens*) sind Gegenstände zur Authentisierung. Sie sind entweder passiv, d.h. reine Datenspeicher, oder aktiv, also in der Lage, selbst Daten zu verarbeiten. Typische passive Zugangsausweise sind konventionelle Schlüssel, Magnetkarten oder maschinenlesbare Ausweise. Aktive Zugangsausweise sind z.B. Smartcards, die einen Chip enthalten, der Berechnungen, z.B. für eine digitalen Signatur, durchführen kann.

*Biometrie* ermöglicht schließlich die Authentisierung durch Messung biologischer Merkmale. Abgetastet werden dabei Fingerabdrücke, die Regenbogenhaut oder die Stimme des Benutzers.

Jeder der hier aufgeführten Mechanismen zur Benutzerauthentifizierung hat seine Stärken und Schwächen. Eine ausführlichere Beschreibung findet sich beispielsweise in den Büchern von Anderson [And01] oder Schneier [Sch00].

### 4.4.3 Autorisierung und Zugangskontrolle

Laut [BSI03a] ist *Autorisierung „die Prüfung, ob eine Person, IT-Komponente oder Anwendung zur Durchführung einer bestimmten Aktion berechtigt ist“. Zugangskontrolle* ist die Autorisierung für den generellen Zugang zum System.

Autorisierung erfordert in den meisten Fällen, dass sich der Benutzer authentisiert hat, also seine Identität bewiesen hat und vom System als Prinzipal verwaltet werden kann. In vielen Systemen verschmelzen Authentifizierung und Zugangskontrolle zu einer Phase; beispielsweise ist ein Benutzer zur Verwendung eines Systems autorisiert, nachdem er sich dort erfolgreich eingeloggt hat. Für bestimmte Aktionen können aber zusätzliche Berechtigungen erforderlich sein.

In der Sicherheitsrichtlinie wird definiert, welche Aktionen von bestimmten Rollen, Prinzipalen oder Gruppenmitgliedern durchgeführt werden dürfen. Diese *Berechtigungen* werden oft in *Zugangskontrolllisten* (englisch *Access Control Lists*) (ACLs) realisiert. Jeder Eintrag in einer ACL definiert einen Aktionstyp, den ein Prinzipal durchführen darf. Einige Berechtigungen schließen automatisch andere mit ein, und manche Aktionen erfordern eine spezielle Kombination unterschiedlicher Berechtigungen. Oft werden die Aktionstypen abhängig von den verwendeten Objekten definiert, wie beispielsweise die Zugriffsrechte auf einzelne Dateien in einem Dateisystem.

Besteht das System aus mehreren Komponenten mit unterschiedlichen Autorisierungsmechanismen, so müssen die Berechtigungen in der Sicherheitsrichtlinie für alle diese Mechanismen beschrieben werden. Es muss beispielsweise definiert werden, welche Prinzipale welche Einträge in einem LDAP-Verzeichnis lesen oder schreiben dürfen.

### 4.4.4 Schutz des Nachrichtenverkehrs

Nachrichten können bei der Übertragung durch zufällige Fehler oder durch einen mutwilligen Eingriff verfälscht werden. Um die Integrität empfangener Nachrichten sicherzustellen, können die in

Abschnitt 3.2.3 vorgestellten Techniken verwendet werden.

Jede Netzwerknachricht enthält außerdem mehr oder weniger zuverlässige Informationen, die sie einer speziellen Sitzung zuordnen, beispielsweise die IP-Adresse des Absenders. Eine höhere Zuverlässigkeit bieten *Message Authentication Codes* (MACs, Abschnitt A.6.1) oder *digitale Signaturen* (Abschnitt A.6.2). Ein MAC setzt einen zuvor vereinbarten geheimen Schlüssel voraus, während digitale Signaturen meist eine externe *Public-Key-Infrastruktur* (vgl. Abschnitt 4.4.6) zur Verteilung der öffentlichen Schlüssel verwenden. Beide Mechanismen bieten natürlich keinerlei Schutz, wenn der MAC oder die digitale Signatur vom Empfänger nicht geprüft werden.

Verschlüsselung ermöglicht vertraulichen Nachrichtenaustausch über öffentliche Netzwerke. Die Sicherheit durch Verschlüsselung beruht auf der Tatsache, dass nur die kommunizierenden Parteien den Schlüssel kennen und Dritte keine Informationen über den Schlüssel erhalten können. Zur Verschlüsselung großer Nachrichtenmengen werden aus Leistungsgründen symmetrische Chiffren benutzt (vgl. Abschnitt B.6).

Allerdings hat auch die Chiffrierung ihre Grenzen: Sorglos an Dritte weitergegebene Schlüsselinformationen lassen Unbefugte unbemerkt mithören. Auch kann man durch Verschlüsselung allein nicht verbergen, dass eine Kommunikation stattfindet, was in manchen Fällen bereits eine geheime Information darstellt. Ebenso haben die verschlüsselten Nachrichten meist eine ähnliche Länge wie der zugehörige Klartext, was ebenfalls Rückschlüsse auf den Inhalt zulässt. Für die hier betrachteten DVEs gelten allerdings keine so strengen Sicherheitsanforderungen.

Verschlüsselung allein schützt die Integrität einer Nachricht in vielen Fällen nicht. In vielen Chiffren kann man durch gezielte Änderungen des Chiffretexts den Klartext verändern, ohne dass das beim Entschlüsseln bemerkt wird. Verschlüsselung macht also die Authentisierung nicht überflüssig.

Werden Nachrichten in unterschiedlichen Schritten verschlüsselt und authentisiert, stellt sich die Frage nach der Reihenfolge der beiden Operationen. Bellare et al. zeigen in [BN00], dass die einzige *beweisbar sichere* Methode ist, zuerst zu chiffrieren und dann den MAC des Chiffretextes zu bilden. Ansonsten lassen sich Rückschlüsse auf einzelne Bits des Klartexts ziehen[1]. Ein weiterer Vorteil dieser Reihenfolge ist, dass ein Empfänger Nachrichten mit ungültigem MAC sofort aussortieren und dadurch auf die aufwändige Entschlüsselung verzichten kann.

Die unterschiedlichen kryptographischen Bausteine zum Schutz des Nachrichtenverkehrs werden im Anhang A genauer beschrieben. Zeitmessungen der Java-Implementierungen zeigt Abschnitt B.6.

### 4.4.5 Revisionsaufzeichnung

Oft werden Angriffe auf ein Sicherheitssystem erst zu einem späteren Zeitpunkt detektiert. Dazu muss man den Zustand des Systems zum Angriffszeitpunkt und die Auswirkungen des Angriffs später reproduzieren können. Alle wichtigen Aktionen des Systems sowie Anfragen von außen werden daher in der *Revisionsaufzeichnung* (englisch *audit log*) protokolliert. Jeder Eintrag erhält dabei zusätzlich einen Zeitstempel.

Durch die Analyse dieser Aufzeichnungen können Schwachstellen des Systems ausfindig gemacht und Angreifer oder Angriffsmuster identifiziert werden. Oft bietet die Revisionsaufzeichnung die einzige Möglichkeit, einen Angriff überhaupt festzustellen.

Der Zugang zur Revisionsaufzeichnung muss ebenfalls kontrolliert werden, da Angreifer sonst die sie entlarvenden Einträge manipulieren oder Phantomangriffe anderer protokollieren könnten.

[1] Eine interessante Konsequenz daraus ist, dass TLS (vgl. Abschnitt 4.6.3) nicht *beweisbar* sicher ist.

Sämtliche Schreibzugriffe dürfen nur lokal und von bestimmten Teilen des Systems vorgenommen werden, und wenn besonders empfindliche Daten gespeichert werden, erfordern auch Lesezugriffe Administratorrechte. Vorhandene Aufzeichnungen dürfen auf keinen Fall verändert werden. In besonders sicherheitsrelevanten Systemen werden alle Log-Nachrichten sofort und direkt auf einen Drucker mit Endlospapier ausgegeben. In einer reinen Softwarelösung empfiehlt es sich, die Log-Nachrichten zu authentisieren und ihre Integrität zu schützen.

### 4.4.6 Zertifikate und Public-Key-Infrastruktur

Ein *Public-Key-Zertifikat* ist ein öffentlicher Schlüssel eines Prinzipals, des *Subjekts* (englisch *subject*), der von einem anderen Prinzipal, dem *Aussteller* (englisch *issuer*), digital signiert wurde. Mit einem Zertifikat kann durch Prüfung der Signatur die Authentizität des öffentlichen Schlüssels geprüft werden. Oft enthält das Zertifikat zusätzlich Bezeichner des Subjekts und des Ausstellers. Der Aussteller bescheinigt dann dem Subjekt, der Besitzer des zugehörigen privaten Schlüssels zu sein. Weitere Bestandteile sind der *Gültigkeitszeitraum*, *die Seriennummern*, *erlaubte Verwendungszwecke* des Schlüsselpaars und weitere Daten über Subjekt und Aussteller. Der Aussteller wird auch *Certification Authority* (CA) genannt.

Da prinzipiell jeder Besitzer eines Schlüsselpaars beliebige Zertifikate ausstellen kann, ist ein Zertifikat nur dann brauchbar, wenn der Aussteller zuverlässig ist, man also darauf vertrauen kann, dass er alle im Zertifikat enthaltenen Angaben sorgfältig geprüft und sichergestellt hat, dass das Subjekt im Besitz des zugehörigen privaten Schlüssels ist. In kommerziellen Anwendungen übernehmen spezielle Sicherheitsunternehmen die Rolle des Ausstellers. Vertrauen in eine solche CA impliziert das Vertrauen in deren Unabhängigkeit und korrekte Dienstleistung, beides Aspekte, die ein Benutzer in der Regel nicht nachprüfen kann.

Der öffentliche Schlüssel eines Ausstellers kann wiederum von einer anderen Instanz zertifiziert werden. In diesem Fall ist für die Prüfung eines einzelnen Benutzer-Zertifikats die Prüfung einer ganzen *Zertifikatskette* notwendig, deren eines Ende das Benutzer-Zertifikat und das andere das Zertifikat einer zuverlässigen *Wurzel-CA* ist. Die Bestimmung einer gemeinsamen vertrauenswürdigen Wurzel-CA ist umso schwieriger, je unterschiedlicher die teilnehmenden Prinzipale sind. Für die Gültigkeit eines Zertifikats gibt es zwei Modelle: Im *Schalenmodell* werden automatich alle Zertifikate, die in der Kette hinter einem ungültigen Zertifikat liegen, mit ungültig. Im *Kettenmodell* dagegen ist es möglich, dass ein Zertifikat länger gültig ist als das Ausstellerzertifikat.

Als Wurzel-Zertifikate dienen entweder *selbst signierte Zertifikate*, in denen sich der Aussteller selbst zertifiziert, oder *überkreuzte Zertifikate* (englisch *cross certificates*), mit denen sich zwei CAs gegenseitig zertifizieren.

Manchmal ist es notwendig, ein Zertifikat vor dem Ablauf seines Gültigkeitszeitraums für ungültig zu erklären, beispielsweise wenn ein privater Schlüssel korrumpiert wurde oder ein Subjekt nicht mehr vertrauenswürdig ist. Eine Möglichkeit sind *Zertifikat-Widerrufslisten* (englisch *Certificate Revocation Lists*) (CRLs): Jede CA erzeugt in regelmäßigen Abständen eine digital signierte Liste aller ungültig gewordenen Zertifikate. Bei der Überprüfung eines Zertifikats ist dann immer zu prüfen, dass es nicht in einer Widerrufsliste aufgeführt ist. Mit CRLs sind im Wesentlichen zwei Probleme verbunden: Zum einen sind die Widerrufslisten, da sie asynchron erzeugt werden, nicht immer auf dem aktuellsten Stand, zum anderen wachsen die Listen schnell an. Obwohl bei der Überprüfung eines Zertifikats eigentlich nur ein einziger Eintrag der gesamten Liste relevant ist, muss stets die gesamte Liste geladen werden. Aufgrund dieser Erkenntnis wurde das *Online Certificate Status Protocol* (OCSP, [OCSP]) entwickelt. Benutzer können damit den Status eines

Zertifikats bei einem *OCSP responder* der CA abfragen, der stets auf dem aktuellen Stand ist. Der Nachteil von OCSP gegenüber den Widerrufslisten ist, dass zur Authentisierung der Antwort – also für jedes einzelne Zertifikat – eine rechnerisch aufwändige digitale Signatur notwendig ist, während die Widerrufsliste mit einer einzigen Signatur beliebig viele Zertifikate auf einmal widerruft.

Alle Protokolle und Dienste zur Verbreitung von Zertifikaten und Widerrufslisten fasst man unter dem Begriff *Public-Key-Infrastruktur* (PKI) zusammen. Da Zertifikate und Widerrufslisten selbst nur öffentliche Daten enthalten, kann man sie in einem öffentlich zugänglichen *Zertifikatsverzeichnis* ablegen, in dem alle Benutzer bei Bedarf nachschlagen können. Außerdem ist der Austausch von Zertifikaten Bestandteil vieler Schlüsselverhandlungsprotokolle (vgl. Abschnitt 4.6.2).

Der De-facto-Standard für Zertifikate und PKI ist X.509 (vgl. [X509]). Als Schwachstelle von X.509 galt der zunächst propagierte globale Namensraum für die Bezeichner der Prinzipale, der inzwischen nicht mehr gefordert wird. Flexibler als X.509, aber auch nicht so weit verbreitet, ist die *Simple Distributed Security Infrastructure* [RL96].

## 4.5 Firewalls

Eine zentrale Eigenschaft des Internets ist die *durchgehende Verbindungsfähigkeit*: Grundsätzlich kann jeder angeschlossene Rechner jedem anderen eine Netzwerknachricht schicken, ohne dass deren Inhalt von dazwischenliegenden Rechnern oder Routern interpretiert werden muss.

In vielen Fällen ist diese Eigenschaft nicht erwünscht, und es werden *Firewalls* eingesetzt, um die möglichen Verbindungen zu reglementieren. *Firewalls* sind Hardware- oder Software-Einheiten, die die Kommunikation zwischen einem privaten und einem öffentlichen Netzwerk kontrollieren. Sie dienen zum Schutz vor Eindringlingen und zur Durchsetzung der Sicherheitspolitik des privaten Netzwerks. Firewalls sitzen meist als *Gateway* an der Grenze zwischen dem öffentlichen und dem privaten Netzwerk, sodass sämtliche Nachrichten zwischen den beiden Netzwerken die Firewall passieren müssen. Ungültige Nachrichten werden abgeblockt.

Man unterscheidet zwischen *zustandslosen* und *zustandsbehafteten Firewalls*. *Zustandslose* Firewalls entscheiden bei jedem einzelnen Netzwerkpaket, ob es passieren darf, beispielsweise anhand der IP-Adresse oder der Portnummer des Senders bzw. Empfängers. *Zustandsbehaftete* Firewalls führen Buch über den Zustand von Verbindungen und versuchen festzustellen, ob Pakete dazu passen.

Eine *application-layer firewall* ist eine Firewall, die die Nachrichten bestimmter Protokolle analysiert und gegebenenfalls abblockt. So können z.B. alle ein- und ausgehenden E-Mails einer Institution schon an der Firewall auf Viren überprüft oder das Tunneln von Protokollen durch HTTP entdeckt und verhindert werden.

Fast alle zustandsbehafteten Firewalls benutzen *Network Address Translation* (NAT, [NAT]) um interne private IP-Adressen in eventuell temporäre, global eindeutige Adressen zu übersetzen. Die am häufigsten eingesetzte Variante *Network Address Port Translation* (NAPT) schließt auch Portnummern mit ein. Für jede Verbindung, die ein Rechner aus dem privaten Netzwerk nach außen aufbaut, öffnet die Firewall dann ein separates Socket, über das sie mit dem Gegenüber kommuniziert. Von außen gesehen besitzen dann alle Rechner des privaten Netzwerks die IP-Adresse der Firewall. Die Firewall modifiziert im IP-Header jedes Pakets die Adresse des privaten Rechners und leitet die Nachricht an den Zielrechner weiter.

Für Server-Dienste, die nach außen zur Verfügung gestellt werden, lassen sich in den meisten Firewalls *virtuelle Server* konfigurieren. Ein Dienst, der über die Portnummer identifiziert wird, wird dann immer auf einen bestimmten privaten Rechner umgeleitet.

Die großen Vorteile von NAT sind:

- Die privaten Rechner und die Topologie des privaten Netzwerks sind von außen unsichtbar.
- Die Kosten für die Eintragung globaler IP-Adressen werden reduziert.
- Es spielt keine Rolle, wie viele Rechner im privaten Netzwerk vernetzt sind.
- Das lokale Netzwerk besitzt einen eigenen lokal eindeutigen Namensraum. Lokale IP-Adressen können dynamisch vergeben werden.
- Der Aufwand einer eventuellen Umstrukturierung des privaten Netzwerks wird stark reduziert.
- Mit virtuellen Servern lässt sich Lastverteilung implementieren, und die Migration von Serverdiensten innerhalb des privaten Netzwerks wird stark vereinfacht.

Allerdings bereitet NAT in einigen Fällen große Schwierigkeiten. Anwendungen können nicht mehr von global eindeutigen IP-Adressen ausgehen. IP-Adressen, die außerhalb des Headers einer Nachricht übertragen werden, werden von einer NAT-Firewall nicht entdeckt und somit auch nicht modifiziert. Ein typisches Beispiel hierfür ist das FTP-Protokoll, in dem der Client dem Server mitteilt, auf welcher Socket-Adresse dieser die gewünschten Dateien senden soll. Obwohl der Client dann eine dazugehörige TCP/IP-Verbindung initiiert, stimmen die übersetzte IP-Adresse und Portnummer mit höchster Wahrscheinlichkeit nicht mit den übertragenen Daten überein. Als Abhilfe werden oft *Application-Layer Gateways* (ALGs) in der Firewall eingesetzt, die unter Kenntnis eines Protokolls solche Pakete aufspüren und passend modifizieren. Für jede Anwendung einen separaten ALG zu implementieren, ist allerdings wenig sinnvoll. Verschlüsselte Pakete können von einem ALG nicht inspiziert werden, und die Modifizierung einer mit einem MAC oder einer digitalen Signatur geschützten Nachricht zerstört deren Integrität. Die durchgehende Sicherheit von Verbindungen widerspricht dem Prinzip der inspizierenden Firewall.

Peer-to-Peer-Anwendungen sind ebenfalls problematisch, denn sie erfordern, dass jeder Teilnehmer „als Server“ erreichbar sein muss.

Da UDP-Übertragung zustandslos ist, erfordert sie keinen Verbindungsaufbau und löst damit auch keine Registrierung in der Adressübersetzungstabelle der Firewall aus. Damit UDP-Nachrichten die Firewall überhaupt durchdringen können, muss ein virtueller Server zur entsprechenden Portnummer registriert werden. Einige NAT-Firewalls bieten zusätzlich so genannte *trigger ports*: Erfolgt ein TCP/IP-Verbindungsaufbau über einen solchen Port, registriert die Firewall automatisch zusätzliche Verbindungen des gleichen privaten Rechners über eine Reihe anderer Ports.

Im IP-Multicasting erfolgt eine explizite Verbindungsregistrierung, sodass die eben geschilderten UDP-Probleme nicht auftreten. IP-Multicast-Adressen werden allerdings wegen des erwarteten hohen Kommunikationsvolumens von vielen Firewalls gesperrt.

Um eine Anwendung NAT-kompatibel zu entwerfen, gibt es eine Reihe wertvoller Hinweise in [Sen02]. Insbesondere sollten keine IP-Adressen und Portnummern außerhalb der TCP- bzw. UDP-Header übertragen werden. Große UDP-Pakete, die fragmentiert werden müssen, sollten auch vermieden werden, da die richtige Zustellung von Nachrichtenfragmenten an verschiedene virtuelle Server des gleichen Dienstes große Probleme bereitet.

Generell gilt, dass eine Anwendung nur durch eine Firewall kommunizieren kann, wenn diese die notwendigen Dienste freigibt. Die richtige Konfiguration der Firewall ist eine nicht zu unterschätzende Aufgabe. Verschlüsselte oder authentisierte Kommunikation ist nur nach Freischaltung eines Ports möglich.

## 4.6 Sicherheitsprotokolle

In Abschnitt 4.4 wurden bereits die grundlegenden kryptographischen Methoden beschrieben, mit deren Hilfe Prinzipale sicher miteinander kommunizieren können. In diesem Abschnitt werden die dabei eingesetzten Protokolle vorgestellt.

### 4.6.1 Sicherheitsassoziation

Die *Sicherheitsassoziation* (englisch *Security Association*) (SA) zwischen zwei Entitäten definiert, wie diese Entitäten Sicherheitsdienste zur sicheren Kommunikation miteinander verwenden. Attribute einer Sicherheitsassoziation sind z.B. (vgl. [ISAKMP]):

- Ein Chiffrieralgorithmus
- Ein Authentisierungsmechanismus
- Die benötigten kryptographischen Schlüssel
- Ein Hash-Algorithmus für die Integritätssicherung
- Ein Algorithmus zur Generierung von Pseudo-Zufallszahlen
- Algorithmenparameter wie Schlüssellänge oder Initialisierungsvektor.

Diese Attribute werden entweder im Voraus vereinbart oder beim Verbindungsaufbau verhandelt. Im Multicasting-Fall werden Sicherheitsassoziationen zwischen einzelnen Teilnehmern und der Gruppe geschlossen (siehe hierzu Abschnitt 4.6.4).

Die Verhandlung der verwendeten Algorithmen ist insbesondere bei einer Verbindung auf tieferen Netzwerkschichten, die direkt von den Betriebssystemen der Kommunikationsteilnehmer verwaltet werden, sinnvoll. In dieser Arbeit soll die gesamte Sicherheitsarchitektur allerdings auf der Anwendungsschicht implementiert werden. Die verwendeten Algorithmen können a priori festgelegt und der Implementierung des DVE hinzugefügt werden, wodurch deren Verhandlung entfällt. Der Initialisierungsvektor muss bei nicht-geordneter oder nicht-verlässlicher Übertragung für jedes Paket einzeln generiert werden. Er kann entweder im Klartext mitgesendet werden, oder aus synchronisierten Zustandsvariablen berechnet werden. Auch im Gruppenkontext ergibt die Verhandlung von individuellen Algorithmen zwischen einzelnen Teilnehmern und der Gruppe wenig Sinn. Für langlebige DVEs sollte allerdings die Möglichkeit der Änderung der kryptographischen Algorithmen offen gehalten werden, da neue Schwachstellen entdeckt werden können. Dieser hoffentlich extrem seltene Fall kann mit einem *Patch* des Systems bewerkstelligt werden.

Damit bleibt für DVEs als einzige Aufgabe die Verhandlung von kryptographischen Schlüsseln zur Chiffrierung und Authentisierung der kommunizierten Nachrichten, die in den folgenden Abschnitten beschrieben wird.

### 4.6.2 Schlüsseletablierung

Als *Schlüsseletablierung* (englisch *key establishment*) bezeichnet man einen Prozess oder ein Protokoll, an dessen Ende die teilnehmenden Parteien einen gemeinsamen geheimen Schlüssel besitzen (vgl. [MvOV97]). *Schlüsseltransport* bezeichnet die Etablierung eines Schlüssels, der von einem Kommunikationspartner generiert wird, während bei der *Schlüsselverhandlung* (englisch *key agreement*) mehrere Parteien in die Generierung des Schlüssels verwickelt sind.

Die *authentisierte Schlüsselverhandlung* zwischen zwei Parteien erfüllt in der Regel drei Funktionen:

- (Gegenseitige) Authentisierung.
- Verhandlung der Sicherheitsassoziation.
- Etablierung eines gemeinsamen, nur den kommunizierenden Parteien bekannten Schlüssels.

Aus dem gemeinsamen Schlüssel können beide Parteien dann spezielle Schlüssel für die Chiffrierung und Authentifizierung der sicheren Verbindung generieren. Die bei der Verhandlung erfolgte Authentisierung kann zusätzlich für die Zugangskontrolle verwendet werden. Oft sind für eine Schlüsselverhandlung mehrere Nachrichten notwendig. Beide Parteien generieren dann während der Verhandlung eine Reihe teilweise temporärer Zustandsvariablen.

Bei der Schlüsselverhandlung wird typischerweise ein kurzlebiger *Sitzungsschlüssel* auf Basis langlebiger *Authentisierungsschlüssel* etabliert. Dadurch wird [MvOV97]

- die Menge an Chiffretext begrenzt, die mit demselben Schlüssel erzeugt wird,
- der Effekt von kompromittierten Schlüsseln gemindert,
- die Langzeitspeicherung vieler unterschiedlicher geheimer Schlüssel vermieden und
- Unabhängigkeit zwischen Sitzungen gewährleistet.

Die grundlegenden Eigenschaften authentisierter Schlüsselverhandlungsprotokolle (vgl. [Kra03], [MvOV97]) sind:

**Authentisierung:** Die Teilnehmer des Protokolls verifizieren eindeutig die Identität des Kommunikationspartners.

**Konsistenz:** Die Teilnehmer müssen ein konsistentes Bild davon haben, wer die Partner in der Kommunikation sind. Konkret bedeutet das, wenn $A$ ein Geheimnis $K$ mit jemandem verhandelt, den er für $B$ hält, muss dieser ebenso glauben, einen Schlüssel $K$ mit $A$ verhandelt zu haben. Bei *man-in-the-middle-Angriffen* ist die Konsistenz oft in Gefahr.

**Geheimhaltung:** Keine dritte Partei darf über einen zwischen zwei aufrichtigen Teilnehmern etablierten geheimen Schlüssel Informationen erhalten können.

Es existieren sehr viele Schlüsselverhandlungsprotokolle, die jeweils für unterschiedliche Anwendungen angepasst sind und zusätzliche spezifische Eigenschaften aufweisen. Diese sind beispielsweise:

**Identitätenschutz:** Die Identität des Verhandlungspartners wird nur seinem Gegenüber offenbart und ist von außen nicht erkennbar.

**Perfekte Vorwärtsgeheimhaltung:** Die Aufdeckung eines Langzeitschlüssels gefährdet nicht die Sicherheit vorangegangener Sitzungen.

**Schlüsselbestätigung:** Die Verhandlungspartner erhalten jeweils einen Beweis, dass der andere den gemeinsamen Schlüssel erhalten hat.

**Lebendigkeitsgarantie:** Jeder Teilnehmer besitzt am Ende der Verhandlung einen Beweis, dass das Gegenüber zur Verhandlungszeit „lebendig“ war, beispielsweise besitzt $B$ eine digitale Signatur von $A$ unter einem von $B$ für die Sitzung neu generierten Wert.

### Angriffe auf die Schlüsseletablierung

Die Schlüsseletablierung ist besonders anfällig für *Denial-of-Service-Angriffe* (DoS), denn sie erfordert oft aufwändige kryptographische Operationen, die besonders einen zentralen Server vollständig auslasten können. Etablierungsprotokolle, die viele Nachrichten benötigen, sind besonders empfindlich, da die Zustandsinformationen von vielen simultan ausgeführten Verhandlungen auch Speichergrenzen sprengen können.

*Cookies* dienen der Abwehr von DoS-Angriffen durch Überlastung des Zielrechners durch halb vollendete oder vorgetäuschte Protokollnachrichten. *Cookies* sind Einträge in einer Nachricht zur schwachen Authentisierung des Kommunikationspartners. Sie werden mit einer Einwegfunktion aus einer Reihe von Verbindungsparametern, z.B. der Socket-Adresse des Verhandlungspartners oder dessen User-ID, und einem geheimen Schlüssel generiert. Beide Verhandlungsteilnehmer erzeugen in der ersten Runde des Etablierungsprotokolls ein solches Cookie und senden es in der ersten Nachricht an das Gegenüber. Alle folgenden Nachrichten müssen das Cookie des Partners enthalten, damit es dieser beim Empfang verifizieren kann. So lässt sich, ohne Zustandsinformationen lokal zu speichern, schnell feststellen, ob eine Nachricht wirklich ein Teil eines begonnenen Etablierungsprotokolls ist. Da der im Cookie verwendete Schlüssel nur dem Erzeuger bekannt ist, sind Cookies schwer fälschbar.

Erfolgt die Schlüsseletablierung über einen paketbasierten Kanal, sollten die Nachrichten nicht größer als die MTU der Verbindung sein, sodass sie nicht fragmentiert werden müssen.

### Schlüsseltransportprotokolle

*Schlüsseltransportprotokolle* kommen besonders dann zum Einsatz, wenn neue Schlüssel nur von einem der beiden Teilnehmer generiert werden, oder ein bereits vorhandener Schlüssel, z.B. ein Gruppenschlüssel, transportiert werden soll.

*Multimedia Internet KEYing* (MIKEY, [MIKEY]) definiert mehrere Protokolle zur Etablierung eines gemeinsamen geheimen Schlüssels für verteilte Echtzeit-Multimedia-Anwendungen in einem heterogenen Netzwerk. Für den vertraulichen Schlüsseltransport bietet MIKEY zwei verschiedene Protokolle an. In beiden Fällen generiert *Initiator A* einen neuen geheimen Schlüssel $K_{new}$ und transportiert diesen auf vertrauliche und authentisierte Weise zum *Responder B*.

Beide Protokolle verzichten zu Gunsten einer minimalen Nachrichtenzahl auf eine Lebendigkeitsgarantie des Gegenüber. Die Gefahr eines *replay-Angriffs*, bei dem sich ein Eindringling mithilfe einer aufgezeichneten Nachricht als *A* ausgibt, kompensieren sie durch die Verwendung und Authentisierung eines Zeitstempels $T$ und einer frischen Zufallszahl $N$. *B* muss also unbedingt prüfen, ob der Zeitstempel neu genug ist, und kann zusätzlich die Zufallszahlen $N$ aller Protokolldurchläufe in einem *replay buffer* speichern, um alte aufgezeichnete Nachrichten zu identifizieren. In beiden Protokollen ist die Bestätigungsnachricht optional. Geht die Nachricht an *B* verloren, muss er sich eventuell erneut am System anmelden.

In der ersten Variante wird ein *vorab vereinbarter geheimer Schlüssel* (englisch *pre-shared key*) $K_{pre}$ benutzt. Aus diesem wird ein Schlüssel $K_{enc}$ zur Verschlüsselung und ein weiterer Schlüssel $K_{mac}$ zur Authentisierung generiert. Der Zeitstempel $T$ und die Zufallszahl $N$ dienen zum Nachweis, dass die Nachricht neu ist. In der Notation aus A.1 ist das Protokoll in Abbildung 4.1 abgebildet. Das MIKEY-Protokoll mit vorab vereinbartem Schlüssel eignet sich beispielsweise zur Verlängerung einer Sicherheitsassoziation, wenn die Menge der übertragenen Nachrichten einen Schlüsselwechsel erforderlich macht. Der bisherige Sitzungsschlüssel kann dann als $K_{pre}$ benutzt werden, um den neuen Sitzungsschlüssel $K_{new}$ zu transportieren.

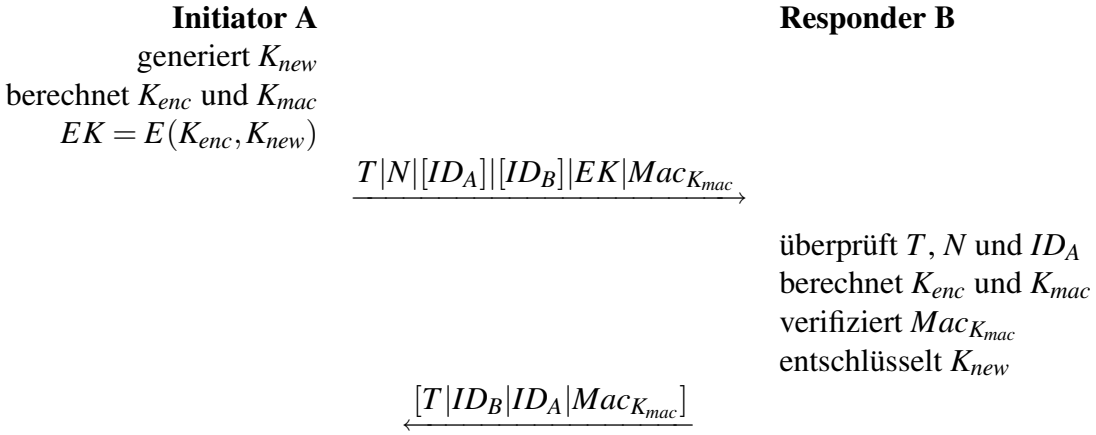



Abbildung 4.1: MIKEY-Schlüsseltransport mit einem vorab vereinbarten Schlüssel

Die zweite Variante benutzt asymmetrische Verschlüsselung und eine digitale Signatur. Es wird vorausgesetzt, dass beide Kommunikationspartner den öffentlichen Schlüssel $K^+$ ihres Gegenüber besitzen. Neben dem neuen Schlüssel $K_{new}$ generiert $A$ einen zufälligen *envelope-Schlüssel* $K_{env}$, aus dem $K_{enc}$ und $K_{mac}$ erzeugt werden. $K_{env}$ wird dann mit dem öffentlichen Schlüssel von $B$ verschlüsselt, und durch eine digitale Signatur authentisiert. Der neue Schlüssel $K_{new}$ wird mit $K_{enc}$ verschlüsselt und mit $K_{mac}$ authentisiert. Das Protokoll ist in Abbildung 4.2 skizziert. Mit diesem Protokoll können beispielsweise neue individuelle Schlüssel an ein einzelnes Mitglied einer Multicast-Gruppe versendet werden, ohne dass eine zusätzliche Unicast-Verbindung aufgebaut werden muss, oder für die Etablierung einer neuen Sicherheitsassoziation, deren Sitzungsschlüssel von der vorhandenen Sicherheitsassoziation vollkommen unabhängig sind.

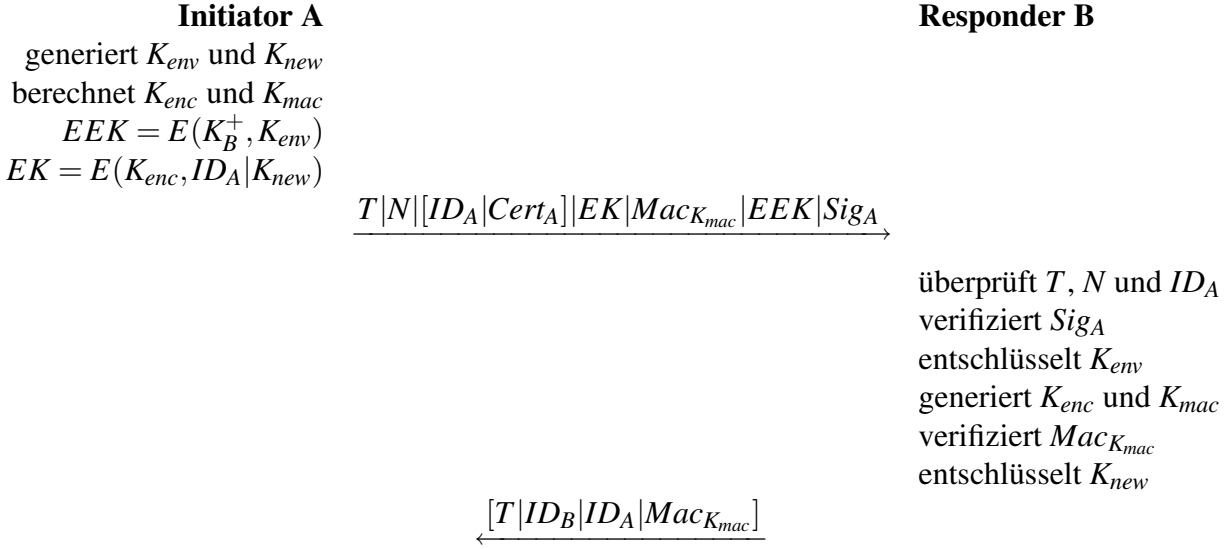



Abbildung 4.2: MIKEY-Schlüsseltransport mit *public-key*-Verschlüsselung

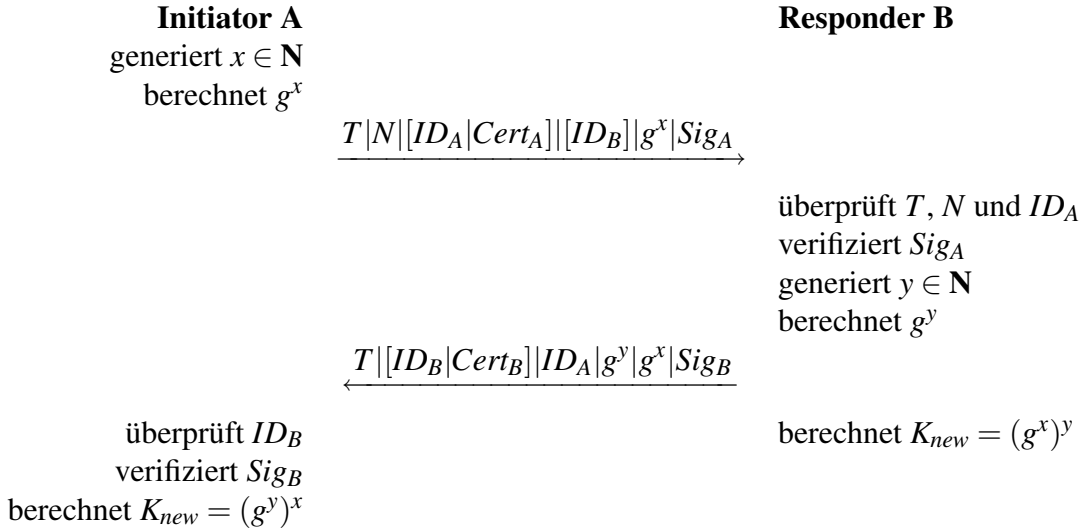



Abbildung 4.3: MIKEY-Schlüsselverhandlung mit digitaler Signatur

### Schlüsselverhandlungsprotokolle

Die meisten Schlüsselverhandlungsprotokolle beruhen auf der in Abschnitt A.5 beschriebenen *Diffie-Hellman-Verhandlung*. Eine Schlüsselverhandlung benötigt mindestens zwei Nachrichten.

Auch MIKEY definiert zwei Schlüsselverhandlungsprotokolle auf der Basis von Diffie-Hellman. Das erste benutzt digitale Signaturen zur Authentisierung und ist in Abbildung 4.3 dargestellt.

Aufgrund seiner zwei Nachrichten benötigt das Protokoll keine Zustandsspeicherung der Schlüsselverhandlungen, und bringt daher eine Grundresistenz gegen DoS-Angriffe mit, da keine Zustandspuffer überlaufen können. Es eignet sich daher besonders gut als Anmeldungsprotokoll für Anwendungen mit vielen Teilnehmern.

Zwar erhält *A* einen Lebendigkeitsbeweis von *B*, da dieser den frisch generierten Teilschlüssel $g^x$ mit signiert, aber *B* erhält keinen solchen Beweis von *A*. Stattdessen werden, wie im letzten Abschnitt beschrieben, Zeitstempel und Zufallszahlen eingesetzt.

Das zweite MIKEY-Schlüsselverhandlungsprotokoll [MIKEY2] ersetzt die Signaturen des vorangegangenen Protokolls durch MACs mit einem vorab vereinbarten geheimen Schlüssel. Da wir ohnehin den Einsatz von Zertifikaten planen, um auf Langzeitschlüssel symmetrischer Algorithmen verzichten zu können, wird es hier nicht weiter behandelt.

Auch die *SIGMA-Protokolle* [Kra03] bauen auf der Diffie-Hellman-Verhandlung auf. Abbildung 4.4 zeigt das *SIGMA-I-Protokoll*, wobei der Einfachheit halber einige Elemente wie Cookies und Zufallszahlen nicht abgebildet sind. Anders als die MIKEY-Protokolle schließt es allerdings gegenseitige Lebendigkeitsbeweise und Identitätsschutz bei Lauschangriffen ein. Der Preis dafür ist eine zusätzliche Protokollnachricht. Die Schlüssel $K_{enc}$ und $K_{mac}$ werden vom Diffie-Hellman-Schlüssel $K_{new}$ abgeleitet.

Das *SIGMA-I*-Protokoll bietet von allen in der Recherche dieser Arbeit untersuchten Schlüsselverhandlungsprotokolle mit drei Nachrichten die maximale Menge von Sicherheitseigenschaften. Es wird daher bei der Verhandlung von Sicherheitsassoziationen zwischen besonders sicherheitsrelevanten Entitäten eingesetzt werden.

Das zweite *SIGMA*-Protokoll, *SIGMA-R*, schützt zusätzlich die Identität von *B* gegen aktive Angriffe, benötigt aber vier Nachrichten. Da DVEs in der Regel keinen Identitätsschutz erfordern, ist es hier nicht von Interesse.

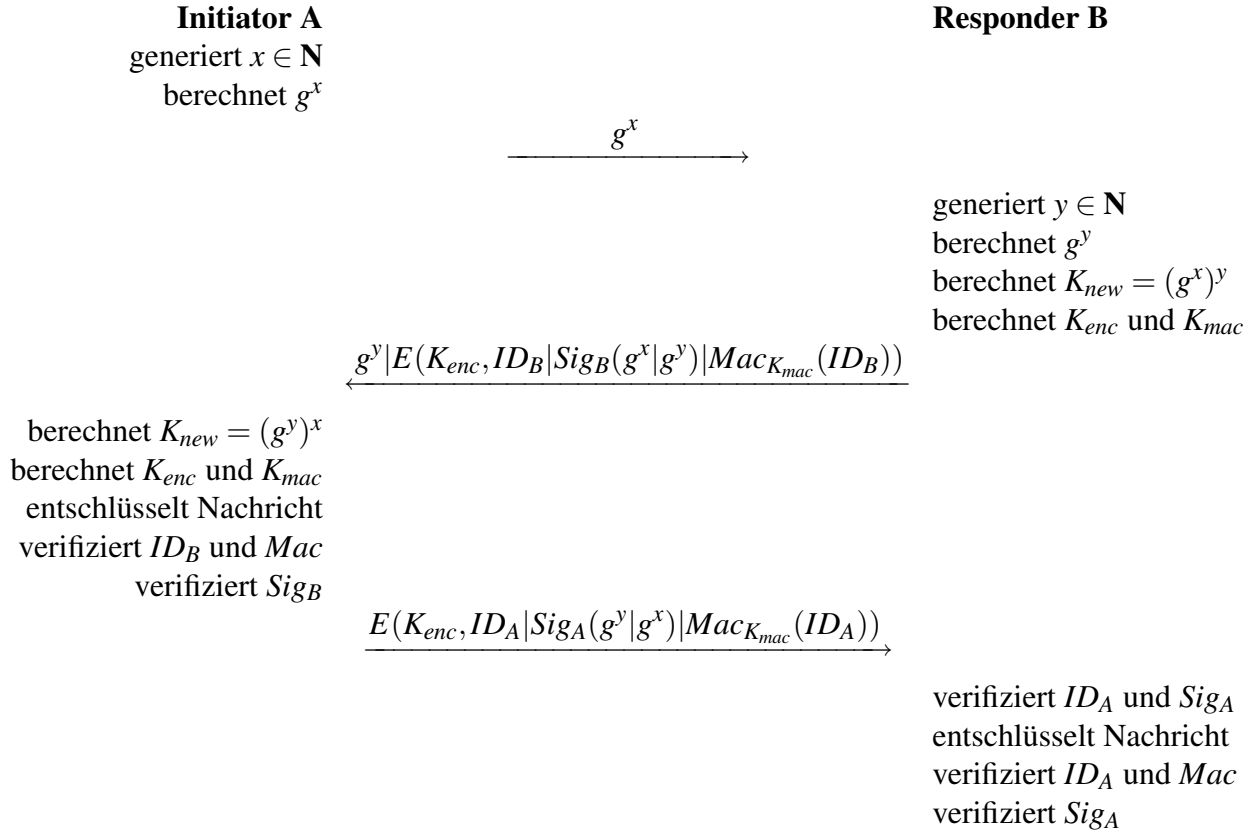



Abbildung 4.4: SIGMA-I-Schlüsselverhandlung

### 4.6.3 Transport Layer Security

*Transport Layer Security* (TLS, [TLS]) bezeichnet eine geringfügig veränderte, standardisierte Fassung von Netscape's *Secure Socket Layer* (SSL), Version 3. TLS ist einer der am weitesten verbreiteten Standards für sichere Verbindungen im Internet. TLS setzt direkt auf einem verlässlichen Transportprotokoll, z.B. TCP/IP, auf und ist unabhängig vom darüber liegenden Anwendungsprotokoll, wodurch vorhandene Anwendungen auf einfache Weise „sicher gemacht" werden können.

TLS implementiert zwei übereinander liegende Protokolle: Das *TLS record protocol* und das *TLS handshake protocol*. Das erste stellt eine vertrauliche und verlässliche Verbindung zur Verfügung, sorgt für die Fragmentierung und bildet die Grundlage für andere Anwendungsprotokolle. Das *TLS handshake protokoll* dient der Schlüsseletablierung und bietet verschiedene Kombinationen von Authentisierung und Verhandlungen der Sicherheitsassoziation.

Über die Sicherheitsprobleme von TLS ist viel geschrieben worden (beispielsweise [BN00], [Vau02]). Als Kompromiss zwischen Sicherheit und der Verwendung von Standards setze ich in dieser Arbeit TLS daher nur an Stellen ein, an denen Standardtechnologien benutzt werden, die explizit TLS zur Sicherung der Nachrichtenübertragung vorsehen, also für HTTPS und LDAPS.

### 4.6.4 Sicheres Multicasting

Um sicher innerhalb einer Gruppe zu kommunizieren, wird eine *Gruppensicherheitsassoziation* verwendet. Die Nachrichten werden mit einer symmetrischen Chiffre verschlüsselt und mit einem MAC als Gruppennachricht authentisiert. Dazu muss jedes Gruppenmitglied die geheimen *Grup-*

*penschlüssel* kennen. Die Registrierung eines neuen Teilnehmers und damit auch der Schlüsseltransport kann dann mit einem der Transportprotokolle aus Abschnitt 4.6.2 durchgeführt werden. Für die Gruppenkommunikation ist Multicasting besonders effizient.

### Multicast-Gruppenschlüsselverwaltung

Manchmal ist es notwendig, einen Gruppenschlüssel zu ändern:

- Der Schlüssel ist Unberechtigten in die Hände gefallen.
- Die maximale Datenmenge, die sicher mit dem Schlüssel verschlüsselt werden kann, ist erreicht.
- Ein Gruppenmitglied soll ausgeschlossen werden.
- Ein Mitglied verlässt die Gruppe, und die Vertraulichkeit der Gruppe kann nur mit einem neuen Schlüssel gewährleistet werden.
- Ein neuer Abrechnungszeitraum für die Gruppengebühr beginnt.

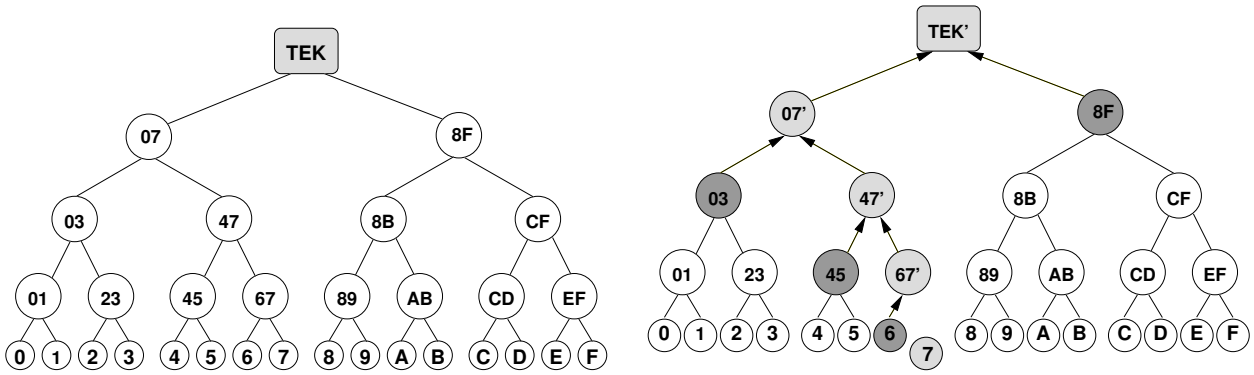
*Gruppenschlüsselverwaltung* beinhaltet daher meist die Möglichkeit eines *Schlüsselwechsels* (englisch *re-key*). Für perfekte Vorwärtsgeheimhaltung (vgl. Abschnitt 4.6.2) sollten die neuen Gruppenschlüssel nicht von alten, eventuell korrumpierten Schlüsseln abgeleitet oder, über die alte Sicherheitsassoziation geschützt, transportiert werden. Ausgeschlossene Mitglieder sollten den neuen Schlüssel auf keinen Fall erhalten. Speziell bei Multicast-Kommunikation in großen Gruppen ist das nicht einfach in effizienter Weise zu erreichen.

Um die Komplexität der Gruppenverwaltung zu reduzieren, wird in vielen Gruppenverwaltungsprotokollen ein einzelner *Gruppenverwalter* eingesetzt, der sowohl als Anmeldeinstanz für neue Gruppenmitglieder als auch als Schlüsselverwalter fungiert. Ein einzelner Gruppenverwalter ist natürlich ein kritischer Ausfallpunkt. Deshalb sind die meisten Gruppenverwaltungsprotokolle mit Verwalter so konzipiert, dass der Verwalter nur bei Veränderungen der Gruppenmitgliedschaft in Aktion tritt, und die Gruppe ansonsten unabhängig arbeiten kann. Oft werden auch mehrere hierarchisch angeordnete Gruppenverwalter eingesetzt, um den Verarbeitungs- und Kommunikationsengpass des zentralen Verwalters zu vermeiden.

In [Köh02] habe ich einige Multicast-Gruppenverwaltungsprotokolle bereits auf ihre Einsetzbarkeit in verteilten virtuellen Umgebungen geprüft. Die Ergebnisse werden im Folgenden kurz zusammengefasst.

Alle hier beschriebenen Verwaltungsmechanismen führen einen Schlüsselwechsel ausschließlich mit symmetrischen Algorithmen durch und beruhen auf einer Hierarchie von Schlüsseln.

*Logical Key Hierarchy* [LKH] benutzt einen binären Baum von Schlüsseln. Die Wurzel ist der *Traffic Encryption Key* ($TEK$), und darunter liegen die *Key Encryption Keys* ($KEK$s), die beim Schlüsseltransport eingesetzt werden. Die Blätter stehen für die individuellen Schlüssel der Gruppenmitglieder. Jedes Gruppenmitglied kennt außerdem alle Schlüssel auf dem Weg von seinem Blatt zum $TEK$ (vgl. Abbildung 4.5(a)). Verlässt ein Mitglied die Gruppe, werden alle ihm bekannten Schlüssel vom Gruppenverwalter neu generiert. Der Schlüsseltransport erfolgt dann per Multicasting, wobei jeder neue Schlüssel mit dem jeweils nicht korrumpierten Kind-Schlüssel verschlüsselt wird. Auf diese Weise können nur die berechtigten Empfänger die von ihnen benötigten Schlüssel dechiffrieren. Abbildung 4.5(b) zeigt ein Beispiel: Der Ausschluss von Gruppenmitglied




(a) Der Schlüsselbaum (b) Mitglied 7 verlässt die Gruppe

Abbildung 4.5: LKH-Schlüsselverwaltung

Nummer 7 verursacht, dass die Schlüssel $K_{67}$, $K_{47}$, $K_{07}$ und der $TEK$ neu generiert werden müssen. Die Schlüsseländerungsnachricht besteht damit aus

$$\begin{array}{|ll|}\hline E(K_6, K'_{67}) & \\ E(K_{45}, K'_{47}) & E(K'_{67}, K'_{47}) \\ E(K_{03}, K'_{07}) & E(K'_{47}, K'_{07}) \\ E(K_{8F}, K'_{TEK}) & E(K'_{07}, K'_{TEK}) \\ \hline\end{array}$$

Mit LKH kann man also nach Abgang eines Mitglieds eine Gruppe von $n$ Teilnehmern mit einer einzigen Multicast-Nachricht der Länge $\mathcal{O}(2\log n - 1)$ mit neuem Schlüsselmaterial versorgen. Der Verwalter speichert insgesamt $2n-1$ Schlüssel. Mehrere Abgänge können in einer Nachricht kombiniert werden, verlängern die Nachricht aber um so mehr, je weniger Schlüssel die Abgänger gemeinsam kannten.

Im *VersaKey Framework* [WCS+99] werden zwei weitere Betriebsarten von LKH eingeführt. Im ersten wird die Anzahl der Schlüssel reduziert, indem auf derselben Stufe im Baum ein Schlüssel für alle linken Äste gilt und ein anderer für alle rechten Äste. Der Baum wird dadurch zur Tabelle, und der Verwalter muss nur noch $2\log n + 1$ Schlüssel speichern. Jeder Schlüsselsatz, den ein Gruppenmitglied besitzt, unterscheidet sich mindestens um einen Schlüssel von jedem anderen Gruppenmitglied.

Die zweite Betriebsart kommt ohne Verwalter aus und ist damit die einzige hier betrachtete reine Peer-to-Peer-Gruppenschlüsselverwaltung. Sie beruht auf *global eindeutigen Mitgliedsbezeichnern* und *Schlüsselhaltern*. Die Bits des Bezeichners definieren, welche Schlüssel aus der Tabelle ein Mitglied benutzt. Für jeden Schlüssel gibt es einen einzigen Schlüsselhalter, der sich durch *heartbeat-Nachrichten* über das Multicast-Socket zu erkennen gibt. Ein neues Mitglied sammelt diese Nachrichten für alle Schlüssel, die ihm zustehen, und beschafft sich den jeweiligen Schlüssel beim Schlüsselhalter über einen sicheren Unicast-Kanal. Meldet sich kein Schlüsselhalter für einen benötigten Schlüssel, generiert der Neuling einen neuen Schlüssel und wird selbst zum Schlüsselhalter. Bei Ausschluss eines Teilnehmers wird zunächst ein *Ausschließer* gewählt. Dieser generiert den $TEK$ und die $KEK$*s*, die er besitzt und die auch der Abgänger kannte, neu. Danach sendet er eine Nachricht per Multicast. Diese enthält den neuen $TEK$ verschlüsselt mit allen noch gültigen $KEK$s, die nach dem oben beschriebenen Schema verschlüsselten neuen $KEK$s und den unverschlüsselten Bezeichner des Abgängers. Beim Erhalt dieser Nachricht prüft jeder Schlüsselhalter, ob sein $KEK$ korrumpiert ist, generiert diesen gegebenenfalls neu und sendet ihn ebenfalls, wie oben beschrie-

ben, verschlüsselt über das Multicast-Socket. Nach einigen Runden erhalten auf diese Weise alle Mitglieder außer dem Abgänger den neuen $TEK$ und die ihnen zustehenden $KEK$s. Der größte Nachteil dieser Betriebsart ist, dass sie auf dem verlässlichen Transport aller Nachrichten basiert. Verloren gegangene *heartbeat*- oder Schlüsselwechsel-Nachrichten können die Integrität des Systems schnell zerstören.

Eine andere Verbesserung von LKH sind *one-way function trees* ([MS03]). Die $KEK$s werden in OFT nicht mehr unabhängig von einander generiert, sondern stattdessen mithilfe von Einweg- und Mischfunktionen aus den Schlüsseln der Kindknoten berechnet. Ist $f$ eine Einwegfunktion und $g$ eine Mischfunktion, dann berechnet man beispielsweise den Schlüssel $K_{CD}$ in Abbildung 4.5(a) als

$$K_{CD} = g(f(K_C), f(K_D)).$$

Auf diese Weise können alle $KEK$s und der $TEK$ rekursiv aus den Blattschlüsseln berechnet werden. Das Bild eines Schlüssels $K$ unter der Funktion $f$ nennt man auch den *geblendeten Schlüssel* (englisch *blinded key*). Jedes Gruppenmitglied kennt die gleichen Schlüssel wie bei LKH plus die geblendeten Schlüssel der jeweiligen Geschwisterknoten, und kann daraus die $TEK$ berechnen. Der Schlüsselwechsel aus Abbildung 4.5(b) verläuft bei OFT wie folgt: Der Geschwisterknoten 6 des Abgängers ersetzt den Knoten 67. Der Gruppenverwalter generiert den neuen Schlüssel $K'_{67}$ und berechnet zunächst alle davon betroffenen Schlüssel neu. Dann sendet er

$$\boxed{\begin{array}{l} E(K_6, K'_{67}) \\ E(K_{45}, f(K'_{67}) \\ E(K_{03}, f(K'_{47}) \\ E(K_{BF}, K'_{07}) \end{array}}$$

per Multicasting, und alle Mitglieder können die von ihnen benötigten Schlüssel berechnen. Die Nachrichtenlänge ist nur noch $\mathcal{O}(\log n)$.

In [SKJH00] kommen Setia et al. zu dem Schluss, dass für große Gruppen mit starker Mitgliederfluktuation periodische Schlüsselwechsel wesentlich effizienter sind als Schlüsselwechsel bei jeder einzelnen Änderung der Gruppenzusammensetzung, und setzen diese Erkenntnis in *Kronos-System* um. Der Gruppenverwalter, in Kronos *domain-wide key distributor* genannt, ist für die $TEK$ verantwortlich und wird durch mehrere *area-wide key distributors* entlastet, die ihrerseits beispielsweise LKH benutzen können, um die $KEK$s ihrer Untergruppe zu verwalten. Der Verwalter kommuniziert mit den Unterverwaltern über einen separaten geschützten Multicast-Kanal.

*MARKS* [Bri99] koppelt den Schlüsselwechsel ebenso von der Gruppenverwaltung ab. Die Gruppenschlüssel, die jeweils ein festes Zeitinterval lang benutzt werden, werden im Voraus mit Einwegfunktionen aus wenigen geheimen Grundschlüsseln berechnet. Der Gruppenverwalter kann neuen Gruppenmitgliedern den Zugang zur Gruppe für einen bestimmten Zeitraum ermöglichen, indem er ihnen eine Kombination von Zwischenergebnissen der Schlüsselberechnung preisgibt, aus denen sie die benötigten Schlüssel für den gewünschten Zeitraum berechnen. Die Gruppenverwaltung kann daher beliebig repliziert werden, und beim Schlüsselwechsel ist keine Kommunikation notwendig.

Wird Multicasting auf der Anwendungsschicht implementiert (vgl. Abschnitt 3.3.5), kann die Gruppenkommunikation auch durch einzelne Sicherheitsassoziationen zwischen den benachbarten Knoten des Overlay-Netzwerks geschützt werden (englisch *hop-by-hop security*). Dabei muss jede Nachricht allerdings immer neu verschlüsselt oder authentisiert werden, was für zeitkritische Übertragungen nicht infrage kommt.

Mehr zum Thema Gruppenverwaltung findet man in [HT00]. Die *IETF Multicast Security Working Group* (MSEC, [MSEC]) befasst sich auch mit Gruppenverwaltungsprotokollen auf der Transportschicht (GDOI [GDOI] und GSAKMP [GSAKMP]) und für kleine Gruppen in Multimedia-Anwendungen (MIKEY, [MIKEY]).

### Multicast-Sender-Authentisierung

Mit der Gruppensicherheitsassoziation aus dem vorangegangenen Abschnitt kann man Nachrichten lediglich auf Gruppenbasis authentisieren. Gruppenmitglieder können sich so als beliebige andere Mitglieder ausgeben. Für individuelle Authentisierung von Multicast-Nachrichten benötigt man andere Methoden.

In [Lau03] hat Lauf die wichtigsten Verfahren zur individuellen Sender-Authentisierung von Multicast-Datenströmen verglichen. Analysiert wurden digitale Signaturen mit RSA, DSA und elliptischen Kurven, sowie EMSS [PCTS00], TESLA (vgl. Abschnitt A.6.3) und BiBa [Per01]. Im Vergleich liefen TESLA und RSA allen anderen Verfahren deutlich den Rang ab, konnten aber auch nicht hundertprozentig überzeugen: Während RSA rechnerisch zu aufwändig (vgl. Abschnitt B.6) ist, verursacht TESLA (vgl. Abschnitt A.6.3) zu lange Latenzen.

### Fazit

Alle in Abschnitt 4.6.4 beschriebenen Mechanismen zur Gruppenverwaltung gehen davon aus, dass alle Gruppenmitglieder zuverlässig sind. Tauschen die Mitglieder untereinander unerlaubt Schlüssel aus, können sie eventuell gar nicht mehr aus der Gruppe ausgeschlossen werden. Eine solche Form der *Kollusion* ist bei bestimmten DVEs durchaus möglich (vgl. Abschnitt 5.3.2) und kann nur durch Verzicht auf Multicast-Kommunikation konsequent ausgeschlossen werden.

Reicht die Authentisierung auf Gruppenbasis nicht aus, muss man entweder eine viel größere Auslastung der teilnehmenden Rechner durch kryptographische Operationen oder eine deutliche Vergrößerung der Latenz und somit Verluste bei der Interaktivität in Kauf nehmen.

Sicheres Multicasting kann für die Gruppenkommunikation in verteilten virtuellen Umgebungen also nur dann effizient eingesetzt werden, wenn die Gruppenmitglieder zuverlässig sind und einander vertrauen. Für die Übermittlung von Update-Nachrichten in DVEs mit stark kompetitivem Charakter kommt Multicasting daher nicht infrage.

# Kapitel 5

# Sicherheitsarchitektur für DVEs

Die Sicherheitsanforderungen einer verteilten virtuellen Umgebung hängen stark von der Art der Anwendung ab. Während in einer militärischen Simulation der Schwerpunkt beispielsweise auf Vertraulichkeit liegt, erfordert ein Computerspiel eine Zugangskontrolle, die die Abrechnung der Spielzeit ermöglicht, oder einen Schutz vor Betrugsversuchen.

Auch wenn viele IT-Produktanbieter behaupten, ihr Produkt in der neuesten Version um Sicherheit erweitert zu haben, ist eine bloße Erweiterung in den meisten Fällen schwierig; denn Sicherheit durchdringt ein System, d.h. sie greift an vielen Stellen in den unterschiedlichsten Schichten in das System ein. Die Authentifizierung eines Benutzers beispielsweise wirkt sich auf das Kommunikationsprotokoll aus, erfordert die Verwaltung von Authentisierungsdaten und dient letztlich dem Zugang zu Systemkomponenten auf höheren Ebenen, auf denen der Benutzer ebenfalls repräsentiert sein muss. Die Kapselung aller Sicherheitsmechanismen in einer einzigen zusätzlichen Software-Schicht oder -komponente ist meistens nicht möglich, und folglich muss das System an vielen Stellen verändert werden. Die Erweiterung um Sicherheitsmechanismen wird zusätzlich dadurch erschwert, dass ursprünglich „unsichere“ Software meist auch ohne ein explizites Vertrauensmodell entworfen wurde. Aus einer Sicherheitsanalyse ergeben sich dann oft völlig neue Rollen oder eine andere Aufgabenverteilung, die einen von Grund auf neuen Entwurf erfordern. Wenn ein Schutz vor Betrugsversuchen seitens der Spieler ergänzt werden soll, kann sich z.B. die Peer-to-Peer-Architektur eines Computerspiels als ungeeignet herausstellen.

In Abschnitt 5.1 werden zunächst die Sicherheitsaspekte von DVEs aus unterschiedlichen Perspektiven dargestellt. Eine Klassifizierung von DVEs nach deren Sicherheitsanforderungen liefert Abschnitt 5.2. Außerdem werden zwei Beispielszenarien vorgestellt. In Abschnitt 5.3 werden die Angriffe auf DVEs behandelt, die dann in Abschnitt 5.4 als Basis für die Sicherheitsanalyse der Beispielszenarien dienen. Der Entwurf und die Implementierung der Sicherheitsarchitektur sind Thema von Abschnitt 5.5, und deren Bewertung folgt in Abschnitt 5.6. Abschnitt 5.7 fasst die Ergebnisse dieser Arbeit zusammen und wagt einen kurzen Ausblick.

## 5.1 Sicherheitsaspekte von DVEs

In diesem Abschnitt wird erläutert, warum Sicherheit für verteilte virtuelle Umgebungen überhaupt relevant ist, und um welche Art von Sicherheit für welche Rollen es eigentlich geht. Die Sicherheitsanforderungen werden aus Sicht der Hersteller (5.1.1), der Benutzer (5.1.2) und der Betreiber (5.1.3) eines DVEs dargestellt. Der Abschnitt 5.1.4 beschäftigt sich mit *cheating*, welches besonders für Online-Computerspiele eine Rolle spielt.

### 5.1.1 Sicherheit für Hersteller

Hersteller von DVEs wollen mit ihrem Produkt in erster Linie Geld verdienen. Sicherheit bedeutet für den Hersteller daher *Sicherung des Gewinns*. Der Hersteller sollte also darauf achten, dass er an jedem Exemplar seines Produktes verdient. Das alleinige Vervielfältigungsrecht der Daten kann durch einen *Kopierschutz* gesichert werden. Alternativ können *Lizenzcodes* verkauft werden, ohne die sich das System nicht oder nur mit eingeschränktem Funktionsumfang starten lässt. Da Lizenzcodes einfach an Unbefugte weiter verteilt werden könnten, überwachen entweder *Lizenz-Server*, dass nur ein Exemplar des Programms zu jeder Lizenznummer läuft, oder die Lizenzen werden an bestimmte Hardware-Merkmale eines Rechners gebunden (englisch *node locked licence*), beispielsweise an einen *Kopierschutzstecker* (englisch *dongle*) oder an den *Media Access Code* der Netzwerkkarte.

Neben dem Gewinn durch den reinen Verkauf der DVE-Software, gibt es die Möglichkeit, die Hardware-Infrastruktur für den Betrieb des DVEs anzubieten. Auf dem Computerspiel-Sektor gibt es mittlerweile einige zunächst günstige oder gar kostenfreie Spiele, bei denen die tatsächliche Teilnahme am Spiel abgerechnet wird. Ein Server-Betreiber kann Spielern so seine Hardware als Spiel-Server anbieten. Sind Server-Betreiber und Hersteller verschieden, kann der Hersteller vertraglich an den Teilnahmegebühren der Spieler beteiligt werden. Den gängigen *präventiven* Sicherheitsmechanismus stellt hier ein *Geschäftsvertrag* dar, den *reaktiven* ein *Gerichtsprozess*. Softwarelösungen sind wohl nicht gefragt.

Andere Modelle zur Vermarktung von Online-Computerspielen, z.B. die Finanzierung über Werbung, werden in [Cos99] beschrieben, bieten aber keine wesentlich neuen Sicherheitsaspekte, weshalb hier nicht weiter darauf eingegangen werden soll.

Ein weiterer wichtiger Sicherheitsaspekt für den Hersteller ist der Schutz seines *intellektuellen Eigentums*. Dazu gehört beispielsweise die Produktidee oder der Produktname, die oft durch Patente oder das Urheberrecht geschützt werden. Dabei ist sich die Fachwelt noch nicht einig, ob sich diese traditionellen Schutzmethoden überhaupt auf Software anwenden lassen. Aber auch für die Details der Implementierung besteht Gefahr: Zwar wird kaum ein Spiel mit dem dazugehörigen Quellcode ausgeliefert, doch auf den heutigen universell einsetzbaren Rechnern können Software und Daten nicht wirksam gegen *reverse engineering* oder Kopieren geschützt werden. In [Pri00] beschreibt Pritchard, selbst ein Spielentwickler, den enormen Aufwand, den einige Hacker zum Knacken eines Spiels in Kauf nehmen, und kommt zum deprimierenden Fazit „*Your game, along with everything on the cheater's computer, is not secure. The files are not secure. Memory is not secure. Services and drivers are not secure.*“ Ein neuerer Trend in der Unterhaltungsindustrie ist das *Digital Rights Management* (DRM), mit dem durch eine integrierte Kombination spezieller „sicherer Hardware“ und „sicherer Software“ auf der Konsumentenseite die Urheberrechte der Hersteller durchgesetzt werden sollen. Ob sich jetzt schon umstrittene DRM-Systeme durchsetzen werden, wird die Zukunft zeigen.

Da also der Gewinn des Herstellers nicht effektiv mit Softwarelösungen gesichert werden kann, wird die Sicherheit für die Hersteller in dieser Arbeit nicht weiter behandelt.

### 5.1.2 Sicherheit für Benutzer

Der Benutzer einer verteilten virtuellen Umgebung hat ein Recht auf *Datenschutz*, d.h. dass eventuell über ihn gespeicherte persönliche Daten, wie Name, Anschrift oder Kreditkarteninformationen, vertraulich bleiben.

Wenn er für die Teilnahme am Spiel bezahlt, muss es für ihn verfügbar – also spielbar – sein.

Des Weiteren sollte ihm allein der Zugriff auf seine virtuellen Entitäten erlaubt sein. Ein mühsam in mehreren Dutzend Stunden Spielzeit entwickelter Computerspiel-Charakter stellt für den Spieler einen gewissen Wert dar, und sollte vor Beschädigungen, unerlaubtem Kopieren oder gar mutwilliger Zerstörung durch Löschen geschützt werden. Gleiches gilt für im Spiel erworbene Gegenstände.

Aus dem Bedürfnis der Spieler, die zeitaufwändige Entwicklung eines mächtigen Avatars in einem Online-Rollenspiel zu verkürzen, hat sich der reale Handel mit virtuellen Gegenständen oder mit ganzen Charakteren entwickelt. Einige Enthusiasten zahlen bis zu 11000 US$ [YC02] für einen besonders seltenen Gegenstand. Der Schutz virtuellen Eigentums bekommt dadurch eine völlig neue Dimension: Der Server, der Entitäten und Avatare speichert, wird zum lukrativen Angriffsziel, und es entsteht ein zusätzlicher Anreiz für Hacker, in die Interna eines Spiels einzudringen, um selbst wertvolle Gegenstände zu generieren oder Avatare zu modifizieren. Solche Angriffe können auch von Insidern geführt werden: In [YC02] wird von der Entlassung eines Spielprogrammierers berichtet, der sein Wissen missbraucht hat, um für sich einen unschlagbaren „Super-Charakter“ zu erzeugen, mit dem er die Avatare anderer Spieler angegriffen und ausgeraubt hat.

Ein oft vernachlässigter Sicherheitsaspekt für Software-Benutzer ist der Schutz des eigenen Rechners und der darauf befindlichen Daten. Ein Online-Computerspiel verbindet schließlich den eigenen Rechner mit anderen Rechnern eines meist öffentlichen Netzwerks, und ungeschützte Daten können ebenso versendet werden wie die Ereignisse des DVEs selbst. Fehlerhafte oder absichtlich destruktive Programme können zusätzlich zu Datenverlust oder Beschädigung der Hardware führen. Der Lizenzvertrag, den ein Software-Benutzer bei der Installation per Mausklick akzeptiert, schließt die Haftung des Herstellers für solche Schäden interessanterweise fast immer komplett aus. Der oft im juristischen Jargon formulierte Text des Lizenzvertrags wird in den seltensten Fällen gelesen oder ernst genommen.

Das beste Sicherheitssystem nützt nichts, wenn die Benutzer sich nicht adäquat verhalten. Den Benutzern muss daher der achtsame Umgang mit privaten Zugangsdaten, wie Passwörtern oder CD-Schlüsseln, beigebracht werden. Ein Schritt in die richtige Richtung sind die Sicherheitshinweise auf den Homepages von Blizzard für Benutzer der Battle.net Server [BNet]. Das System muss schwache Passwörter (vgl. Abschnitt 4.4.2) ablehnen.

Speziell in persistenten Online-Welten entsteht durch die lange Zeit, die die Teilnehmer dort miteinander verbringen können, ein starkes Gemeinschaftsgefühl und damit so etwas wie eine virtuelle Gesellschaft. Mit Sicherheit könnte dann auch die „innere Sicherheit“ dieser virtuellen Gesellschaft bezeichnet werden. Tatsächlich gibt es verteilte virtuelle Umgebungen, die Gesetze oder einen moralischen Kodex implementieren. In Ultima Online [UO] werden z.B. die Avatare von Spielern, die die Avatare „unschuldiger“ anderer Spieler töten, für einen gewissen Zeitraum als „vogelfrei“ eingestuft, und können so lange ihrerseits ohne Bestrafung getötet werden. Hersteller und Betreiber einer persistenten Online-Welt müssen hier sehr behutsam vorgehen, da zu restriktive Mechanismen den Spielspaß für einige Spieler ebenso gefährden wie zu schwache oder gar fehlende Mechanismen für andere. Die Behandlung dieser Problematik mit ihren psychologischen und soziologischen Aspekten würde allerdings den Rahmen dieser Arbeit sprengen. Einen guten Einstiegspunkt in diesen Themenbereich bieten [San00] und [Kos].

### 5.1.3 Sicherheit für Betreiber

Die Teilnehmer einer verteilten virtuellen Umgebung in militärischen, medizinischen oder industriellen Anwendungen gelten in der Regel als vertrauenswürdig. In diesen Anwendungen spielt dage-

gen die Vertraulichkeit des Inhalts des DVEs eine große Rolle. Die Daten einer virtuellen militärischen Übung sollten feindlichen Mächten nicht in die Hände fallen, ebenso wenig wie Details einer industriellen Simulation etwaige Konkurrenten etwas angehen. Im medizinischen Bereich müssen sensible Patientendaten geschützt werden.

Vertraulichkeit innerhalb einer Gruppe kann nur gewährleistet werden, wenn der Zugang zur Gruppe durch angemessene Autorisierungsmechanismen geschützt wird. Die Zugangskontrolle wird umso wichtiger, wenn Teilnehmer für die Partizipation am System bezahlen (5.1.1). In diesem Fall muss auch die Verfügbarkeit des Spiel-Dienstes gesichert sein, da ein ausgefallenes System keinen Gewinn einbringt. Speziell bei erweiterbaren DVEs ist Autorisierung auch zur Laufzeit notwendig, wenn Erweiterungen nur von bestimmten Benutzern vorgenommen werden dürfen.

Gegen das Risiko, dass ein Teilnehmer vertrauliche Daten an unberechtigte Dritte weiterleitet, gibt es kaum brauchbare IT-Lösungen; eine angemessene Bezahlung und angenehme Arbeitsbedingungen senken die Bedrohung durch Korruption sicherlich effektiver. Die Benutzer müssen außerdem durch deutliche Hinweise oder durch Schulungen auf ihre Pflicht zur Geheimhaltung vertraulicher Daten hingewiesen werden, um die unabsichtliche Weitergabe sensibler Daten zu verhindern (vgl. Abschnitt 5.1.2).

### 5.1.4 Cheating

*Cheating* ist das wohl größte und am schwierigsten zu meisternde Sicherheitsproblem in Online-Computerspielen. Yan und Choi definieren in [YC02]:

> *Any behaviour that a player may use to get an unfair advantage, or achieve a target that he is not supposed to is cheating.*

Da jeder Angriff einem Ziel dient, das der Angreifer eigentlich nicht erreichen sollte, schließt diese Definition eigentlich *jeden* erdenklichen Angriff auf ein Mehrspieler-Online-Computerspiel ein.

Das englische Verb *to cheat* deckt in seiner Bedeutung die gesamte Bandbreite vom vergleichsweise harmlosen *mogeln* oder *schummeln* bis hin zum ernsthaften *täuschen* oder gar *betrügen* ab. Entsprechend breit gefächert ist die Meinung der Spieler zu diesem Thema. Im Folgenden behalten wir daher den englischen Begriff bei.

In einem Einpersonenspiel bereitet *cheating* keine Probleme: Der Spieler muss mit sich selbst ausmachen, ob er mogeln möchte oder nicht. Viele Spielhersteller unterstützen ihn sogar dabei, indem sie beispielsweise *cheat codes* im WWW veröffentlichen. Aufrichtigen Spielern in einem Mehrspielerspiel kann dagegen durch die Anwesenheit eines einzigen *cheaters* das ganze Spiel verdorben werden. In einer Umfrage von Greenhill [Gre97] zum Online-Computerspiel *Diablo* gaben 35% der Befragten zu, bereits gemogelt zu haben, und 55% hatten einen Betrug beobachtet. Greenhill argumentiert, dass die *cheater* zwar in der Minderheit, jedoch hinreichend viele sind, um den Spielspaß der aufrichtigen Spieler zu zerstören. Die Anonymität der Spieler in einem Online-Spiel und der Mangel vieler Spiele an Detektions- und Reaktionsmechanismen führen dazu, dass ein *cheater* keine ernsthaften Konsequenzen zu befürchten hat.

Der Spielspaß spiegelt sich direkt im finanziellen Gewinn des Spielherstellers oder Server-Betreibers wider. Costikyan schreibt in [Cos99], die Art und Weise, in der die Spieler behandelt würden, entscheide über Erfolg oder Misserfolg eines Computerspiels. Eine gute Kundenbetreuung sei der Schlüsselfaktor für den Online-Erfolg. Es liegt also im direkten finanziellen Interesse des Herstellers und des Server-Providers, ein möglichst „mogel-resistentes“ Produkt herzustellen.

Da erlaubtes Verhalten in vielen Fällen schwierig von *cheating* zu unterscheiden ist, können unüberlegte Gegenmaßnahmen ein Spiel sogar unfair machen, z.B. wenn ein zu erfolgreicher Spieler

in einem *first-person-shooter* ausgeschlossen wird, weil seine schnelle Reaktion vom System als *cheating* eingestuft wird. Zwischen dem Schutz vor Betrug und der Fairness der Regeln muss daher ein ausgewogener Kompromiss gefunden werden.

Viele Spielhersteller versuchen, Implementierungsdetails durch Verschleierung zu schützen, indem sie z.B. Dateiformate geheim halten. Hacker suchen mit immensem Ehrgeiz nach neuen Möglichkeiten zum *cheating*; der damit verbundene hohe Aufwand scheint dabei kaum abzuschrecken (vgl. Abschnitt 5.1.1). Hier sind wirksame Sicherheitsmechanismen notwendig.

### Anti-Cheating-Software

Angesichts immer wiederkehrender gleicher *cheat*-Angriffe auf unterschiedliche Computerspiele haben sich auf dem Markt einige Systeme etabliert, die mit leichten Anpassungen Angriffe auf einzelne Spiele verhindern oder erschweren sollen.

*PunkBuster* [PB] besitzt eine Client-Server-Architektur. Jeder Spieler muss einen PunkBuster-Client zur Überwachung des lokalen Spiels starten. Der PunkBuster-Server sammelt die Daten der Clients ein und analysiert sie. Der Client kann den vom laufenden Spiel belegten Speicher periodisch oder auf Anfrage nach bekannten *cheats* oder unerlaubten Modifikationen scannen und hilft bei der Authentifizierung von Updates. Der Server kann automatisch Daten von den Clients einfordern, z.B. Spielerdaten, Screenshots oder Tastaturbindungen, und diese untersuchen. Ein Administrator kann daraufhin Spieler temporär oder permanent ausschließen. Viele manuelle Administratorfunktionen können auf dem Server automatisiert werden. Eine übergreifende Hardware-GUID-Datenbank (vgl. Abschnitt 4.4.1) verhindert, dass sich verbannte *cheater* erneut an einem Spiel beteiligen. PunkBuster implementiert also eine Sicherheitspolitik der Überwachung und der harten Bestrafung bei Verstößen. Für die Spieler ist PunkBuster natürlich kostenlos, aber dessen Verwendung ist für das Spiel auf bestimmten Servern vorgeschrieben. Die Herstellerfirma EvenBalance verkauft das angepasste System direkt an den Spielhersteller. Für diesen ist das Verkaufsmodell attraktiv, denn er delegiert die Lösung von Sicherheitsfragen, beispielsweise Sicherheitsupdates zur Bekämpfung neuer *cheats*, dauerhaft an die Hersteller von PunkBuster.

*Anti-cheat-Software* implementiert also zusätzliche Sicherheitsmechanismen um ein vorhandenes Computerspiel herum. Statt die Sicherheitslücken auf der Entwurfsebene zu schließen, und somit einige Angriffe im Voraus unmöglich zu machen, setzen *anti-cheat*-Programme auf Detektion von Regelverstößen und harte Bestrafung der *cheater*. Dabei werden sehr schwere Geschütze wie die Hardware-GUID aufgefahren, die bereits als Eingriff in die Privatsphäre der Spieler gewertet werden können. Dass Programme den Zugriff des Benutzers auf den eigenen Rechner begrenzen, ist man inzwischen gewohnt. Dass sie dafür eine Betriebssystemaufgabe – die Kontrolle eines anderen Programms in seiner binären Form im Speicher – übernehmen und mit anderen Rechnern in geheimer, für den Benutzer nicht nachvollziehbarer Weise kommunizieren, macht viele stutzig.

Für die Hacker lautet die logische Konsequenz, statt des Spiels selbst die Überwachungssoftware zu knacken. Wie der Spiel-Client selbst, so lässt sich auch der PunkBuster-Client nur anhand seiner Netzwerknachrichten identifizieren, und ein modifizierter Client wird nicht erkannt, solange er „richtige Antworten“ liefert. Außerdem gefährden Angriffe, beispielsweise das gelungene Vortäuschen einer falschen GUID, durch die generische Architektur von PunkBuster unter Umständen gleich mehrere Spiele auf einmal.

Ein weiteres *anti-cheat*-System ist Pandora [Pan]. Der Hersteller war allerdings zu keiner Aussage über den Funktionsumfang des Produkts zu bewegen. Hier scheint noch das Prinzip der Sicherheit durch Verschleierung zu regieren (vgl. Abschnitt 4.2.1).

## 5.2 Klassifizierung nach Sicherheitsanforderungen

Die Sicherheitsanforderungen verschiedener DVEs hängen extrem von der Anwendung ab. Daher klassifiziere ich an dieser Stelle verteilte virtuelle Umgebungen nach ihren Sicherheitsanforderungen.

In *kollaborativen DVEs* steht die Zusammenarbeit im Vordergrund. In vielen industriellen Simulationen soll beispielsweise ein Produkt gemeinsam und gleichzeitig entwickelt werden. Da dabei keiner gegen andere gewinnen kann, spielt *cheating* keine Rolle. Konflikte können weniger autoritär aufgelöst werden, und das Peer-to-Peer-Paradigma ist anwendbar. Im Gegensatz dazu stehen bei *kompetitiven DVEs* Konkurrenz und Wettkampf im Vordergrund. Die meisten Mehrspieler-Computerspiele fallen in diese Kategorie. Da jeder das Spiel gewinnen will, trauen die Teilnehmer einander nicht. Diese Umgebungen sind hochgradig *cheat*-gefährdet. Ohne eine maßgebliche neutrale Regelungsinstanz sind diese Umgebungen nicht zu meistern.

Wie schon in Abschnitt 5.1.1 erläutert wurde, gibt es auch finanzielle Sicherheitsinteressen zum Betrieb eines DVEs. Daher erfolgt eine weitere Unterscheidung in *kommerzielle DVEs*, bei denen es für den Hersteller oder Betreiber um Gewinn geht, und *freie DVEs*, bei denen die Teilnahme keine Gebühr erfordert. Viele VR-Chat-Umgebungen sind frei, während die meisten Computerspiele kommerziell sind, sei es durch eine Benutzungsgebühr oder den Verkauf der Teilnehmersoftware.

Unterschiede zeigen sich auch in der Zugangspolitik: In *offenen DVEs* darf jeder erreichbare Rechner partizipieren, während *geschlossene DVEs* nur ausgesuchten Teilnehmern den Zugang gestatten. Im zweiten Fall muss also eine Zugangskontrolle implementiert werden.

Ähnlich kann man zwischen *öffentlichen DVEs* und *vertraulichen DVEs* unterscheiden, wobei im ersten Fall der Nachrichtenverkehr für alle lesbar übertragen wird, während im zweiten Fall der Inhalt der Nachrichten vertraulich ist, und beispielsweise durch Verschlüsselung geschützt werden muss.

Militärische und industrielle Simulationen werden aufgrund der einfacheren Implementierung und der besseren Performance überraschend häufig als *offene* und *öffentliche DVEs* konzipiert, werden dann zum Schutz der Vertraulichkeit und des Zugangs allerdings auf privaten Firmen- oder Institutionsnetzwerken betrieben.

Die Erweiterung der Szene, z.B. durch Einbringen neuer Entitäten oder Anfügen neuer Räume, ist ebenfalls ein Sicherheitsaspekt, denn nicht jeder sollte beliebige Änderungen an der Szene vornehmen dürfen. Daher differenzieren wir zwischen *erweiterbaren* und *statischen DVEs*.

Eine interessante Kombination ist *HalfLife-TV* [HLTV], eine Erweiterung des Computerspiel Klassikers HalfLife, bei dem zwar nur eine kleine Gruppe Spieler Nachrichten senden darf, und somit aktiv am Spiel teilnimmt, der dabei produzierte Datenverkehr aber öffentlich empfangen werden kann, um das Spiel passiv wie einen Film anzusehen. In unserer Kategorisierung ist HalfLife-TV also ein *kompetitives*, *kommerzielles*, *geschlossenes* und gleichzeitig *öffentliches DVE*.

Um eine konkrete Sicherheitsarchitektur zu entwerfen, betrachten wir für den Rest der Arbeit zwei Beispielszenarien, die in den folgenden Absätzen dargestellt werden.

### 5.2.1 Szenario I: Fantasy-Rollenspiel

Das erste Szenario ist ein Fantasy-Rollenspiel, das über das Internet gespielt werden kann. Die Spieler bewegen ihre Avatare in einer Märchenwelt und versuchen, durch Kämpfe und das Sammeln von Erfahrungspunkten und Gegenständen ihren Charakter zu einem immer mächtigeren Wesen zu entwickeln. Den Spielern wird vom Betreiber eine zeitabhängige Benutzungsgebühr in

Rechnung gestellt.

Es handelt sich also um ein kompetitives, kommerzielles DVE. Da Spieler für die Teilnahme bezahlen müssen, muss das Spiel geschlossen sein. Der Nachrichtenverkehr ist öffentlich. Durch den stark kompetitiven Inhalt des Spiels besteht eine hohe Gefahr des *cheating*. Der Betreiber sollte Sicherheitsvorkehrungen treffen, um die fairen Spieler in seinem Kundenkreis zu halten.

Die Entwicklung eines Avatars zieht sich über mehrere Sitzungen hin. Um eine unerlaubte Manipulation der Avatare auszuschließen, muss der Betreiber deren persistente Speicherung übernehmen und ihre alleinige Nutzung durch den jeweiligen Besitzer garantieren.

### 5.2.2 Szenario II: Innenarchitektur-Simulation

Ein Innenarchitektenbüro verwendet als Entwurfshilfe für die Einrichtung eines großen Gebäudes ein verteiltes VR-System, das es mehreren Mitarbeitern verschiedener Außenstellen ermöglicht, gleichzeitig die unterschiedlichen Teile des Gebäudes einzurichten.

Davon ausgehend, dass alle Mitarbeiter von einem guten Entwurf profitieren, handelt es sich um ein kollaboratives DVE. Weil der Betrieb vom Unternehmen selbst übernommen wird, wird keine Teilnahmegebühr erhoben. Der Inhalt des DVEs ist allerdings vertraulich, da Konkurrenten den Entwurf ausspionieren könnten. Da ein Architektenbüro in der Regel keine exklusiven Netzwerkverbindungen zwischen den einzelnen Außenstellen unterhält, müssen die Daten über das Internet transportiert werden, und zum Schutz verschlüsselt werden. Der Zugang sollte nur authentifizierten Mitarbeitern möglich sein. Die Verfügbarkeit des Systems ist wichtig, da die Arbeitszeit der Mitarbeiter kostbar ist.

Die wichtigste Eigenschaft der Szene ist ihre Erweiterbarkeit, wobei die Erweiterungen und Änderungen, verglichen mit den Update-Nachrichten des Fantasy-Rollenspiel-Szenarios, mit vergleichsweise niedriger Häufigkeit auftreten werden. Auch Kollisionen sind dabei relativ selten. Deshalb können zur Konsistenzsicherung langsamere und zuverlässigere Methoden eingesetzt werden.

Die von den Mitarbeitern geleistete Arbeit steckt im Zustand der Szene. Dieser muss auf jeden Fall persistent abgespeichert werden können. Hierbei muss die Vertraulichkeit der Daten gewährleistet werden.

Damit die simulierten Räume einen realistischen Eindruck bieten, sollte die Steuerung eines Avatars mit kurzen Antwortzeiten möglich sein.

## 5.3 Mögliche Angriffe auf DVEs

Eine vollständige Liste aller möglichen Angriffe kann es wohl niemals geben, da auch die Art der Angriffe sich mit der Zeit entwickelt. Außerdem sind konkrete Angriffe stark von der spezifischen Anwendung der verteilten virtuellen Umgebung abhängig. Dennoch werden im Folgenden die wichtigsten Angriffe dargestellt.

Wie schon in 5.1.4 beschrieben, fallen die meisten Angriffe auf ein Online-Computerspiel in die Rubrik *cheating*. Mehrere Autoren haben bereits eine Taxonomie von *cheat-Angriffen* [Pri00, YC02, HM03] aufgestellt. Die ausführlichste von Yan und Choi [YC02] beschreibt beispielsweise elf Angriffskategorien:

1. Kollusion[1]

[1] unerlaubte Absprache oder betrügerische Zusammenarbeit von Teilnehmern

2. Missbrauch von Grundsätzen (englisch *policy*) und Vorgehensweisen (englisch *procedure*)
3. Manipulation virtueller Entitäten
4. Angriff auf Passwörter
5. Verweigerung eines Dienstes gegenüber anderen Spielern
6. Angriff durch fehlende Geheimhaltung
7. Angriff durch fehlende Authentifizierung
8. interner Missbrauch
9. soziale Manipulation
10. Modifikation der Spielsoftware oder -daten
11. Ausnutzen eines Fehlers oder einer Designschwäche

Alle diese Einteilungen basieren auf der *Angriffsmethode*. Für die Entwicklung einer Sicherheitsarchitektur hat sich das leider als wenig hilfreich herausgestellt, denn die meisten Kategorien liefern keinen Hinweise darauf, welche Teile des Systems der Angriff trifft, oder an welcher Stelle welche Gegenmaßnahmen ergriffen werden können. Außerdem fallen viele Angriffe gleichzeitig in mehrere Kategorien: Ein von einem Teilnehmer gefälschter Zeitstempel eines Ereignisses ist z.B. eine Datenmodifikation (10) und kann die Manipulation einer virtuellen Entität (3) zur Folge haben, wobei der Grundsatz (2) ausgenutzt wird, dass Spieler die Zeitstempel selbst verfassen, was wiederum eine Designschwäche (11) darstellt.

Um den Entwurf einer Sicherheitsarchitektur zu erleichtern, und um auch andere verteilte virtuelle Umgebungen als nur Online-Spiele mit abzudecken, stelle ich in dieser Arbeit eine neue Taxonomie vor, die sich stärker an den einzelnen Komponenten des Systems und deren Eigenschaften orientiert. Eine eindeutige Kategorisierung von Angriffen bleibt jedoch schwierig.

### 5.3.1 Angriffe auf Schwachstellen des Kommunikationsprotokolls

Da der gemeinsame Zustand der Szene mittels Netzwerkkommunikation aufrechterhalten wird, hängt die Konsistenz der Szene direkt von der Integrität des Kommunikationsprotokolls ab.

Wie in Kapitel 3 beschrieben wurde, kommen aus Leistungsgründen viele Protokolle nicht infrage, obwohl sie entsprechend nützliche Qualitätsmerkmale aufweisen. Die Art der durch das Kommunikationsprotokoll ermöglichten Angriffe hängt davon ab, welcher Kompromiss hier geschlossen wurde. Im Folgenden werden daher die Angriffe beschrieben, die durch das Fehlen bestimmter Eigenschaften möglich werden.

#### Integrität

Besitzt das Protokoll keine Integritätsgarantie, können Nachrichten von Angreifern verändert werden. Dies stellt insbesondere eine Gefahr bei der Verwendung von Overlay-Netzwerken dar, bei denen Teilnehmer die Rolle von Routern übernehmen und Nachrichten anderer Teilnehmer weiterreichen. Die Integrität von Nachrichten kann durch MACs gesichert werden, deren Schlüssel nur die kryptographischen Endpunkte der Verbindung kennen.

Ebenso kann ein Angreifer ungültige Nachrichten senden. Die mit am häufigsten auftretenden *buffer-overflow-Angriffe* beruhen auf der Schwachstelle einiger Programme, keine Indexüberprüfung für Feldindizes durchzuführen. Dadurch können andere Daten unerlaubt überschrieben werden oder in besonders schweren Fällen sogar beliebige Programmabschnitte auf dem Zielrechner ausgeführt werden. Schutz vor *buffer-overflow-Angriffen* bietet eine strenge Überprüfung aller eingehenden Nachrichten vor deren Bearbeitung oder die Verwendung einer Programmiersprache, die Pufferüberläufe abfängt und Feldindizes prüft, wie z.B. Java.

### Ordnung von Ereignissen

Zur zeitlichen Anordnung von Ereignissen ist man, wenn das Protokoll keine Ordnungsgarantie liefert, auf Zeitstempel oder Sequenznummern in den Nachrichten angewiesen. Werden diese von einem Angreifer gefälscht, können Ereignisse vor- beziehungsweise zurückdatiert werden. Beide Angriffe können sich lohnen: Durch Vordatierung in die Zukunft könnte ein Online-Spieler versuchen, sich den erster Zugriff auf einen noch gar nicht vorhandenen Gegenstand zu sichern, z.B. auf die Schätze eines Monsters, das noch von mehreren Spielern angegriffen wird. Ein in der Vergangenheit liegender Zeitpunkt wäre günstig, um zu behaupten, dass der Avatar die Stelle, an der gerade eine Bombe explodiert, längst verlassen hat. In kompetitiven DVEs ist daher von Zeitstempeln, die von Teilnehmern generiert werden, abzuraten.

Unabsichtlich falsche Zeitstempel können entstehen, wenn ein Angreifer die Uhr eines Teilnehmers durch gefälschte Nachrichten eines Synchronisationsprotokolls manipuliert. Eine dadurch fälschlich vorgehende Uhr kann z.B. die Reaktion des Teilnehmers verlangsamen, während eine nachgehende Uhr vermeintlich alte und daher ungültige Nachrichten zur Folge hat. Zeitabhängige Protokolle wie TESLA (vgl. Abschnitt A.6.3) reagieren natürlich besonders empfindlich auf solche Veränderungen. Synchronisation über das Netzwerk sollte daher nur mit authentisierten vertrauenswürdigen Teilnehmern vorgenommen werden. Das *Network Time Protocol* (NTP, [NTP]) bietet hier die Möglichkeit, die ausgetauschten Pakete mit MACs zu authentisieren.

### Verlässlichkeit

Fehlt dem Kommunikationsprotokoll eine Auslieferungsgarantie, so hat ein Teilnehmer die Möglichkeit, den Empfang von Nachrichten abzustreiten. Ein Online-Spieler wird dann beispielsweise unverwundbar, indem er den Empfang aller Nachrichten abstreitet, die einen Angriff auf seinen Avatar darstellen. Ebenso kann er behaupten, dass eine angeblich von ihm gesendete Nachricht leider von den anderen Teilnehmern nicht empfangen wurde, z.B. die Nachricht, mit der er seinen Avatar längst aus der Gefahrenzone bewegt hat, in der er gerade angegriffen wird.

Werden trotzdem Protokolle ohne Auslieferungsgarantie in einem kompetitiven DVE verwendet, so sollten die Teilnehmer keine maßgeblichen Entscheidungen treffen können, in denen sie von der Unverlässlichkeit profitieren können. Lässt es die Netzwerkbandbreite zu, so sollten absolute Updates verwendet werden, die sich nicht auf einen vorherigen Zustand beziehen. Tatsächlich verloren gegangene Nachrichten müssen dann nicht erneut gesendet werden, da sie durch die nächste empfangene Nachricht obsolet werden.

### Latenz

Was passiert, wenn die Benutzer stark unterschiedliche Latenzzeiten haben? Das DVE könnte versuchen, die technisch bedingte langsame Reaktion einiger Teilnehmer fair auszugleichen, indem

es diesen einen kleinen Vorteil einräumt, beispielsweise Ereignisse zurückdatiert. Ein Teilnehmer könnte dann absichtlich alle unwichtigen Nachrichten verzögern und damit eine höhere Netzwerklatenz vortäuschen, während er wichtige Nachrichten sofort absendet, und damit auf unfaire Weise bevorzugt wird.

Ein fairer Kompromiss zwischen Benutzern mit unterschiedlichen Verbindungslatenzen muss behutsamer gewählt werden. Um überhaupt Konsistenz und Echtzeitfähigkeit in ein gesundes Gleichgewicht zu bringen, muss die maximale Latenz einer Teilnehmerverbindung begrenzt werden. Innerhalb dieser Grenzen muss das System Fairness gewährleisten.

### Bandbreite

Da die Bandbreite einer Netzwerkverbindung beschränkt ist, kann ein *Überflutungsangriff* (englisch *flooding attack*) geführt werden. Dabei werden entweder einzelne Teilnehmer oder sogar der gesamte Verbund mit beliebigen Nachrichten überflutet. Resultat ist die *Dienstverweigerung* des Systems (englisch *Denial of Service*) (DoS), weil für den regulären Nachrichtenverkehr nicht mehr genügend Bandbreite zur Verfügung steht. Besonders bösartig sind solche Angriffe, wenn sie von mehreren Rechnern aus gleichzeitig geführt werden (englisch *Distributed Denial of Service*) (DDoS). Auch in DVEs besteht das Risiko solcher Angriffe, z.B. um industrielle oder militärische Simulationen zu sabotieren, um einen Gegner in einem Mehrspieler-Spiel nicht zum Zuge kommen zu lassen oder aus Rache für den Ausschluss aus einem Spiel. Da DVEs besonders hohe Anforderungen an den Durchsatz des Verbindungsnetzwerks haben, reagieren sie sogar besonders empfindlich auf DoS-Angriffe. Zentrale Dienste des Systems müssen daher besonders geschützt werden.

Eine effiziente Verteidigung gegen Überflutungsangriffe ist extrem aufwändig. Sie erfordert beispielsweise ein großes Netzwerk mit vielen alternativen Verbindungsmöglichkeiten, die Replikation zentraler Dienste und ein Detektionssystem, das auch während eines Angriffs noch die Abstimmung der verschiedenen Replica ermöglicht. Für die meisten DVEs stehen diese Ressourcen nicht zur Verfügung.

Trotzdem sollte man auch aus Skalierbarkeitsgründen zentrale Dienste replizieren und möglichst wenig globalen Netzwerkverkehr produzieren, damit nur ein Teil des Verbunds von Überflutungsangriffen betroffen ist.

### Verletzung des Protokolls

Insbesondere Protokolle, die eine feste Abfolge von Nachrichten benötigen, sind durch Verletzungen gefährdet: Ein Angreifer kann diese Protokolle initiieren und im späteren Verlauf abbrechen, verzögern oder nicht mehr bzw. falsch antworten. Wenn der Verarbeitungsaufwand auf dem Zielrechner groß genug ist, lassen sich auf diese Weise DoS-Angriffe durchführen. Werden Verstöße gegen das Protokoll nicht abgefangen, kann der Zielrechner sogar komplett ausfallen.

Ein sehr bekannter Vertreter dieser Angriffskategorie beruht auf der Verletzung des *TCP/IP-handshake-Protokolls* (vgl. Abschnitt 3.3.4): Im *SYN-Flooding-Angriff* initiiert der Angreifer eine Unmenge von TCP/IP-Verbindungen, ohne den *handshake* zu Ende zu führen. Der Puffer, in dem der Zielrechner alle diese halb geöffneten Verbindungen speichert, kann dann überlaufen, mit dem Resultat, dass der Zielrechner keine weiteren Verbindungen mehr zulässt. Da Gegenmaßnahmen nur auf der Betriebssystemebene oder in der Router-Konfiguration ergriffen werden können, sind diese Angriffe für diese Arbeit nicht weiter relevant.

Jede Protokollimplementierung muss die Möglichkeit eines böswilligen Gegenübers berücksichtigen. Ein rechnerisch aufwändiges Protokoll sollte immer einen gewissen Vorschuss an Arbeit auf der Initiatorseite fordern: Bevor beispielsweise ein Server eine Nachricht digital signiert, sollte der Client eine ähnlich aufwändige Operation durchgeführt haben. Die entsprechende Nachricht muss einen einfach zu verifizierenden *Beweis ihrer Neuheit* (englisch *proof of freshness*) enthalten, um Wiederholungsangriffe (englisch *replay attacks*) auszuschließen. Der Server kann schwache Authentisierung durch ein *cookie* (vgl. Abschnitt 4.6.2) realisieren.

Wenig zeitkritische Protokolle, beispielsweise zur Anmeldung neuer Teilnehmer, können auf der Server-Seite von einem eigenen Thread mit niedriger Priorität durchgeführt werden und auf eine feste Anzahl von Durchläufen pro Zeitinterval begrenzt werden.

Allgemein sollte alles überprüft werden, sobald die notwendigen Informationen vorliegen, und dabei die schnelleren Überprüfungen zuerst durchgeführt werden: Bei einer digital signierten, mit einem Zeitstempel und einem Cookie versehenen Nachricht wird also zuerst der Zeitstempel, dann das Cookie und zuletzt die digitale Signatur geprüft.

## 5.3.2 Regelbruch

Entitäten einer verteilten virtuellen Umgebung verhalten sich nach gewissen Gesetzmäßigkeiten. Oft können sich Avatare beispielsweise nur nach bestimmten physikalischen Gesetzen bewegen oder andere Entitäten nicht durchdringen. Im Folgenden sind die Angriffe dargestellt, die über einen Bruch solcher Regeln ausgeführt werden.

### Manipulation des lokalen Zustands

Der lokale Zustand der Szene, den jeder Teilnehmer zu deren Darstellung benötigt, entzieht sich der Kontrolle von außen. Böswillige Spieler können hier beliebige Änderungen vornehmen, ohne dass diese nach außen sichtbar werden.

Håkon und Mørch [HM03] teilen diese Angriffe in *Objektkorrelation* (englisch *object correlation*) und *Objektaufdeckung* (englisch *object exposure*) ein. *Objektkorrelation* verbindet Objekte miteinander, z.B. eine Gewehrkugel mit der Position eines Gegners in einer *automatischen Zielhilfe* (englisch *aiming proxy*).[2] *Objektaufdeckung* macht für den Spieler eigentlich unsichtbare Informationen sichtbar, z.B. werden Wände durchsichtig oder es werden zu jedem gegnerischen Avatar zusätzliche Informationen wie dessen momentane Lebensenergie angezeigt.

Die automatische Zielhilfe ist ein interessantes Beispiel dafür, dass sich die Grundlagen der Sicherheitsanalyse angesichts schummelnder Spieler von denen konventioneller Systeme unterscheiden. Hier kommuniziert der Spieler nicht direkt mit dem Server, sondern schaltet eine weitere Entität dazwischen und startet so einen *man-in-the-middle-Angriff*, der in der konventionellen Sicherheitsanalyse nur von böswilligen Dritten geführt wird. Die Kooperation des Spielers mit der Zielhilfe macht eine zuverlässige Detektion theoretisch unmöglich, da der Spieler alle Informationen zu seiner Identifizierung mit der Zielhilfe teilen kann.

Objektaufdeckung kann am erfolgreichsten verhindert werden, indem ein Teilnehmer nur die Informationen bekommt, die in seinem Sichtbereich liegen.

[2] In [HM03] werden auch Hilfen, die Gegner besser sichtbar machen, als *Objektkorrelation* aufgeführt, die eigentlich eher *Objektaufdeckung* darstellen.

### Ungültige Ereignisse

Ein böswilliger Teilnehmer kann ungültige Ereignisse erzeugen, die den Regeln des DVE widersprechen. Beispielsweise könnte ein Spieler in einem *first-person-shooter* versuchen, sich schneller als erlaubt zu bewegen. In einem kompetitiven DVE müssen daher alle Ereignisse auf ihre Gültigkeit hin überprüft werden. Die Regeln dazu müssen deterministisch auswertbar sein. Ist Prüfung auf Gültigkeit zu aufwändig, erhält das System einen weiteren Angriffspunkt für DoS-Angriffe (vgl. Abschnitt 5.3.1); denn ein Angreifer kann große Mengen ungültiger Nachrichten versenden, deren Prüfung die kontrollierenden Rechner komplett auslastet.

Hier gilt das Prinzip „überprüfe, sobald möglich“: Je früher ein ungültiges Ereignis die Verarbeitungspipeline verlässt, desto besser. Die Überprüfungen müssen so einfach wie möglich gehalten werden. Mit Hashes oder MACs können sinnlose und nicht dem Protokoll entsprechende Nachrichten noch vor der Deserialisierung aussortiert werden. Ereignis-Objekte sollten als erstes auf gültige Werte geprüft werden, usf.

Besonders schwer zu detektieren sind ungültige Ereignisse, die auf *race conditions* beruhen, also aufgrund fehlender Synchronisation mehrerer Threads entstehen. In einigen Spielen kann man beispielsweise einen Gegenstand verdoppeln, indem man ihn fallen lässt und sofort wieder aufnimmt. Für die Konsistenz wichtige mehrfädige Programmabschnitte müssen sehr sorgfältig synchronisiert oder im Zweifelsfall sequentiell ausgeführt werden.

### Ungültige Erweiterung

DVEs, die die Einbindung neuer Entitäten oder Ereignistypen zur Laufzeit ermöglichen, können besonders leicht einem DoS-Angriff zum Opfer fallen. Der Angreifer registriert beispielsweise einen neuen Ereignistypen, dessen Bearbeitung immer einen Fehler auslöst oder in eine Endlosschleife läuft. Besonders schwierig ist die Ursache zu finden, wenn sich der Effekt erst mit einer gewissen Verzögerung oder auf gewisse Ereignisse hin zeigt.

Die Szene darf also nur von vertrauenswürdigen Teilnehmern erweitert werden. Die dabei verwendete Authentisierung muss entsprechend stark sein (vgl. Abschnitt 5.3.3).

Die hinzugefügten Entitäten dürfen nur über eine minimale Schnittstelle mit der Anwendung kommunizieren, und dürfen weder eigene Threads starten noch Netzwerkverbindungen aufbauen. In Java kann man dazu die neuen Entitäten in einer eigenen *Java-sandbox* (vgl. Abschnitt B.1) ausführen.

### Kollusion

*Kollusionsangriffe* gehören zu den am schwierigsten aufzudeckenden Angriffen. Hierbei arbeiten mehrere Teilnehmer in unerlaubter Weise zusammen, um Informationen, die dem jeweils anderen nicht bekannt sein dürfen, auszutauschen. In einem *first-person-shooter* kann sich ein Spieler, der zwei Rechner besitzt, gleichzeitig in zwei gegeneinander kämpfenden Teams anmelden, und somit stets über die aktuelle Strategie und die Position der Mitglieder beider Gruppen Bescheid wissen, was ihm einen enormen Vorteil verschafft. Der Nachweis, dass hier geschummelt wurde, kann eigentlich nur dadurch endgültig erbracht werden, dass sich ein Spieler selbst verrät. Andere Gegenmaßnahmen müssen den Kompromiss zwischen Schutz vor Betrug und Fairness (vgl. Abschnitt 5.1.4) berücksichtigen.

Bei kommerziellen DVEs sollte die unerlaubte Weitergabe von Zugangsschlüsseln möglichst unattraktiv gemacht werden, z.B. indem ein Zugangsschlüssel einem Teilnehmer nur die Kontrolle

des eigenen Avatars ermöglicht. Wer zahlt schon freiwillig dafür, einen Freund den eigenen Charakter steuern zu lassen? Gruppenschlüssel kommen hier bestimmt nicht infrage.

### 5.3.3 Angriffe auf Authentisierungen

Die Angriffe in dieser Rubrik können in der Regel durch starke Authentifizierungsmechanismen verhindert werden.

#### Zugangskontrolle

Die Zugangskontrolle zu einem DVE kann auf unterschiedliche Weise angegriffen werden. Zum einen können Angreifer versuchen, in das System einzubrechen, um z.B. die Benutzungsgebühren eines Online-Spiels zu sparen.

Auch Vertraulichkeit setzt Authentisierung voraus. Ein Unberechtigter, dem es gelingt sich anzumelden, erlangt dadurch meist schon den Zugang zu vertraulichen Daten.

#### Unerlaubter Zugriff auf fremde Entitäten

Wie in Abschnitt 2.5.3 beschrieben wurde, sollte jede Entität zu einem festen Zeitpunkt einen eindeutigen Besitzer haben. Ein Angreifer könnte versuchen, Ereignisse einer Entität, die eigentlich von einem anderen Teilnehmer kontrolliert wird, zu erzeugen. Auf diese Weise können z.B. virtuelle Entitäten gestohlen werden. Über die Wichtigkeit des Schutzes virtuellen Eigentums wurde bereits in Abschnitt 5.1.2 ausführlich berichtet.

Übernimmt ein Angreifer gar die Kontrolle über einen fremden Avatar (englisch *impersonation*), so kann er im Namen des eigentlichen Besitzers großen Schaden anrichten. Auch in offenen Systemen, wie virtuellen Chat-Rooms, die keine besonderen Zugangsprivilegien erfordern, sollten solche Übernahmeangriffe verhindert werden. Zum einen ist es für Benutzer sehr erniedrigend, die Kontrolle über das virtuelle Ich zu verlieren (vgl. [Dib]), und zum anderen sollten Benutzer auch keine unschuldigen Teilnehmer eines angeblichen Übernahmeangriffs bezichtigen können.

#### Bruch von Privilegien des Systems

Einige Aktionen des Systems außerhalb der Szene erfordern ebenfalls besondere Privilegien. Ein Angriff dieser Kategorie besteht beispielsweise in einer unberechtigten Erweiterung der Szene, bei der ungültige Daten das System stören, beleidigende oder politisch unkorrekte Inhalte eingebracht werden oder ähnliches.

Die Änderung der Benutzerdaten erfordert in der Regel Administratorrechte und kann ebenfalls von Unberechtigten angegriffen werden.

### 5.3.4 Angriffe auf die Vertraulichkeit

Einige DVEs speichern sensible Daten, die attraktive Angriffsziele darstellen und daher vor unberechtigtem Zugriff geschützt werden müssen. Dazu gehören:

- Persönliche Benutzerdaten, wie Name, Adresse, Geburtsdatum, E-Mail-Adresse und Kreditkarteninformationen.

- Vertrauliche Inhalte der Umgebung, z.B. industrielle oder militärische Geheimnisse, medizinische Daten von Patienten.
- Teilnehmerdaten, beispielsweise IP-Adresse, Betriebssystem(-version), die für direkte Angriffe genutzt werden können, sowie auf den Teilnehmern gespeicherte Daten, die nicht mit dem DVE in Zusammenhang stehen.
- Schlüsselmaterial, wie Passwörter, Gruppenschlüssel, Lizenz- oder CD-Codes.

Auch Laufzeitdaten können für einen Abhörangriff interessant sein. So sollte vertraulicher Chat zwischen Mitgliedern einer Online-Community nicht abhörbar sein.

Vertraulichkeit von Daten, die über öffentliche Netzwerke geschickt werden, wird durch Verschlüsselung realisiert. Verschlüsselung allein macht allerdings noch kein System sicher: Das verwendete Schlüsselmaterial darf nur berechtigten Teilnehmern bekannt sein, und diese müssen zu diesem Zweck authentifiziert werden. Schlüssel sollten nur eine gewisse Lebensdauer haben und müssen nach Verwendung zerstört werden (vgl. Abschnitt 4.2.2). Die Menge der Daten, deren Sicherheit von einem Schlüssel abhängt, muss begrenzt sein. Korrumpierte Schlüssel müssen durch neue ersetzbar sein (vgl. Abschnitt 4.4.4).

## 5.4 Konsequenzen aus der Angriffsanalyse

Die geschilderten Angriffe sind in verschiedenen Anwendungsfällen unterschiedlich wahrscheinlich und in ihren Auswirkungen unterschiedlich gefährlich. Im Folgenden werden die Konsequenzen der Angriffsanalyse für die in den Abschnitten 5.2.1 und 5.2.2 beschriebenen Szenarien beschrieben.

### 5.4.1 Teilnehmer-Registrierung durch Zertifikate

In beiden Szenarien müssen Teilnehmer registriert werden. Das Ergebnis der Registrierung sollte ein Berechtigungsnachweis mit langer Lebensdauer sein, damit die persönlichen Angaben nur einmal angegeben werden müssen. In dieser Arbeit werden dazu aus folgenden Gründen *Zertifikate* eingesetzt:

- Ein gemeinsames Wurzel-Zertifikat für die gesamte Umgebung stellt kein Problem dar, da die Benutzer dem Betreiber ohnehin vertrauen müssen.
- Die geheimen Zugangsinformationen, also die privaten Schlüssel, werden dezentral gespeichert. Ein zentraler Speicher wäre ein interessantes Angriffsziel.
- Zertifikate selbst unterliegen keiner Geheimhaltung und könne in öffentlich zugänglichen Verzeichnissen gespeichert werden.
- Mit Zertifikaten können Aufgaben, insbesondere die Ausstellung von Zertifikaten selbst, einfach delegiert werden.
- Registrierungs- und Laufzeitumgebung können getrennt werden.
- Viele weitere Sicherheitsprotokolle, z.B. einige aus Abschnitt 4.6.2, sind in Zertifikaten und digitalen Signaturen verankert.

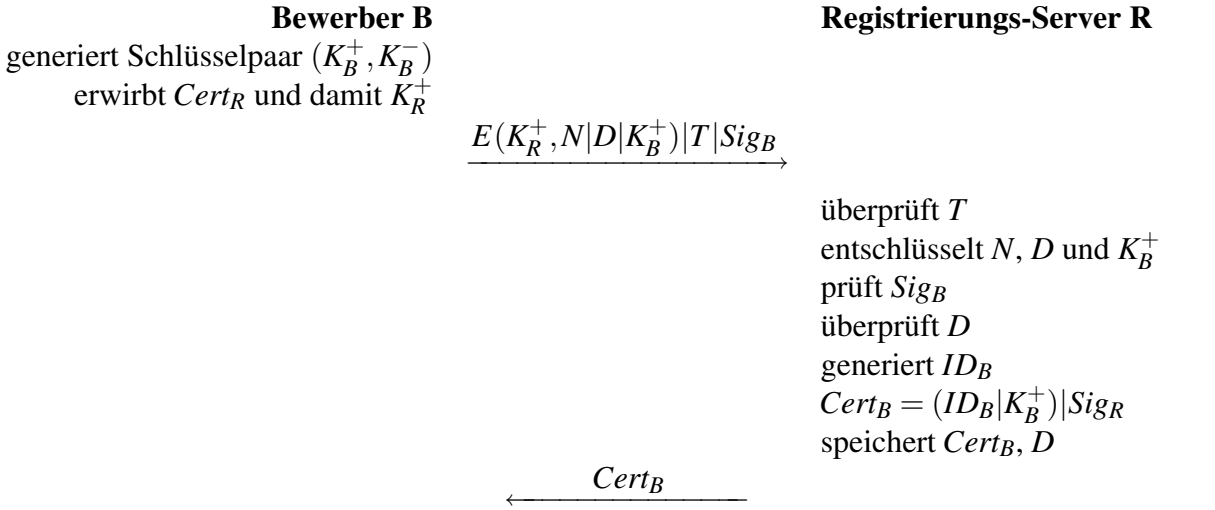



Abbildung 5.1: Zertifizierungsprotokoll

Zur Registrierung generiert ein *Bewerber B* zunächst ein Schlüsselpaar $(K_B^+, K_B^-)$. Mit dem privaten Schlüssel $K_B^-$ signiert er einen *Zertifizierungsantrag* (englisch *Certificate Signing Request*) (CSR), der persönliche Daten $D$, beispielsweise Name, Adresse und falls erforderlich Zahlungsinformationen sowie den öffentlichen Schlüssel $K_B^+$ enthält. Die Daten werden aus Datenschutzgründen mit dem öffentlichen Schlüssel des *Registrierungs-Servers R* verschlüsselt und übermittelt. Der Registrierungs-Server fungiert als *Certification Authority* (CA). Er prüft die Angaben, ordnet dem Bewerber einen eindeutigen Bezeichner $ID_B$ zu und stellt mithilfe des privaten Schlüssels zu seinem *Zugangs-Zertifikat* ein *Teilnehmer-Zertifikat* $Cert_B$ aus, das den Bezeichner $ID_B$ an den öffentlichen Schlüssel $K_B^+$ bindet. Für die Anmeldung an die Simulation benötigt der Teilnehmer zusätzlich nur noch den privaten Schlüssel, den er lokal und mit einer Passphrase verschlüsselt speichert.

Das Zertifizierungsprotokoll ist in Tabelle 5.1 skizziert. Es kann asynchron ausgeführt werden, z.B. per E-Mail. Die Verschlüsselung bewirkt, dass keine dem Datenschutz unterliegenden Daten öffentlich kommuniziert werden. Eine externe Registrierung, beispielsweise per Post, ist theoretisch ebenso möglich, doch da für Menschen schwer lesbare öffentliche Schlüssel angegeben werden müssen, wird hier davon abgesehen.

Für noch größere Sicherheit wird der Registrierungs-Server in zwei Komponenten unterteilt. Die *Registration Authority* (RA) fungiert als Benutzerschnittstelle und regelt alle Aufgaben, die eine Online-Verbindung erfordern. Sie nimmt die CSRs entgegen, prüft die Angaben und sendet die ausgestellten Zertifikate an den Bewerber, bzw. legt sie in einem öffentlichen Verzeichnis ab. Die CA besitzt keine eigene Verbindung zum öffentlichen Netzwerk, kann aber mit der RA kommunizieren. Sie speichert den privaten Schlüssel, stellt die Zertifikate aus und signiert Widerrufslisten. Außerdem speichert sie alle persönlichen Daten aus den CSRs und alle ausgestellten Zertifikate und Widerrufslisten.

Ist maximale Sicherheit erforderlich, wird die Kommunikation zwischen RA und CA per Hand, also über ein tragbares Speichermedium, durchgeführt. Die CA kann dann vollständig vom Netzwerk getrennt werden, und der private Schlüssel stellt somit keinen Angriffspunkt mehr dar.

Mit einem gemeinsamen Wurzel-Zertifikat können mehrere Registrierungs-Server gleichzeitig betrieben werden. Die einzelnen Registrierungs-Zertifikate werden mit dem zum Wurzel-Zertifikat gehörigen privaten Schlüssel signiert. Um Kollisionen und zusätzliche Kommunikation zu vermeiden, müssen die Registrierungs-Server disjunkte Teilnehmer-IDs generieren.

Im Falle mehrerer Registrierungs-Server kann man einen guten Kompromiss zwischen Performance und Sicherheit schließen, indem die Wurzel-CA, wie oben beschrieben, vom Netz getrennt arbeitet, während die einzelnen Registrierungs-Server im Netzwerk arbeiten. Der Schaden, den ein unbrauchbar gewordener privater Schlüssel eines Registrierungs-Servers anrichtet, wird begrenzt, da nur die damit signierten Benutzer-Zertifikate ungültig werden und neu beantragt werden müssen. Gleichzeitig wird die Zertifizierungszeit verkürzt, da die Registrierungs-Server parallel und im Netzwerk arbeiten.

Für den Widerruf von Zertifikaten wird nicht das OCSP (vgl. Abschnitt 4.4.6) eingesetzt, da sowohl Anfrage als auch die Antwort digital signiert, und somit aufwändig zu erzeugen und zu verifizieren sind. CRLs bündeln mehrere Widerrufe unter einer einzigen Signatur und eignen sich daher hervorragend zur persistenten Speicherung im Verzeichnis. Aktuelle Widerrufsinformationen werden über die ohnehin authentisierten Kommunikationskanäle der Laufzeitumgebung kommuniziert.

### 5.4.2 Verzeichnisdienste

Um überhaupt an einem DVE teilzunehmen, benötigt ein Benutzer als erstes einen Einstiegspunkt, beispielsweise die Adresse eines Registrierungs-Servers. In dieser Arbeit wird hierzu ein LDAP-Verzeichnisdienst benutzt (vgl. Abschnitt 3.3.7). In seinem Verzeichnis werden die Adressen, Typen und Daten verschiedener Server-Dienste sowie die Zertifikate aller Teilnehmer gespeichert.

Für die verschiedenen Einträge können dabei unterschiedliche Zugriffsrechte erteilt werden. So können Registrierungs-Server beispielsweise neue Benutzer-Zertifikate ablegen, während nicht authentisierte Benutzer nur einen Registrierungs-Server finden können, weitere Einträge aber erst nach Anmeldung und damit Authentisierung abfragen können.

LDAP-Verzeichnis-Server können einfach repliziert werden, womit eine höhere Skalierbarkeit erreicht und der Schaden bei einem Ausfall gemindert wird.

### 5.4.3 Statische Daten

In beiden Szenarien gehen wir davon aus, dass alle statischen Daten wie Geometrie und Textur der graphischen Objekte, Sound usw. bereits auf dem Teilnehmer vorliegen oder als Java-Archiv über einen HTTP-Server bezogen werden können. Sind diese Daten sicherheitsrelevant, kann die Übertragung mit TLS unter Verwendung der Teilnehmer-Zertifikate geschützt werden.

### 5.4.4 Fantasy-Rollenspiel

In diesem Abschnitt wird die Sicherheitsanalyse des in Abschnitt 5.2.1 beschriebenen Fantasy-Rollenspiel-Szenarios durchgeführt. Die essenziellen Sicherheitsanforderungen werden dabei in der Grundarchitektur verankert.

#### Sicherheitsrichtlinie

Die beiden Hauptrollen im Computerspiel sind der *Betreiber*, der das Spiel als Dienst anbietet, und der *Spieler*, der diesen Dienst in Anspruch nimmt.

Die Sicherheitsanalyse bringt für das Computerspiel folgende wesentliche Konsequenzen:

Tabelle 5.2: Sicherheitsrichtlinie für Spieler und Betreiber im Computerspiel-Szenario

| *Schutzbereich* | *Spieler* | *Betreiber* |
|---|---|---|
| *persönliche Daten* | versichert Richtigkeit | garantiert Datenschutz |
| *Gebühr* | versichert Zahlung | garantiert korrekte Berechnung<br>garantiert Verfügbarkeit für Zahlungszeitraum |
| *Zugangsdaten* | garantiert Geheimhaltung | garantiert Geheimhaltung<br>versichert korrekte Prüfung bei Teilnahme |
| *Integrität der Software* | keine Modifikationen | garantiert Funktionalität<br>keine Gefährdung des Teilnehmersystems<br>Patches für entdeckte Fehler |
| *Integrität des Netzwerkprotokolls* | Prüfung aller Nachrichten<br>kein Versand illegaler Nachrichten | Prüfung aller Nachrichten<br>Detektion von Betrugsversuchen<br>Ausschluss von *cheaters* |
| *Integrität der Szene* | keine illegalen Modifikationen<br>keine ungültigen Ereignisse | Verwaltung des maßgeblichen Zustands in fairer Weise<br>zuverlässige persistente Speicherung der Szene und der Avatare |
| *Spielregeln* | versichert Einhaltung | formuliert Spielregeln<br>garantiert Durchsetzung<br>Detektion von Verstößen<br>Ausschluss von *cheatern* |

- Alle maßgeblichen Entscheidungen müssen von Rechnern des Betreibers in fairer Weise getroffen werden.
- Spieler sind potenziell unzuverlässig.
- Spieler, die gegen die Regeln verstoßen, die Spielsoftware modifizieren oder die Rechner anderer Spieler angreifen, müssen vom Betreiber kurzzeitig oder dauerhaft ausgeschlossen werden.
- Das Verbindungsnetzwerk ist öffentlich und daher unzuverlässig.
- Einzelne Spieler dürfen nur so viele Informationen erhalten, wie sie zur aktuellen Darstellung der Szene und zum Spielen des Spiels benötigen.
- Für den Betreiber ist die Durchsetzung der Teilnahmegebühr entscheidend.

Die Sicherheitsrichtlinie des Spieles muss die Pflichten der Spieler und des Betreibers enthalten. Diese sind einander in Tabelle 5.2 gegenübergestellt.

Alle beteiligten Rechner des Betreibers sowie die dort ausgeführte Software sind zuverlässig und setzen die Sicherheitsrichtlinie durch. Sie bilden die *trusted computing base*. Die Sicherheitsrichtlinie muss sowohl vom Betreiber wie auch von den Spielen akzeptiert werden. Dazu kann sie z.B. in den Lizenzvertrag aufgenommen werden, dem jeder Benutzer bei der Installation zustimmen muss.

Bei Verstößen gegen die Sicherheitsrichtlinie droht Spielern der dauerhafte Ausschluß. Der Betreiber könnte beispielsweise zu einer Schadensersatzzahlung verpflichtet werden.

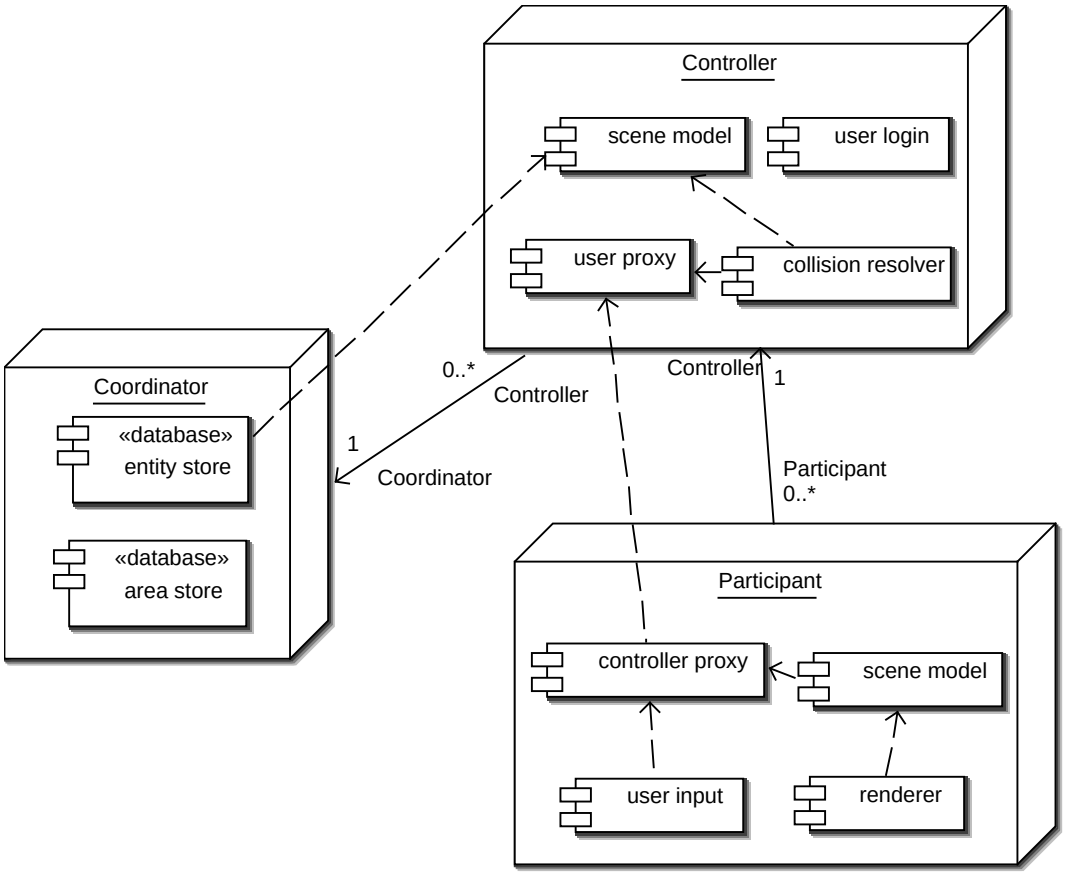



Abbildung 5.3: Aufgabenverteilung im Spielszenario

### Grundarchitektur

Ein auf Wettkampf basierendes Computerspiel erfordert ein hohes Maß an Konsistenz der Szene. Da jeder Teilnehmer selbst ein potenzieller Angreifer ist, müssen zentrale Entscheidungsinstanzen eingeführt werden, die den maßgeblichen Zustand verwalten. Es kommen also nur Client-Server-, Distributed-Server- oder Client-Proxy-Server-Architekturen infrage. Die Entscheidung in dieser Arbeit fiel auf die Client-Proxy-Server-Architektur, denn eine reine Client-Server-Architektur skaliert zu schlecht, während die Distributed-Server-Architektur schwer zu koordinieren ist.

Abbildung 5.3 skizziert die Aufgabenverteilung. Der Server unserer Architektur heißt *Koordinator* und koordiniert die Arbeit der Proxies, hier *Kontrolleure* genannt, da sie die Entscheidungsinstanzen für einen Bereich der Szene sind. Der Koordinator weist den Kontrolleuren disjunkte Bereiche der Szene zu, sodass keine zwei Koordinatoren gleichzeitig denselben Ausschnitt der Szene verwalten, und damit alle zeitkritischen Entscheidungen lokal und ohne zusätzliche Kommunikation mit anderen Kontrolleuren getroffen werden können. Der Koordinator ist außerdem für die persistente Speicherung der dynamischen Entitäten sowie das Sammeln der Zugriffsdaten der Spieler verantwortlich.

Die Anmeldung eines neuen Kontrolleurs beim Koordinator erfolgt recht selten und erfordert maximale Sicherheit. Deshalb wird hier das SIGMA-I-Protokoll aus Abbildung 4.4 verwendet.

Der Koordinator stellt als zentrale Komponente einen kritischen Ausfallpunkt dar. Allerdings tritt er nur bei der Anmeldung neuer Kontrolleure, zum Einsammeln der Zugriffsdaten und zur periodischen Speicherung der Szene in Aktion. Der erste Fall tritt sehr selten auf: Beim Start des

Spiels, bei der Erweiterung des Kontrolleur-Netzwerks oder bei Wiederanmeldung eines zuvor ausgefallenen Kontrolleurs. Ebenso werden die Zugriffsdaten eher selten abgefragt. Sie können, wie der Szenenzustand auch, von den Kontrolleuren lokal zwischengespeichert werden und erst bei der Wiederherstellung nach einem Ausfall des Koordinators übertragen werden. Die persistente Speicherung des Szenenzustands erfolgt in regelmäßigen, relativ großen Zeitabständen, z.B. in Intervallen von zehn Minuten. Somit funktioniert die Laufzeitumgebung die meiste Zeit über unabhängig vom Koordinator.

Die Kontrolleure bilden untereinander einen geschützten Kommunikationsverbund, über den sicherheitsrelevante Daten regelmäßig ausgetauscht werden. Der Kommunikationskanal des Verbunds muss vertraulich, verlässlich und integritätsgeschützt sein. Bei einer geringen Anzahl von Kontrolleuren reicht für die Gruppenkommunikation emuliertes Multicasting auf der Basis von Java-NIO-TCP/IP-Sockets (vgl. Abschnitt 3.4.5) aus. Ein neuer Kontrolleur muss sich dann individuell bei allen vorhandenen Kontrolleuren anmelden. Steigt die Anzahl der Kontrolleure, lohnt sich der Umstieg auf *application-layer multicasting* (vgl. Abschnitt 3.3.5). Der Koordinator kann dabei die Gruppenverwaltung übernehmen.

Teilnehmer kommunizieren nie direkt mit dem Koordinator, sondern immer nur mit ihrem aktuellen Kontrolleur. Der Koordinator bleibt damit für sie unsichtbar und somit auch schwieriger angreifbar. Um die Verfügbarkeit und die Echtzeitreaktion der Entscheidungsinstanzen zu gewährleisten, muss die Teilnehmerzahl pro Kontrolleur a priori beschränkt werden.

Die Teilnahme am Spiel setzt das grundsätzliche Vertrauen in die Kontrolleure und den Koordinator voraus. Die Rechner teilnehmender Spieler kommen also nicht für diese Rollen infrage. Teilnehmer, die den Verwaltungsbereich eines Kontrolleurs verlassen, werden von diesem an den Kontrolleur des neuen Bereichs delegiert.

Da die Teilnehmer nicht mehr selbst entscheiden dürfen, ob Ereignisse gültig sind, senden sie *Aktionsanfragen* an den Kontrolleur und erhalten erst nach Billigung dieser Anfragen die tatsächlich zu verarbeitenden Ereignisse. Ein Kompromiss zwischen Latenz und Fairness wird durch die Einführung einer festen *Rundenzeit* $T_{round}$ geschlossen, mit der der Kontrolleur Synchronisationspunkte für alle von ihm überwachten Spieler setzt. Innerhalb einer Runde kann jeder dieser Spieler für jede seiner Entitäten genau eine Aktionsanfrage stellen. Jede weitere Anfrage wird vom Kontrolleur ignoriert. Alle innerhalb einer Runde beim Kontrolleur eingegangenen Aktionsanfragen werden in der nächsten Runde verarbeitet, als wären sie gleichzeitig eingegangen. Die dadurch entstehende Ordnung der Ereignisse macht Zeitstempel und damit auch die Synchronisation der Teilnehmeruhren überflüssig. Der Kontrolleur prüft alle Anfragen auf Gültigkeit und Kollisionen und gleicht diese falls nötig mit anderen Kontrolleuren ab. Die dafür maximal benötigte Zeit nennen wir $T_{control}$. Ist $n$ die Anzahl der Teilnehmer eines Kontrolleurs, so erwartet man als Aufwandsschätzung

$$T_{control} = \mathcal{O}(n^2), \tag{5.1}$$

da prinzipiell alle Ereignisse miteinander auf Kollisionen überprüft werden müssen. Anschließend sendet der Kontrolleur die daraus resultierenden bereinigten Ereignisse an seine Teilnehmer. Dazu benötigt er die Verarbeitungszeit $T_{compose}$. Abbildung 5.4 zeigt, wie die Gesamtlatenz jeder Aktion durch die zentralisierte Architektur ansteigt(vgl. Abschnitt 3.1.1).

Für die Antwortzeit gilt

$$T_{response} \leq \max(T_{latency}, T_{round}) + T_{control} + T_{compose} + T_{latency}. \tag{5.2}$$

Um die bestmögliche Interaktivität zu erreichen, sollte $T_{round}$ also unter der maximal zu erwartenden Netzwerklatenz liegen. Eine untere Schranke ist die Summe aus $T_{control}$ und der Update-

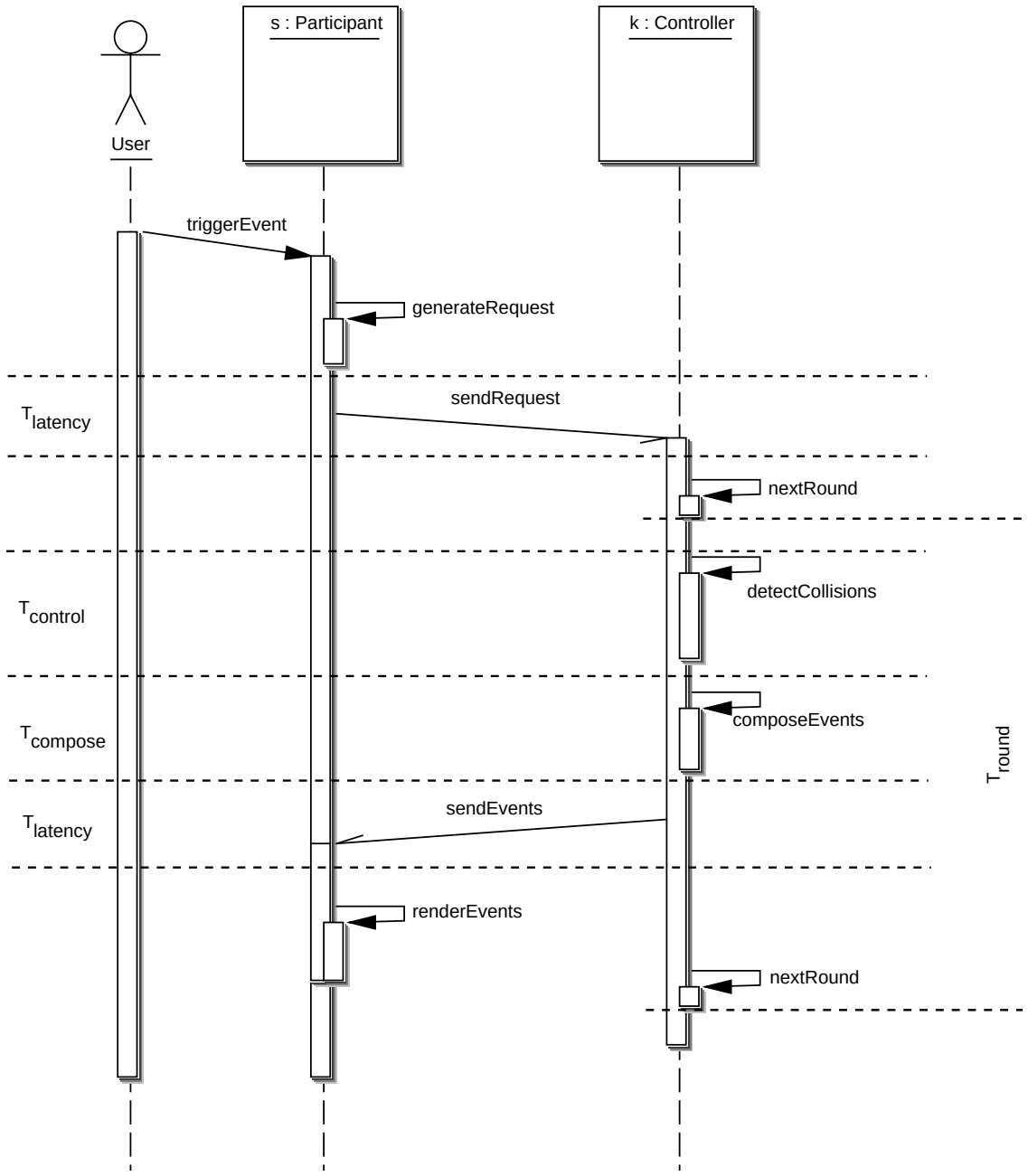



Abbildung 5.4: Latenz im Spielszenario

Sendezeit $T_{compose}$, da sonst nicht alle Ereignisse innerhalb der nächsten Runde verarbeitet werden können. Damit ergibt sich

$$T_{control} + T_{send} \leq T_{round} \leq T_{latency}, \tag{5.3}$$

also eine Kopplung von Skalierbarkeit – die linke Seite wächst mit der Anzahl der Teilnehmer – und Interaktivität. Das war zu erwarten, da wir mit der zentralisierten Topologie den Schwerpunkt auf Verlässlichkeit gesetzt haben (vgl. Abschnitt 2.3.3).

### Teilnehmer-Registrierung

Um die Spielzeit einmal abrechnen zu können, benötigt der Betreiber persönliche Daten über den Benutzer in der realen Welt, wie Name, Kreditkartennummer usf. Diese Daten muss jeder Spieler bei der *Teilnehmer-Registrierung* angeben. Um den Datenschutz zu gewährleisten, ist es angeraten, die Wurzel-CA, wie in Abschnitt 5.4.1 vorgeschlagen, offline zu betreiben und die Benutzerdaten ebenfalls vor den Rechnern der Laufzeitumgebung abzuschirmen.

### Zugangskontrolle

Die Laufzeitumgebung benötigt lediglich das Wurzel-Zertifikat. Bei der Anmeldung authentisiert sich ein Spieler mithilfe seiner Zertifikatskette und seines privaten Schlüssels bei einem Kontrolleur. Zur Anmeldung benutzen wir die MIKEY-Schlüsselverhandlung mit digitalen Signaturen (vgl. Abbildung 4.3), verzichten dadurch auf einen expliziten Lebendigkeitsbeweis des Teilnehmers sowie auf den Identitätsschutz zu Gunsten eines schlanken Protokolls mit nur zwei Nachrichten, das keine Zustandsverwaltung benötigt und damit auch resistenter gegen DoS-Angriffe auf die Anmeldung ist. Von dem bei der Verhandlung etablierten *Sitzungsschlüssel* können dann weitere Schlüssel zur Authentisierung und zur Verschlüsselung des kommenden Nachrichtenverkehrs abgeleitet werden, wodurch eine Sicherheitsassoziation zwischen dem Kontrolleur und dem Spieler entsteht.

Um den Kontrolleur nicht mit der Überprüfung und der Generierung der digitalen Signaturen zu überlasten, läuft die Anmeldung in einem eigenen Thread mit niedriger Priorität, dessen Ausführung nach jeder geglückten Anmeldung für eine feste Zeit ausgesetzt wird, in der sich der Kontrolleur anderen Aufgaben widmen kann.

Die Kontrolleure unterrichten einander in regelmäßigen Abständen über die angemeldeten Spieler, Zertifikatswiderrufe oder den temporären Ausschluss von Spielern. Dadurch kann vermieden werden, dass sich Benutzer mehrfach oder illegitim anmelden.

Die Zustände der Avatare werden sowohl bei der periodischen Sicherung des Gesamtzustands der Szene als auch am Ende einer Sitzung von den Kontrolleuren an den Koordinator weitergegeben, der sie in der Datenbank ablegt, aus der sie beim nächsten Login wieder geladen werden können. Manipulationen werden so ausgeschlossen.

Die Trennung zwischen Laufzeit- und Registrierungsumgebung kann sich auch bei der Detektion von Angriffen auf die Kontrolleure als vorteilhaft erweisen, da die Zugriffsdaten zum Abrechnungszeitpunkt unter weniger Zeitdruck kontrolliert werden können. Fallen Angriffe bestimmter Teilnehmer auf, werden deren Zertifikate vom Registrierungs-Server widerrufen.

### Update-Nachrichten

In Abschnitt 4.6.4 wurde bereits gezeigt, dass Multicasting für die Übertragung der Update-Nachrichten und der Aktionsanfragen nicht infrage kommt. Die Kontrolleure müssen die Nachrichten also einzeln an die Teilnehmer versenden.

Zum Schutz vor unberechtigten Zugriffen auf Entitäten müssen alle Aktionsanfragen an die Kontrolleure von den Teilnehmern authentisiert werden. Da sich die Authentisierungszeit als Summand auf die Gesamtverarbeitungszeit auswirkt, muss hier ein möglichst schneller Algorithmus eingesetzt werden. Wir benutzen daher einen MAC mit einem vom Sitzungsschlüssel abgeleiteten *Authentisierungsschlüssel*. Damit niemand maßgebliche Update-Nachrichten fälschen kann, müssen diese ebenfalls vom Kontrolleur mit einem MAC und dem Authentisierungsschlüssel authentisiert

werden.

Um weiterhin illegale Objektaufdeckung (vgl. Abschnitt 5.3.2) unmöglich zu machen, dürfen Teilnehmer nur Ereignisse empfangen, die sich in ihrem Sichtbereich abspielen. Diese muss der Kontrolleur für jeden seiner Teilnehmer einzeln bestimmen und per Unicast senden. Die Auswahl der für die einzelnen Teilnehmer sichtbaren Ereignisse verlängert die Verarbeitungszeit $T_{control}$. Aus Gleichung 5.3 folgt dann, dass der Schutz vor illegaler Objektaufdeckung die Skalierbarkeit des Systems herabsetzt. Die Sichtbereichsberechnung kann sich allerdings lohnen, wenn dadurch die Anzahl der zu sendenden Ereignisse und damit die Sendezeit $T_{send}$ reduziert wird. Ergebnisse hierzu werden im Abschnitt 5.6.1 präsentiert.

Durch Abhören der Kommunikation anderer Teilnehmer kann ein Spieler an für ihn nicht sichtbare Ereignisse geraten, z.B. liefern die Anfragen anderer Teilnehmer wertvolle Zusatzinformationen über deren zukünftige Aktionen. Als Gegenmaßnahme können die Nachrichtenkanäle individuell verschlüsselt werden. Die dazu notwendigen *Chiffrierschlüssel* werden aus dem Sitzungsschlüssel erzeugt. Die notwendige Dauer der Geheimhaltung ist vergleichsweise kurz, wenn nur Reflexverstärkungsangriffe erschwert werden sollen. In diesem Fall können Chiffrieralgorithmen mit kurzen Schlüsseln verwendet werden.

Im Internet ist das Abhören relativ schwierig, sobald man *man-in-the-middle-Angriffe* ausgeschlossen hat. In bestimmten LANs lassen sich allerdings verhältnismäßig einfach so genannte „Paketschnüffler“ (englisch *packet sniffer*) einsetzen. Um die abgehörten Nachrichten verwerten zu können, ist eine zusätzliche Modifikation des Clients notwendig. Das benötigte Expertenwissen macht diesen Angriff eher unwahrscheinlich, während die Leistungseinbußen deutlich sind. Verschlüsselung im Spielszenario wurde daher als eine separate Option eingestuft, deren Kosten in Abschnitt 5.6.1 genauer untersucht werden.

#### Abrechnung

Die persönlichen Benutzerdaten werden erst wieder relevant, wenn eine Abrechnung stattfindet. Dazu benötigt der *Abrechnungs-Server* die Zugriffsdaten der Laufzeitumgebung und die Daten des realen Spielers aus dessen Registrierung, um Rechnungen zu erstellen bzw. die Abbuchungen vorzunehmen. Die Zugriffsdaten werden von den Kontrolleuren signiert, um unerlaubte Manipulationen zu verhindern.

Die Kontrolleure führen über Anmeldungen und Abmeldungen der Spieler Buch. Da ein Teilnehmer durch äußere Einwirkung vom Spiel getrennt werden könnte, werden Spieler, die über einen längeren Zeitraum keine Ereignisse senden, vom Kontrolleur automatisch abgemeldet. Der Kontrolleur muss die *Zugriffsdaten* persistent speichern, damit sie bei einem Systemausfall nicht verloren gehen.

Die Abrechnung findet nicht sehr häufig statt, etwa einmal im Monat. Die Laufzeitumgebung kommuniziert also insgesamt sehr selten mit dem Abrechnungs-Server. Für optimale Sicherheit können die Komponenten der Registrierungs-Server, die persönliche Daten der Spieler und den privaten Schlüssel des Zugangs-Zertifikats speichern, vor dem restlichen Netzwerk abgeschirmt werden. Die Kommunikation mit diesen Komponenten kann sogar manuell erfolgen, z.B. über einen mobilen Datenträger, um Netzwerkangriffe auf die sicherheitsrelevanten Daten auszuschließen.

### 5.4.5 Innenarchitektur-Simulation

Viele Aspekte der in Abschnitt 5.2.2 vorgestellten Innenarchitektur-Simulation sprechen dafür, dieses DVE als Peer-to-Peer-System zu entwerfen:

- Da die Teilnehmer nicht miteinander in Konkurrenz stehen, hat das System relativ geringe Konsistenzanforderungen.
- Die Veränderungen der Szene sind nicht so zeitkritisch.
- Alle autorisierten Teilnehmer sind vertrauenswürdig.

Außerdem ist die erwartete Anzahl von Teilnehmern wesentlich geringer als im Fantasy-Rollenspiel-Szenario. Als Richtwert nehmen wir etwa 20 Teilnehmer an.

Da reine Peer-to-Peer-Systeme mit verlässlicher Kommunikation und Gruppenkontrolle sehr aufwändig zu programmieren sind, wurde auch hier ein etwas zentraler organisierter Kompromiss geschlossen.

### Raumverwaltung

Das Ziel der Simulation ist, gleichzeitig mehrere Räume einzurichten. Beginnend mit den leeren Räumen fügen die Benutzer immer mehr Einrichtungsgegenstände und Accessoires hinzu, bis der Gesamtentwurf abgeschlossen ist. Ein wesentlicher Bestandteil muss also die *persistente Speicherung* der Räume und deren aktueller Zustände sein.

Die disjunkten Räume liefern eine natürliche Gebietszerlegung der Szene (vgl. Abschnitt 2.4.2), die eine skalierbare Lastverteilung ermöglicht. Um die Konsistenz zu gewährleisten, sollte zu einem festen Zeitpunkt genau ein Teilnehmer den maßgeblichen Zustand eines Raumes verwalten. Wir nennen ihn im Folgenden den *Verwalter* des Raums. Alle Änderungen der Szene müssen mit dem Verwalter abgestimmt werden. Ein *Koordinator* koordiniert, wie im Rollenspiel-Anwendungsfall, die Verteilung der Räume an die Verwalter und übernimmt die zentrale Speicherung der Szenendaten sowie die Vergabe eindeutiger Bezeichner für jeden Raum.

Durch diese beiden zentralen Rollen handelt es sich nicht mehr um ein reines Peer-to-Peer-System. Allerdings kann jeder Teilnehmer die Rolle des Verwalters übernehmen. In dieser Hinsicht sind alle Teilnehmer also gleichrangig.

In einem Innenarchitekturbüro werden die Arbeitsplatzrechner typischerweise für verschiedene Tätigkeiten genutzt. Man kann nicht erwarten, dass sämtliche Rechner ständig ausschließlich für die Simulation zur Verfügung stehen. Daher muss die Verwaltung eines Raums *delegierbar* sein. Die Rolle des Verwalters kann auch in regelmäßigen Zeitabständen reihum an alle Teilnehmer des Raums delegiert werden, um die damit verbundene Rechen- und Kommunikationslast zu verteilen. Fällt ein Verwalter aus, können die verbliebenen Teilnehmer die Verwaltung auf diese Weise schnell neu regeln. Die Delegierbarkeit der Verwaltung beseitigt auch viele der typischen Probleme einer zentralen Architektur.

Wie bei allen Entwicklungsprozessen kommt es auch hier manchmal vor, dass Arbeitsschritte zurückgenommen werden müssen, z.B. wenn sich eine Idee als unvorteilhaft erweist. Deswegen sollte die gesamte Entwicklungsgeschichte des Raums gespeichert werden, sodass man später auch auf frühere Entwicklungsstadien zurückgreifen kann.

Im Fall, dass der Koordinator ausfällt, können die Zustandsdaten vom Verwalter auch lokal zwischengespeichert werden, dann allerdings in angemessen sicherer verschlüsselter Form. Sobald der Koordinator wieder erreichbar ist, können alle in der Ausfallzeit vorgenommenen Veränderungen eingespielt werden. Die Funktion des restlichen Systems ist zunächst also nicht gefährdet.

Die Rolle des Koordinators zu delegieren ist wenig sinnvoll, da er die große Raum-Datenbank verwaltet, deren Replikation zu aufwändig wäre. Sämtliche Kommunikation mit dem Koordinator

muss authentisiert und verschlüsselt werden, da die Konsistenz und die Geheimhaltung der Szenendaten höchste Priorität haben.

### Teilnehmeranmeldung

Auch hier werden zur Teilnehmer-Anmeldung Zertifikate verwendet, die ausgehend von einem gemeinsamen Wurzel-Zertifikat von einer zentralen Zertifizierungsinstanz, z.B. einem Systemadministrator des Unternehmens, ausgestellt werden. Online-Zertifizierung ist nicht erforderlich.

Da alle Teilnehmer das gemeinsame Wurzel-Zertifikat kennen, kann jeder Teilnehmer die anderen authentifizieren, wie es sich für ein Peer-to-Peer-System gehört. Zertifikate und Widerrufslisten können in replizierbaren Verzeichnissen abgelegt werden.

Die typische Benutzung des Systems verläuft wie folgt: Ein Teilnehmer meldet sich beim Koordinator für einen bestimmten Raum an. Diese Anmeldung erfolgt mit dem SIGMA-I-Protokoll, denn aufgrund der niedrigen Teilnehmerzahl können wir uns hier mehr Sicherheit auf Kosten einer längeren Verhandlung leisten. Nach geglückter Anmeldung lädt der Teilnehmer den zuletzt gespeicherten Zustand des Raums. Wird der Raum bereits von einem anderen Teilnehmer verwaltet, teilt der Koordinator dem Neueinsteiger die Daten zur Kontaktaufnahme mit diesem Verwalter mit. Ansonsten wird der neue Teilnehmer als der neue Verwalter des Raums vermerkt.

Meldet sich ein neuer Teilnehmer danach bei dem Verwalter eines Raums an, teilt dieser ihm alle Veränderungen seit der zuletzt vom Koordinator gespeicherten Version mit. Danach kann der Teilnehmer an der vom Verwalter koordinierten Gestaltung des Raums mitwirken.

Ein anderer Fall ist die Registrierung eines neuen Raums. Ein Teilnehmer hat ein Rohmodell eines neuen Raumes entworfen. Er meldet diesen neuen Raum beim Koordinator an. Dieser ordnet dem Raum einen neuen Bezeichner zu, speichert den Ausgangszustand des Raums ab, und ernennt den Teilnehmer, der den Raum registriert hat, auf Wunsch zu dessen aktuellem Verwalter.

### Kommunikation innerhalb eines Raums

Der gesamte Nachrichtenverkehr innerhalb eines Raums wird verschlüsselt und mit MACs authentisiert. Innerhalb eines Raums werden zwei Arten von Nachrichten versandt: Zustands-Updates der Avatare und Änderungen des Raums.

Der Avatar eines Benutzers legt in diesem System hauptsächlich die Kameraposition fest. Kollisionen mit anderen Avataren oder Entitäten der Szene sind zwar für den Benutzer unschön, aber für die Hauptfunktion des Systems, Räume einzurichten, nicht weiter relevant. Daher verzichten wir bei den Zustands-Updates von Avataren zugunsten besserer Interaktivität auf eine zentrale Kollisionserkennung durch den Verwalter, und damit auf Konsistenz. Gegenseitige Durchdringung von Avataren oder Kollisionen mit anderen Entitäten der Szene können jeweils lokal aufgelöst werden. Bei besonders leistungsschwachen Teilnehmern kann die Kollisionsdetektion sogar ganz abgeschaltet werden, um die Rechenzeit für wichtigere Berechnungen zu verwenden. Jeder Benutzer kann den Zustand seines Avatars selbst direkt manipulieren und die zugehörigen Update-Nachrichten über einen Multi-Session-Kanal (vgl. Abschnitt 3.4.5) an alle anderen Teilnehmer des Raums versenden. Wenn verfügbar, sollte der Kanal IP-Multicasting basiert sein. Ansonsten genügt auch ein Unicasting-basierter Kanal; denn angesichts der wenigen Teilnehmer ist *application-layer multicasting* überdimensioniert. Falls individuelle Nachrichtenauthentisierung notwendig ist, kann auch das TESLA-Protokoll A.6.3 eingesetzt werden, da Avatar-Update-Nachrichten wegen der abgeschwächten Konsistenz nicht mehr so zeitkritisch sind. In diesem Fall müssen alle Teilnehmeruhren zusätzlich mit dem NTP-Protokoll synchronisiert werden. Eine Sichtbereichsberechnung ist bei so

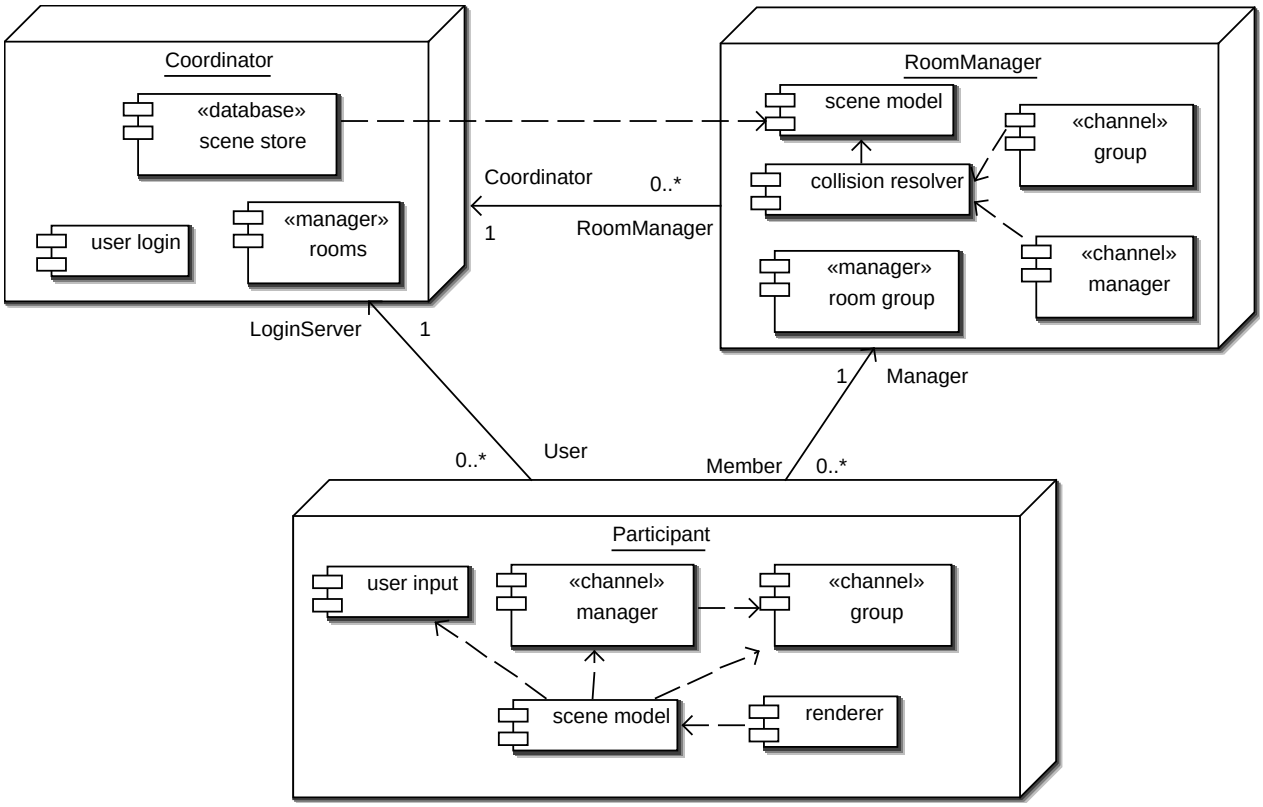



Abbildung 5.5: Aufgabenverteilung im Innenarchitektur-Szenario

wenig Teilnehmern ebenfalls unnötig.

Bei Änderungen des Raums müssen Kollisionen allerdings sehr wohl detektiert und aufgelöst werden, da man sich hier keine Inkonsistenzen leisten kann. Der Verwalter löst eventuelle Konflikte auf und sendet die maßgeblichen Änderungen an alle. Dies entspricht dem Einer-an-Viele-Kommunikationsmuster aus Abschnitt 3.4.5. Auf Verwalterseite wird also ein Multi-Session-Kanal eingesetzt und auf Teilnehmerseite der dazu passende Single-Session-Kanal. Der Verwalter kann so über denselben Kanal alle Einzelanfragen empfangen und alle maßgeblichen Updates per Multicast senden.

Die Teilnehmer eines Raums bilden eine Gruppe, die einen Gruppenschlüssel teilt, von dem der Gruppenauthentisierungsschlüssel und der Gruppenchiffrierschlüssel abgeleitet werden. Neue Teilnehmer melden sich mit einem Schlüsselverhandlungsprotokoll mit digitalen Signaturen (vgl. Abschnitt 4.6.2) an. Für das Gruppenschlüsselmanagement kann beispielsweise *Kronos* (vgl. Abschnitt 4.6.4) eingesetzt werden, wobei die Rolle des globalen Gruppenverwalters dem Koordinator zufällt, während die Raumverwalter die lokale Gruppenverwaltung übernehmen.

Die gesamte Aufgabenverteilung ist in Abbildung 5.5 dargestellt. Der Koordinator verwaltet die Raum-Datenbank und eine Liste der aktuellen Verwalter der einzelnen Räume. Des Weiteren ist er die erste Anlaufstelle für ein Benutzer-Login. Ein Raumverwalter besitzt das maßgebliche Modell des Raums und löst alle Kollisionen innerhalb des Raums auf. Er verwaltet außerdem die aktuelle Benutzergruppe des Raums und sendet den Raumzustand in regelmäßigen Abständen zur Speicherung an den Koordinator. Die Teilnehmer einer Raumgruppe kommunizieren alle nicht kollisionsrelevanten Nachrichten über den Gruppenkanal und die übrigen Nachrichten über den Verwalter-Kanal, verarbeiten Benutzereingaben und stellen die Szene dar.

## 5.5 Entwurf und Implementierung

Basierend auf der Analyse und der Grundarchitektur des vorangehenden Abschnitts werden im Folgenden der Entwurf und die Implementierung der wesentlichen Sicherheitskomponenten für DVEs vorgestellt.

### 5.5.1 Revisionsaufzeichnung

Die im Folgenden beschriebenen Softwarekomponenten protokollieren alle sicherheitsrelevanten Vorgänge für eine spätere Analyse. Dazu wurde ausgiebig von den Klassen des java.util.logging-Pakets Gebrauch gemacht, mit dem die Granularität der Log-Nachrichten gesteuert werden kann und in dem die Formatierung und Ausgabe der Log-Daten abstrahiert wird. Dadurch können die Ausführlichkeit, die Formatierung und das Ausgabemedium einfach und ohne Modifikation der protokollierenden Klassen an die jeweilige Anwendung angepasst werden.

Als Name des Logger-Objekts, das eine Komponente benutzt, wurde immer der Name des Pakets gewählt, in dem die Komponente enthalten ist. Dadurch benutzen unterschiedliche Komponenten unterschiedliche Log-Kanäle.

Die Log-Nachrichten der implementierten Komponenten sind, wenn die feinste Log-Granularität gewählt wird, sehr ausführlich, da sie auch bei der Fehlersuche verwendet wurden. Daher wird in den nachfolgenden Abschnitten nicht jeder Log-Eintrag explizit erwähnt.

### 5.5.2 Architektur der Teilnehmerregistrierung

Wie in der Analyse beschrieben, soll die Zugangskontrolle mithilfe von Zertifikaten erfolgen, die die Teilnehmer bei der Registrierung erhalten. In diesem Abschnitt stelle ich die Architektur der Teilnehmerregistrierung dar. Diese ist in zwei Pakete geteilt: Das Paket dvesec.cert enthält die grundlegenden Klassen für Zertifikate, Widerrufslisten und die Erzeugung der beiden, während dvesec.cert.pki die Klassen für die Public-Key-Infrastruktur (PKI) umfasst.

### 5.5.3 Eindeutige Bezeichner

Jeder Benutzer im System und jede kommunizierende Entität wird durch einen *eindeutigen Bezeichner* (ID) gekennzeichnet, der vom Registrierungs-Server bei der Registrierung zugewiesen wird. Die Menge der möglichen Bezeichner wird in disjunkte Teilmengen auf die Registrierungs-Server verteilt, um Kollisionen auszuschließen. Der Bezeichner wird durch ein Objekt der Klasse dvesec.cert.DVEPrincipal repräsentiert. Benutzer können einen Namen frei wählen. Der Name dient nur der einfacheren menschlichen Kommunikation und stellt wie im realen Leben keinerlei Eindeutigkeit sicher.

### 5.5.4 Zertifikattypen

Im Rahmen der Registrierung treten vier Rollen auf: Der *Teilnehmer* stellt eine Zertifizierungs-Anfrage, die von einem *Registrierungs-Server* geprüft wird. Ist die Anfrage gültig, erzeugt der Registrierungs-Server ein neues Zertifikat und sendet es an den Teilnehmer sowie an den *LDAP-Verzeichnisdienst*, über den es öffentlich abgefragt werden kann. Der *Betreiber* ermächtigt schließlich die Registrierungs-Server zur Erstellung der Teilnehmer-Zertifikate.

Tabelle 5.6: Operationen der verschiedenen Rollen

| *Rolle* | *Operationen* | *Zertifikats-Klassenname* |
|---|---|---|
| Teilnehmer | Client-Authentisierung<br>Verschlüsselung von Daten und Schlüsseln<br>nicht abstreitbare digitale Signatur<br>Aufbau von TLS-Verbindungen | UserCertificate |
| LDAP-Server | Aufbau von TLS-Verbindungen | ServerCertificate |
| Registrierungs-Server | Signatur von Server-Zertifikaten<br>Signatur von Teilnehmer-Zertifikaten<br>Signatur von Widerrufslisten | CACertificate |
| Betreiber | Signatur von Registrierungs-Zertifikaten<br>Signatur von Widerrufslisten | RootCertificate |

Um die einzelnen Rollen auch im Programm unterscheiden zu können, wurde für jede Rolle eine eigene *Zertifikats-Klasse* vorgesehen, anhand derer dann entschieden werden kann, ob eine Operation zulässig ist. Die verschiedenen Klassen und Operationen sind in Tabelle 5.6 zusammengefasst.

Da die Standardbibliotheken in Java bereits in vielen Klassen die Verwendung von X.509-Zertifikaten vorsehen, wurden diese als Basis für die Zertifizierungsarchitektur gewählt. Die gemeinsamen X.509-Attribute aller DVE-Zertifikattypen sind in Tabelle 5.7 aufgeführt. Der X.509-Standard sieht eine Reihe von Erweiterungen der Grundattribute von Zertifikaten vor, mithilfe derer die verschiedenen Zertifikattypen unterschieden werden können. Tabelle 5.8 beschreibt X.509-Erweiterungen zur Unterscheidung der DVE-Zertifikattypen.

Die Erweiterungen digitalSignature, keyEncipherment und keyAgreement werden von der *Java Secure Socket Extension* (JSSE, vgl. Abschnitt B.3) zum Aufbau von TLS-Verbindungen benötigt. nonRepudiation dient der Nicht-Abstreitbarkeit einer digitalen Signatur. Zum Ausstellen eines Zertifikats benötigt man keyCertSign und für die zugehörigen Widerrufslisten cRLSign. Zur Unterscheidung zwischen Client und Server während einer Authentisierung dienen userAuthentication und serverAuthentication. Die maximale Länge einer Zertifikatskette unterhalb eines Registrierungs-Zertifikats wird durch pathLenConstraint festgeschrieben. Trägt ein Zertifikat hier den Wert eins, können also keine weiteren Registrierungs-Zertifikate mehr in der Zertifikatskette folgen. Die

Tabelle 5.7: Verwendete X.509-Attribute

| *X.509-Attribut* | *Bedeutung* |
|---|---|
| version | Die X.509-Version. Immer Version 3. |
| serialNumber | Die Seriennummer des Zertifikats. Jedes Paar (issuer, serialNumber) darf nur ein eindeutiges Zertifikat bezeichnen. |
| signature | Der verwendete Signaturalgorithmus und die digitale Signatur. |
| issuer | Der zertifizierende Signierer des Zertifikats. |
| validity | Der Gültigkeitszeitraum des Zertifikats. |
| subject | Der zertifizierte Besitzer des privaten Schlüssels. |
| subjectPublicKeyInfo | Der zertifizierte öffentliche Schlüssel des Subjekts und dessen zugehöriger Algorithmus. |
| subjectKeyIdentifier | Bezeichner für den zertifizierten Schlüssel. |
| authorityKeyIdentifier | Bezeichner für den verwendeten Zertifizierungsschlüssel. |

Tabelle 5.8: X.509-Attribute zur Unterscheidung der DVE-Zertifikattypen

| *Erweiterung* | *Wert* | *Teilnehmer* | *LDAP* | *Registr.* | *Betreiber* |
|---|---|---|---|---|---|
| keyUsage | digitalSignature | × | × | | |
| | nonRepudiation | × | | | |
| | keyEncipherment | × | × | | |
| | dataEncipherment | × | | | |
| | keyAgreement | × | × | | |
| | keyCertSign | | | × | × |
| | cRLSign | | | × | × |
| extendedKeyUsage | clientAuthentication | × | × | | |
| | serverAuthentication | | × | | |
| basicConstraints | cA | | | × | × |
| | pathLenConstraint | -1 | -1 | $= 1$ | $\geq 2$ |
| Sonstige Merkmale | | | | | issuer=subject |

Beschränkung der Pfadlänge eines Betreiber-Zertifikats muss also größer oder gleich zwei sein. Im Gegensatz zu Registrierungs-Zertifikaten sind Betreiber-Zertifikate selbst signiert.

Durch Ausschluss *überkreuzter Zertifikate* (englisch *cross certificates*) bilden die Zertifikate für jedes Betreiber-Zertifikat einen Baum, dessen Wurzel das Betreiber-Zertifikat ist, dessen innere Knoten die Registrierungs-Zertifikate darstellen und dessen Blätter die Benutzer- und Server-Zertifikate bilden. Jedes Blatt-Zertifikat besitzt also eine eindeutige Zertifikatskette, die es mit seinem Betreiber-Zertifikat verbindet. Die zur Verifizierung notwendige Suche nach einer gültigen Zertifikatskette wird dadurch eine triviale Aufwärtsdurchquerung des Baums.

Die *Java Cryptographic Architecture* (vgl. Abschnitt B.2) definiert die Klassen für die Arbeit mit X.509-Zertifikaten. Die Klasse java.security.cert.X509Certificate beschreibt dabei die Zertifikate selbst. Unglücklicherweise ist sie abstrakt und wird in der Implementierung von Sun an die nicht dokumentierte konkrete Klasse sun.security.x509.X509CertImpl vererbt. Ebenso verhält es sich mit den Klassen CertPath für Zertifikatsketten und X509CRL für Widerrufslisten. Zudem fehlt der JCA generell die Möglichkeit, Zertifikate, Zertifikatspfade oder Widerrufslisten selbst zu generieren. Lediglich bereits vorhandene, serialisierte Zertifikate können unter Zuhilfenahme einer java.security.cert.CertificateFactory deserialisiert werden.

Grundsätzlich gibt es mehrere Möglichkeiten, die von uns benötigten Zertifikatsklassen auf der Basis der JCA zu implementieren. Die erste wäre die Spezialisierung der abstrakten Klasse X509Certificate durch Vererbung. Da viele andere Klassen allerdings intern die nicht dokumentierte – und damit nicht zur offiziellen API gehörende – X509CertImpl-Klasse verwenden, scheidet diese Möglichkeit aus. Vererbung einer nicht in der API enthaltenen Klasse zerstört die Portabilität, da diese in anderen JVMs eventuell gar nicht vorhanden ist. Die zweite Möglichkeit wäre, direkt mit X.509-Zertifikaten zu arbeiten und eine Hilfsklasse zu implementieren, die die Typermittlung der Zertifikate mittels statischer Methoden ermöglicht, beispielsweise public static boolean Tool.isUserCertificate(X509Certificate). Dabei würden die Zertifikattypen nicht direkt auf Klassen abgebildet und der Umweg über die Hilfsklasse ist ein wenig sperrig. Meine Wahl fiel schließlich auf eine Aggregation des X.509-Zertifikats in einer Klasse pro Typ. Die Vorteile sind:

- Die aggregierten Zertifikate können dort eingesetzt werden, wo X.509-Zertifikate benötigt werden, beispielsweise in der JSSE oder in LDAP-Klassen.

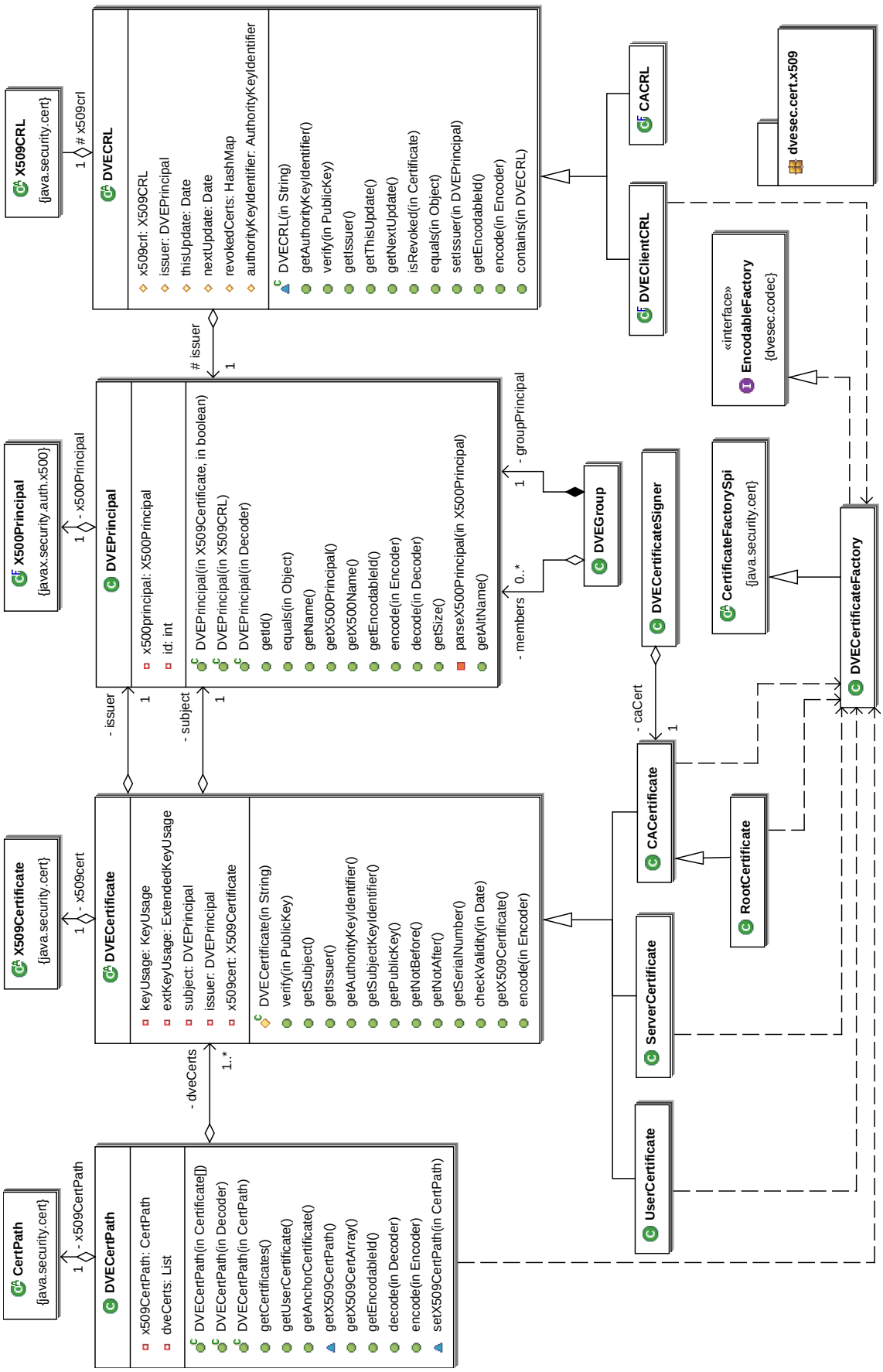



Abbildung 5.9: Klassendiagramm des dvesec.cert-Pakets

- Die Anwendung kann ansonsten mit typisierten Zertifikaten arbeiten. Wenn beispielsweise ein Registrierungs-Zertifikat an eine Methode übergeben werden soll, kann dies direkt als CACertificate markiert werden.

Als Nachteil erwies sich der etwas höhere Programmieraufwand. Abbildung 5.9 zeigt das Klassendiagramm des dvesec.cert-Pakets mit den Basisklassen der Registrierung. Die Basisklassen DVEPrincipal, DVECertificate, DVECertPath und DVECRL aggregieren jeweils ein Objekt der entsprechenden Klasse aus der JCA. Alle diese Klassen implementieren außerdem das dvesec.codec.Encodable-Interface (aus Platzgründen nicht dargestellt). Die Zertifikattypen sind als konkrete Implementierungen der abstrakten Basisklasse DVECertificate umgesetzt. Die Klasse DVECertificateFactory ist sowohl eine Zertifikatsfabrik im JCA-Sinne, als auch im Sinne der dvesec.codec-API. Mit ihr können alle Zertifikattypen deserialisiert werden. Wer ein CACertificate besitzt, kann damit einen DVE-CertificateSigner instanziieren, mit dem auf Basis des BouncyCastle-Providers (vgl. Abschnitt B.5) Zertifikate ausgestellt werden können. Das Paket dvesec.cert.x509 enthält alle direkt von BouncyCastle abhängigen Klassen, sodass andere Implementierungen zur Generierung von Zertifikaten mit wenig Aufwand erstellt werden können. Schließlich gibt es zwei konkrete CRL-Klassen: Ein CACRL-Objekt ermöglicht die Bearbeitung von Widerrufslisten, während die DVEClientCRL lediglich eine unveränderbare Widerrufsliste darstellt. Gruppen von Prinzipalen werden als DVEGroup abgebildet. Jede Gruppe hat zur Identifikation einen eigenen Prinzipal groupPrincipal und enthält mehrere Prinzipale als members.

### 5.5.5 Ankündigung und Verzeichnisdienst

Wie bereits in der Analyse beschrieben, setzen wir einen LDAP-Server zur Speicherung der Zertifikate und Serverdienstdaten ein.

Die Beschreibung der Zugriffsrechte von Prinzipalen auf die Einträge kann in LDAP auf Kontextbasis angegeben werden. So kann man beispielsweise allen Entitäten, deren Einträge im Kontext certification authorities zu finden sind, Schreibrechte auf den Kontext users geben, damit sie die generierten Zertifikate dort ablegen können. Alle Einträge in unserem Verzeichnis sind daher auf verschiedene Unterkontexte verteilt, die den Zertifikattypen entsprechen: root authorities, certification authorities, public servers, servers und users. Die Unterscheidung zwischen servers und public servers ermöglicht, dass die Daten zum Zugriff auf die Dienste der Laufzeitumgebung (servers) nur für authentisierte Benutzer zugänglich sind, und die Dienste dadurch einen Grundschutz vor DoS-Angriffen haben.

Die Klassen des Verzeichnispakets dvesec.directory sind im Diagramm 5.10 dargestellt. Jeder Verzeichniseintrag wird durch ein Objekt der Klasse DirectoryEntry repräsentiert. Jeder Eintrag besitzt mindestens die LDAP-Attribute name und description. Dabei ist name der *eindeutige Name* (englisch *Distinguished Name*) (DN) des Eintrags, während description den Typ beschreibt, der durch den Wert der Klassenkonstante DESCRIPTION gegeben ist, die jede konkrete Implementierung von DirectoryEntry definiert. Die RDN des Eintrags ist die ID des zugehörigen Prinzipals als LDAP-Attribut Common Name (CN), und wird von der Methode getRelativeDN() zurückgegeben. Der Kontext des Eintrags besteht aus seinem Unterkontext, gefolgt vom Wurzelkontext, und kann mit getContextDN() abgefragt werden. Die Menge aller LDAP-Attribute eines Eintrags erhält man mit der Methode getAttributes(). Durch die Implementierung der Encodable-Schnittstelle können Verzeichniseinträge kommuniziert und mithilfe des DirectoryCodec deserialisiert werden. Die Klassen RootContext und SubContext beschreiben den Wurzelkontext bzw. die Unterkontexte.

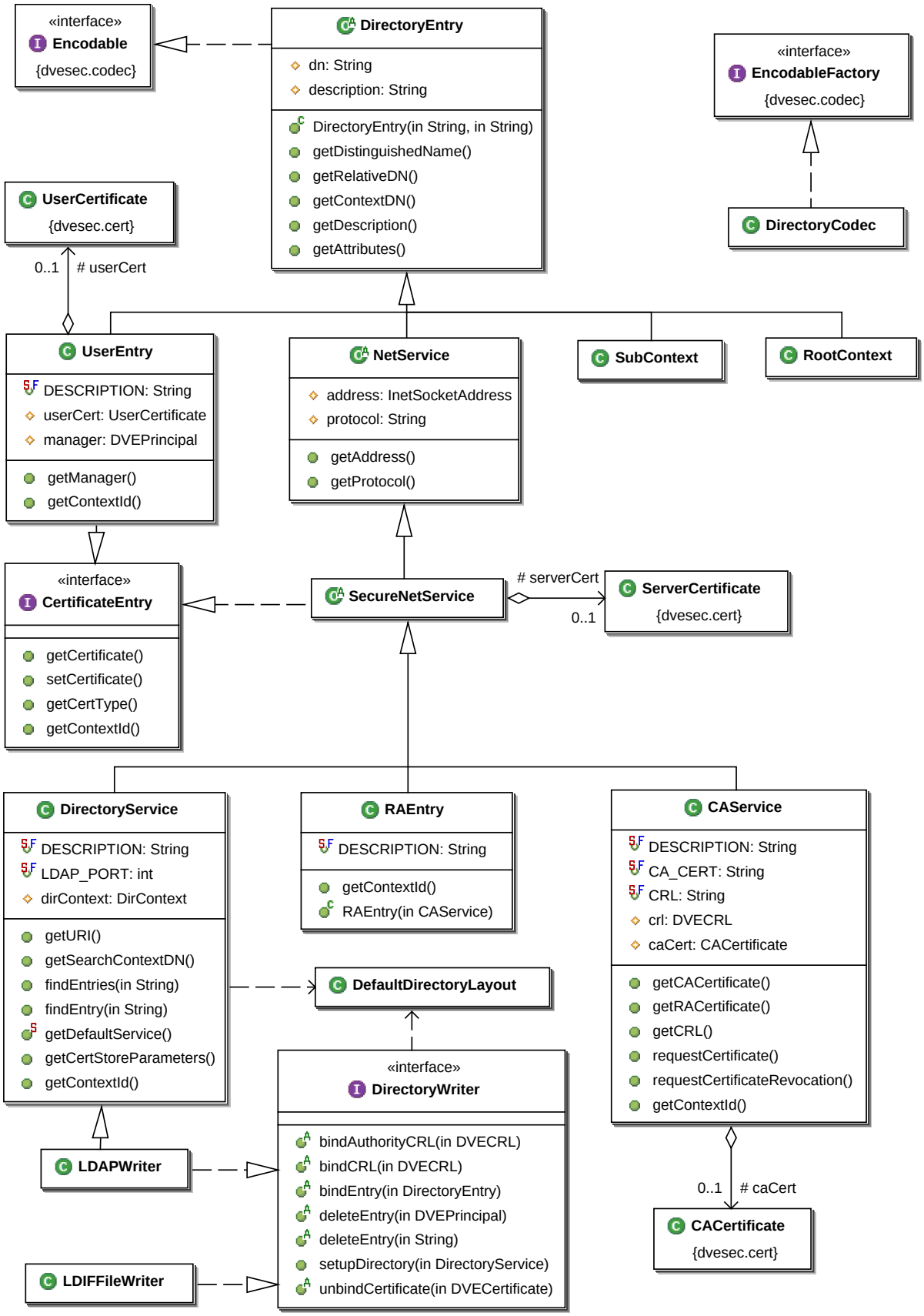



Abbildung 5.10: Klassendiagramm des dvesec.directory-Pakets

Die Klasse NetService stellt nicht nur den Verzeichniseintrag eines Netzwerkdiensts – also seine Socketadresse und sein Protokoll – dar. Alle davon abgeleiteten konkreten Klassen definieren zusätzlich auch Methoden, um diesen Dienst zu nutzen. Alle Einträge, die Zertifikate enthalten, implementieren die Schnittstelle CertificateEntry. Objekte vom Typ CertificateEntry definieren auch die Parameter eines Zertifizierungsantrags (siehe nächster Abschnitt). Teilnehmerzertifikate werden als UserCertificate im Verzeichnis eingetragen. SecureNetService erweitert NetService um ein Server-Zertifikat, sodass der Dienst über TLS angesprochen werden kann.

Ein LDAP-Verzeichnisdienst selbst wird über die Klasse DirectoryService beschrieben und kann über die Methoden findEntries genutzt werden. Als Suchparameter dient dabei entweder der Typ (String) oder eine Menge von Suchattributen (Attributes). Die Methoden findEntry liefern ein einzelnes, aus den Treffern zufällig ausgewähltes Suchergebnis. Den Uniform Resource Identifier (URI) des LDAP-Servers erhält man mit getURI, während getSearchContextDN den Namen des Wurzelkontextes des Verzeichnisses zurückgibt. Um ganz am Anfang den ersten Verzeichnisdienst zu finden, benutzt man die statische Methode DirectoryService.getService(). Ein Verzeichnisdienst kann unverschlüsselt oder über eine mit TLS geschützte Verbindung abgefragt werden.

Um Einträge im Verzeichnis abzulegen, wurde das Interface DirectoryWriter entworfen. Mit den bind-Methoden können Einträge oder Attribute im Verzeichnis abgelegt werden, während unbindCertificate ein Zertifikatsattribut löscht und die delete-Methoden ganze Einträge bzw. Subkontexte löschen. Mit setupDirectory() wird das Grundlayout, also der Wurzelkontext und die Subkontexte, generiert, das in der Klasse DefaultDirectoryLayout gespeichert ist. Da die Funktionsweise des gesamten Systems wesentlich von der Integrität des Verzeichnisses abhängt, müssen sich Prinzipale für den Schreibzugriff unbedingt authentisieren, und die ausgetauschten Nachrichten sollten verschlüsselt sein. Die meisten LDAP-Server benutzen die Authentisierungsbibliothek SASL (vgl. Abschnitt 3.3.7). SASL bietet insbesondere die für unseren Fall sehr attraktive Möglichkeit der externen Authentisierung über den TLS-Verbindungsaufbau: Nach geglücktem TLS-Handshake (vgl. Abschnitt 4.6.3) gilt der Client als Subjekt seines Zertifikats authentisiert. Der Verzeichnis-Server benötigt also keine weiteren Authentisierungsdaten als die Zertifikate, die er ohnehin speichert. Der LDAPWriter implementiert das DirectoryWriter-Interface mit externer TLS-Authentisierung. Beim *Bootstrap* tritt jetzt ein Henne-Ei-Problem auf, da die Zertifikate für die TLS-Verbindung noch nicht auf beiden Seiten vorliegen, aber für die Authentisierung des Schreibzugriffs zum Ablegen neuer Zertifikate im Verzeichnis benötigt werden. Für diesen Fall gibt es den LDIFFileWriter, der ebenfalls DirectoryWriter implementiert. Anstatt einen LDAP-Server zu kontaktieren, schreibt er alle Schreibvorgänge in eine Datei im LDIF-Format. Diese Datei kann auf vertraulichem Wege zum LDAP-Server transportiert werden, und dort mithilfe eines Programms wie ldapModify und des Administrator-Passworts manuell eingepflegt werden. So werden bei der Authentisierung in keinem Fall Passwörter über das Netzwerk gesendet.

Da der private Schlüssel zum Signieren der Zertifikate nur der CA, nicht aber der RA bekannt ist, benötigt eine *Certification Authority* ein weiteres Schlüsselpaar für die Ver- bzw. Entschlüsselung der persönlichen Daten des Antragstellers im CSR (vgl. Abschnitt 5.4.1). RAEntry dient der RA zum Beantragen eines eigenen Zertifikats, was in den folgenden Abschnitten erläutert wird. CAService beschreibt den Zertifizierungsservice und enthält die beiden Zertifikate SecureNetService.serverCert und CAService.caCert, sowie eine Widerrufsliste CRL. Mit requestCertificate können Benutzer Zertifikate und mit requestCertificateRevocation einen Widerruf beantragen.

### 5.5.6 Effiziente Public-Key-Infrastruktur für DVEs

Alle Zertifikate werden also, wie im vorigen Abschnitt beschrieben, in einem Verzeichnis gespeichert, das über LDAP abgefragt und beschrieben werden kann. In diesem Abschnitt beschreiben wir die restlichen Teile der Public-Key-Infrastruktur (PKI), nämlich deren Verwaltung auf der Benutzer-Seite und deren Erzeugung und Widerruf durch eine CA. Die Entwurfsziele dabei waren:

- Minimierung des Überprüfungsaufwands von Zertifikaten.
- Keine unnötige Zentralisierung des Systems.
- Persistente Speicherung von Zertifikaten und geschützte Speicherung der zugehörigen privaten Schlüssel.
- Minimierung des Kommunikationsaufwands zur Verbreitung der Zertifikate.

#### Benutzerseite

Ein Benutzer der PKI ist ein Prinzipal, der mithilfe seines Zertifikats und dem dazugehörigen privaten Schlüssel sicher kommunizieren möchte, also insbesondere ein Teilnehmer oder ein Server. Außerdem möchte er die Zertifikate seiner Kommunikationsteilnehmer verifizieren und eventuell persistent speichern.

Die Verifikation eines Zertifikats verläuft in mehreren Schritten: Zunächst muss eine *Zertifikatskette* gebildet werden, die mit einem zuverlässigen Zertifikat beginnt, mit dem zu überprüfenden Zertifikat endet und deren Glieder – abgesehen vom selbst signierten Betreiber-Zertifikat – jeweils durch den Vorgänger ausgestellt worden sind. Dieser Schritt wurde dadurch vereinfacht, dass die Zertifikatstruktur streng hierarchisch ist, also keine Querverweise enthält, und einen Baum bildet. Da der Registrierungs-Server eines Zertifikats darin aufgeführt ist, muss zur Konstruktion der Zertifikatskette nur rekursiv das Zertifikat des jeweiligen Registrierungs-Servers gefunden werden, bis ein Betreiber-Zertifikat erreicht wird. Gesucht wird in jedem Schritt ein Zertifikat zu einem gegebenen Subjekt. Das LDAP-Verzeichnis ermöglicht dies bereits in effizienter Weise, da alle Zertifikate als Attribute des Subjekts abgelegt sind. Um den Kommunikationsaufwand mit dem Verzeichnis-Server zu minimieren, werden bereits bekannte Zertifikate in einem lokalen Cache-Speicher abgelegt, der sie als Subjekt-Zertifikat-Paare in einer Hash-Tabelle organisiert.

Im Weiteren muss der Gültigkeitszeitraum jedes Zertifikats der Kette geprüft werden. Zertifikate durchlaufen immer die Zustände „noch nicht gültig“, „gültig“ und „abgelaufen“ in genau dieser Reihenfolge. Ein abgelaufenes Zertifikat bleibt für immer ungültig. Ebenso prüft man den Widerruf: Ein widerrufenes Zertifikat kann nie wieder gültig gemacht werden. Das bedeutet, sowohl „abgelaufen“ als auch „widerrufen“ sind finale Zustände, in denen das Zertifikat auf ewig ungültig bleibt. Wir sehen deshalb bei der Speicherung von Zertifikaten ein zusätzliches *Flag* foreverInvalid vor. Läuft ein Zertifikat ab oder wird es widerrufen, so wird automatisch der gesamte Ast, der am zugehörigen Knoten im Zertifikatsbaum hängt, mit ungültig, und kann genauso als foreverInvalid markiert werden. Dies entspricht dem Schalenmodell der Zertifikatsgültigkeit, das dem Kettenmodell vorgezogen wurde, da der benötigte Gültigkeitszeitraum verglichen mit anderen Anwendungen relativ kurz ist und da Zertifikate dann mithilfe desselben Zertifikats widerrufen werden können, mit dem sie ausgestellt wurden, was auch die Implementierung erleichtert. Um schnell auf die Kindknoten eines Zertifikats zugreifen zu können, werden die Zertifikate als Baum abgespeichert.

Der letzte Schritt besteht in der Verifizierung aller digitalen Signaturen der Zertifikatskette. Dies ist, obwohl unsere Zertifikatsketten aufgrund der Typisierung maximal drei Glieder lang sein sollten

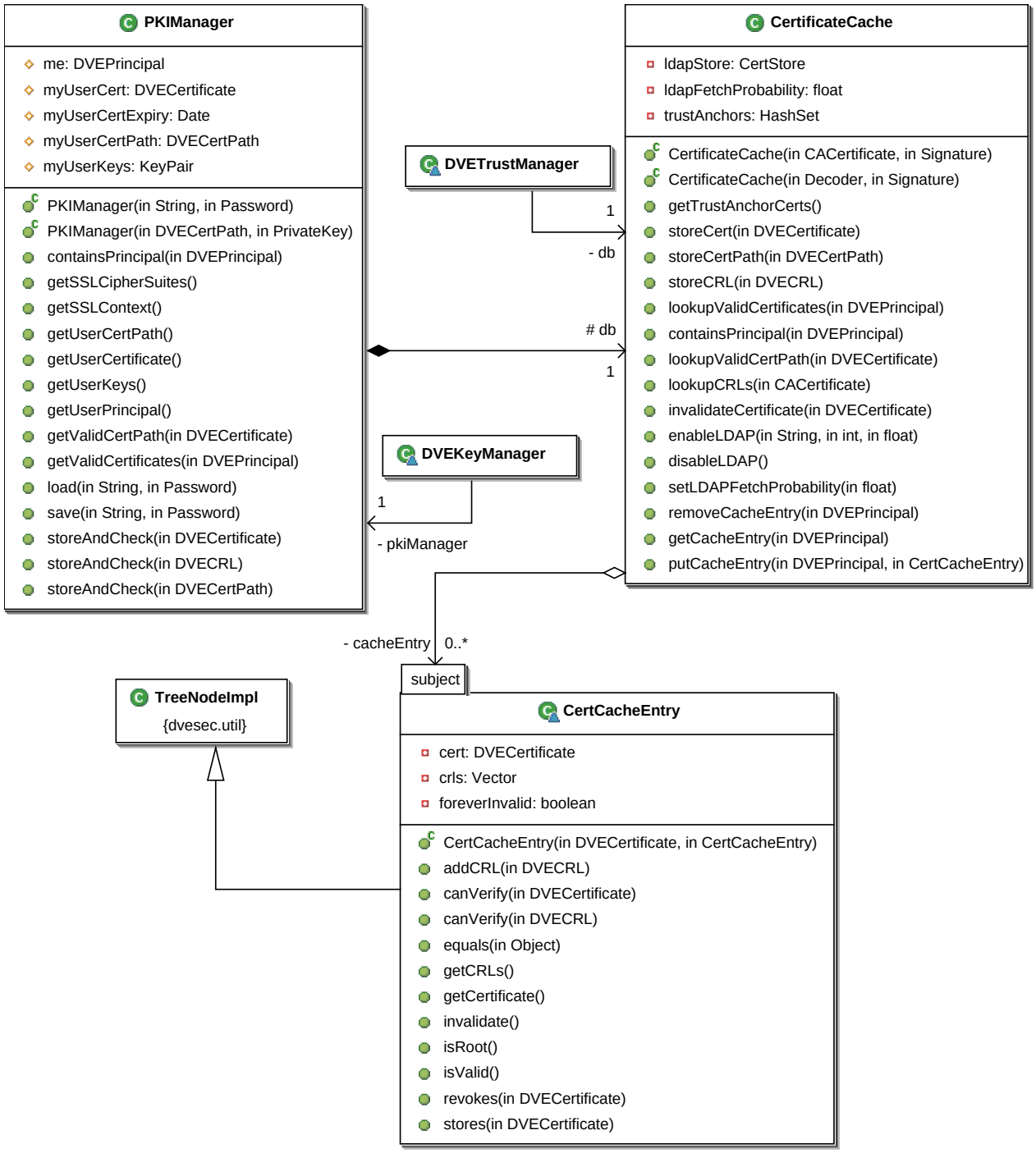



Abbildung 5.11: Benutzerseite des dvesec.cert.pki-Pakets

– ein Betreiber-, ein Registrierungs- und ein Server- bzw. Teilnehmer-Zertifikat –, eine rechnerisch aufwändige Operation (vgl. Abschnitt B.6). Signaturen sollten daher nicht unnötig oft geprüft werden. Wir markieren daher jedes Zertifikat, dessen Signatur bereits geprüft wurde, sodass bei der nächsten Überprüfung auf diesen Schritt verzichtet werden kann.

Abbildung 5.11 zeigt die wichtigsten Klassen des dvesec.cert.pki-Pakets für die Benutzer-Seite. Jedes Zertifikat entspricht einem Eintrag der Klasse CertCacheEntry. Die Einträge bilden einen Baum. Um einen Eintrag zu erzeugen, übergibt man dem Konstruktor das Zertifikat selbst, so-

wie den Eintrag des zugehörigen Registrierungs-Zertifikats. Noch im Konstruktor wird zunächst der Gültigkeitszeitraum, der Widerrufsstatus und schließlich die digitale Signatur überprüft. Dabei wird eventuell sofort das foreverInvalid-Flag gesetzt. Die Methoden canVerify geben true zurück, wenn der öffentliche Schlüssel des Zertifikats zur Verifikation der Signatur des Arguments benutzt werden kann. Die Methode invalidate() setzt das foreverInvalid-Flag eines Knotens und aller seiner Kinder. Widerrufslisten werden mit addCRL(DVECRL) in dem gleichen Eintrag gespeichert, um auch später eingefügte Zertifikate überprüfen zu können.

Der CertificateCache legt eine zusätzliche HashMap für die CertCacheEntries an. Die Einträge der zuverlässigen Betreiber-Zertifikate speichert er in der Menge trustAnchors. Die store-Methoden erzeugen gegebenenfalls neue Einträge und verknüpfen sie mit den vorhandenen. Analog kann man mit den lookup-Methoden auf Einträge im Cache zugreifen. Weiterhin kann ein LDAPCertStore abgefragt werden, um eventuell fehlende Zertifikate oder aktuelle Widerrufslisten aus einem Verzeichnis zu laden. Wie oft eine solche Abfrage stattfindet, wird durch das Attribut ldapFetchProbability definiert. Der Wert Eins bedeutet, dass alle fehlenden Zertifikate, bzw. bei jeder Zertifikatsprüfung die zugehörigen Widerrufslisten im Verzeichnis nachgeschlagen werden. Null deaktiviert den LDAP-Zugriff, und Werte dazwischen setzen die Wahrscheinlichkeit eines Zugriffs.

Der PKIManager stellt das Benutzer-Interface für alle wesentlichen Funktionen der PKI dar: Die Speicherung der persönlichen PKI-Daten – also Subjekt, Zertifikatskette, Zertifikat und Schlüsselpaar –, ein Frontend für den CertificateCache und eine Fabrik für SSL-Verbindungen. Außerdem können alle Daten in einer mithilfe einer Passphrase verschlüsselten und signierten Datei gespeichert werden, sodass ein Benutzer nicht für jede Sitzung neue Schlüssel und Zertifikate benötigt. Die Klassen DVEKeyManager und DVETrustManager bekommt der normale Benutzer nie zu Gesicht. Sie werden intern vom JSSE-Framework (vgl. Abschnitt B.3) für den Aufbau von TLS-Verbindungen benutzt.

### Certification-Authority-Seite

Die Klassen der CA-Seite bauen stufenweise aufeinander auf. Abbildung 5.12 zeigt ein Klassendiagramm. Zu Grunde liegt der bereits bekannte dvesec.cert.DVECertificateSigner. Mit ihm lassen sich mithilfe eines privaten Schlüssels und eines CACertificate-Objekts Zertifikate generieren. Der CAManager ist vom PKIManager abgeleitet, und speichert also die Schlüssel und Zertifikate der CA. Außerdem kann er Widerrufslisten generieren und führt Buch über die verwendeten IDs und die ausgestellten Zertifikate. Die Methode newPrincipal liefert einen neuen Prinzipal mit einer gültigen individuellen ID. Jeder CAManager verteilt die IDs im Bereich von seiner eigenen ID me.getId()+1 bis me.getId()+MAX_CERTS_PER_CA.

Der RootManager erfüllt die gleiche Funktion für die Betreiber-CA. Im Gegensatz zum CAManager kann er Registrierungs-Zertifikate verteilen und besitzt keine Beschränkung des ID-Bereichs[3]. Nach jedem ausgestellten Registrierungs-Zertifikat erhöht er den nextFreePrincipalId-Zähler um den Wert der Konstante MAX_CERTS_PER_CA, sodass sich die ID-Bereiche nicht überschneiden können. An dieser Stelle muss noch einmal angemerkt werden, dass die Registrierungs-Server zur *trusted computing base* gehören, und daher vertrauenswürdig über die Einhaltung der ID-Grenze wachen.

Ein CAManager wird von einer CertificationAuthority verwaltet, die CertificateSigningRequests und CertificateRevocationRequests (beide ohne Abbildung) bearbeitet. Die Entscheidung, welche Anträge bearbeitet werden, bzw. welche Zertifikate widerrufen werden, trifft ein Administrator über

[3] außer denen des Integer-Bereichs

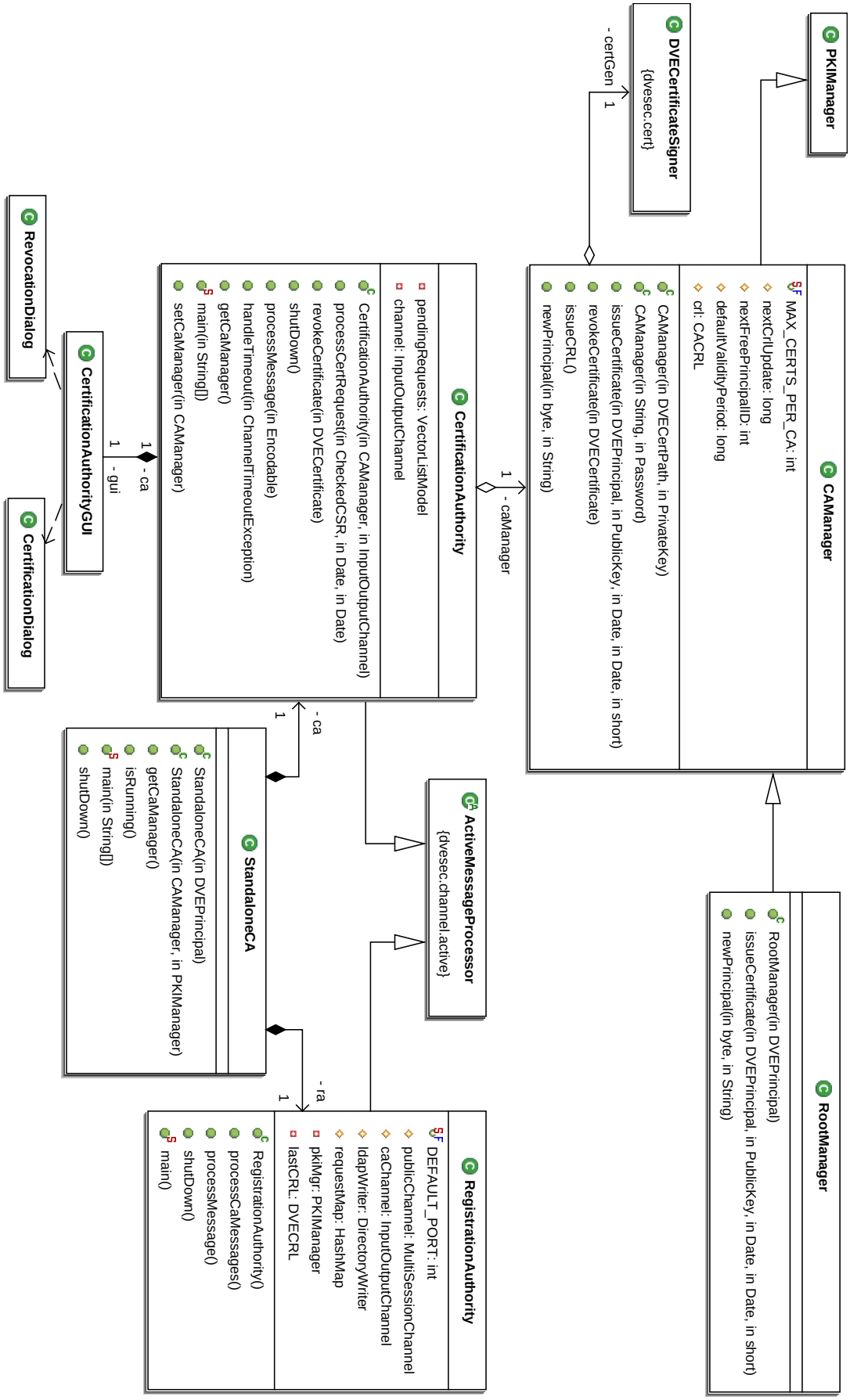



Abbildung 5.12: CA-Seite des dvesec.cert.pki-Pakets

die CertificationAuthorityGUI.

Die CertificationAuthority kommuniziert über einen Ein-/Ausgabe-Kanal channel mit der RegistrationAuthority, die über das öffentliche Netzwerk zu erreichen ist. Dadurch werden CA und RA getrennt. Die CA, die den wertvollen Zertifizierungsschlüssel speichert, kann mit einer Firewall vor allen Rechnern der Laufzeitumgebung außer ihrer RA abgeschottet werden oder gar offline arbeiten und nur über Dateien mithilfe eines dvesec.channel.io.GUIFileChannel mit der RA kommunizieren. Der Kommunikationskanal muss aber verlässlich und vertraulich sein. RA und CA können als eigenständige Programme mit eigener main()-Methode gestartet werden. Zu Beginn fordert die RegistrationAuthority von der CertificationAuthority ein Server-Zertifikat an, mit dessen Hilfe Bewerber vertrauliche Daten in ihren Anträgen schützen können. Die RegistrationAuthority versorgt einen Verzeichnis-Server über den ldapWriter mit Zertifikaten, Widerrufslisten und dem eigenen CA-Eintrag.

CA und RA erben beide von ActiveMessageProcessor (vgl. Abschnitt 3.4.4) und laufen daher jeweils in einem eigenen Thread. Da die Anzahl der für den Betrieb eines DVE verfügbaren Server wahrscheinlich begrenzt ist, können sie mit niedriger Priorität auf einem Rechner gestartet werden, auf dem bereits ein anderer Dienst läuft, ohne diesen zu sehr zu belasten.

Sollen CA und RA auf dem gleichen Rechner betrieben werden, kann man sie synchron als StandaloneCA starten. Die Kommunikation der beiden Komponenten erfolgt dann intern über einen dvesec.channel.io.QueueChannel.

### Zertifizierungsprozess

Um ein Zertifikat zu beantragen, generiert ein Antragsteller zunächst ein Schlüsselpaar. Als nächstes erzeugt er ein CertificateEntry-Objekt mit dem Namen, den er im DVE zu tragen wünscht. Dieser muss nicht eindeutig sein, da die Eindeutigkeit durch die ihm später zugewiesene ID gewährleistet wird. Jetzt benötigt er die Adresse eines Registrierungs-Servers. Mithilfe der statischen Methode DirectoryService.getDefaultService() erhält er ein DirectoryService-Objekt, mit dem er das LDAP-Verzeichnis mittels findEntries("CA") nach Registrierungsdiensten durchsuchen kann. Das Ergebnis besteht aus eventuell mehreren CAService-Objekten, aus denen er eines auswählen muss. Diesem übergibt er mit der Methode requestCertificate() seinen Eintrag, sein Schlüsselpaar und falls erforderlich seine persönlichen Daten.

Aus den Eingabedaten erzeugt der CAService ein CertificateSigningRequest-Objekt. Dazu werden die persönlichen Daten mit dem öffentlichen Schlüssel der RA, der ebenfalls im CAService-Objekt gespeichert ist, verschlüsselt, ein Zeitstempel hinzugefügt und eine digitale Signatur mithilfe des privaten Schlüssels des Antragstellers erzeugt. Der signierte CSR wird über einen von den Attributen protocol und address definierten Kanal an die RA gesendet.

Die RA empfängt den CSR und sieht zunächst nach, ob nicht bereits ein CSR von diesem Antragsteller vorliegt. Danach werden Zeitstempel, Signatur und nach Entschlüsselung die übrigen Daten überprüft, und der Antrag wird an die CA weitergesendet.

Die CertificationAuthority fragt eine neue User-ID beim CAManager ab und fügt den CSR in eine Liste zu bearbeitender Anträge ein. Über eine graphische Benutzeroberfläche entscheidet ein Administrator über Ausstellung oder Ablehnung. Automatische Entscheidungsinstanzen sind möglich, wurden aber nicht implementiert. In beiden Fällen wird die Entscheidung über die CertificationAuthority an den CAManager zurückgeliefert. Dieser erzeugt das neue Zertifikat, legt es zusammen mit den persönlichen Daten des Antragstellers in der Datenbank ab und liefert der CertificationAuthority eine Zertifikatskette vom Wurzel-Zertifikat bis zum neuen Benutzer-Zertifikat zurück. Diese sendet

die Kette in einer CertificationAnswer zurück an die RA.

Nach jeder empfangenen Nachricht über den öffentlichen Kanal sowie bei jedem Timeout prüft die RA, ob Nachrichten von der CA empfangen wurden. Auf diese Weise genügt ein gemeinsamer Thread für die Bedienung beider Kanäle. Liegt eine CertificationAnswer vor, wird der dazugehörige CSR gesucht, der Verzeichniseintrag entry um das Zertifikat ergänzt und über den ldapWriter im Verzeichnis abgelegt. Schließlich wird die CertificationAnswer an das CAService-Objekt des Antragstellers zurückgesendet.

Der CAService des Antragstellers prüft, ob die Antwort tatsächlich dem Antrag entspricht, extrahiert die neue Zertifikatskette und liefert sie als Ergebnis zurück.

Der frisch registrierte Teilnehmer instanziiert einen PKIManager mithilfe der Zertifikatskette und seines privaten Schlüssels. Damit seine Berechtigungsnachweise nicht verloren gehen, speichert er sie über den PKIManager verschlüsselt lokal ab. Dann kann er sich beim Verzeichnisdienst neu und diesmal authentisiert anmelden, um die Daten des nächsten Dienstes zu erhalten.

Der Widerruf eines Zertifikats erfolgt analog, nur dass nach Prüfung der Antragsdaten keine weitere Entscheidung notwendig ist. Widerrufene Zertifikate werden sowohl in der CRL der CA gespeichert, die in regelmäßigen Abständen an die RA und von dieser an den LDAP-Server weitergegeben wird, als auch sofort von der RA im Verzeichnis gelöscht.

### 5.5.7 Authentisierung und Zugangskontrolle

Die Authentisierung und die Zugangskontrolle zu den Diensten des Systems mithilfe von Zertifikaten wurde in den Klassen des Pakets dvesec.access umgesetzt. Sie liefern die Basis für die Implementierung aller zugangskontrollierten Dienste wie Koordinator, Kontrolleur, Verwalter, Benutzerdatenbank, Entitätendatenbank usf. Abbildung 5.13 zeigt das Klassendiagramm.

Alle Dienste, die Authentisierung und Zugangskontrolle erfordern, erweitern die Klasse AccessControlledServer. Ein AccessControlledServer stellt den autorisierten Benutzern einen Dienst über einen MultiSessionChannel bereit. Er verwaltet für jeden angemeldeten Benutzer eine eigene Sicherheitsassoziation. Zur Anmeldung dient ein TCPChannelServer. Die Authentisierung wird von einem Authenticator-Objekt übernommen. Dieses nimmt neue Anmeldungen über den TCPChannelServer an, authentifiziert den Neuling und authentisiert den Server, prüft mithilfe eines AccessControllPolicy-Objekts, ob dem Neuling der Zugang gewährt wird, verhandelt eine neue Sicherheitsassoziation und registriert nach geglücktem Protokolllauf eine neue Session mit der neuen Sicherheitsassoziation beim MultiSessionChannel. Der Authenticator ist ein Thread, sodass die Anmeldung neuer Sitzungsteilnehmer im Hintergrund erfolgen kann. Außerdem kann die Wartezeit zwischen zwei Anmeldungen mit setWaitTime(int) gesetzt werden, um die durch die Anmeldungen anfallende Rechenlast zu reduzieren. Als Authentisierungsprotokolle wurden die beiden Diffie-Hellman-Protokolle MIKEY (vgl. Abbildung 4.3) und SIGMA-I (vgl. Abbildung 4.4) implementiert. Die leichte Einbindung anderer Authentisierungsprotokolle wird durch das Interface AuthenticationProtocol ermöglicht. Die DefaultPolicy erlaubt jedem Benutzer mit einem gültigen Zertifikat den Zugang. Andere Zugangskontrollregeln, z.B. über Zugangskontrolllisten, können einfach als Subklassen von AccessControlPolicy realisiert werden.

Sämtliche Konfigurationsdaten für einen sicheren Dienst werden in der Klasse ServerConfig gespeichert. Dazu gehört auch der LDAP-Eintrag AccessControlledService des Dienstes. Benutzer erhalten ein AccessControlledService-Objekt über den Verzeichnisdienst, und können sich mit der Methode connect() anmelden. Nach geglückter Anmeldung sind der Ein- und Ausgabekanal des AccessControlledService-Objekts über MAC- und Verschlüsselungsfilter mit dem Ein- und Aus-


Abbildung 5.13: Klassendiagramm des dvesec.access-Pakets

gabekanal des AccessControlledServer verbunden.

### 5.5.8 Abrechnung

Für die Abrechnung der Teilnahmegebühr im Fantasy-Rollenspiel-Szenario müssen die Teilnahmezeiten der einzelnen Spieler erhoben werden.

Jedes Session-Objekt ermöglicht den Zugriff auf den Zeitpunkt seiner Erzeugung mit den Methoden getStartTime() und setStartTime(). Der MessageBuffer einer Nachricht aktualisiert bei der Deserialisierung den Zeitpunkt des letzten Nachrichtenempfangs der jeweiligen Sitzung mittels Session.setLastReceiveTime(). Somit können inaktive Sitzungen leicht erkannt werden und von Zeit zu Zeit automatisch geschlossen werden. Selbstverständlich müssen die Zugriffszeiten gespeichert werden.

In regelmäßigen Abständen meldet sich der Abrechnungs-Server an einem zuvor vereinbarten Socket bei jedem Kontrolleur an. Nach geglückter gegenseitiger Authentisierung mit dem SIGMA-I-Protokoll (vgl. Abbildung 4.4) und der Etablierung eines verlässlichen, verschlüsselten und authentisierten Kommunikationskanals überträgt der Kontrolleur die Teilnahmezeiten seiner Teilnehmer. Der Abrechnungs-Server kombiniert diese Daten mit den Personendaten und fordert die Gebühr ein.

Meldet sich der Abrechnungs-Server zu lange nicht, können die Kontrolleure die Teilnahmezeiten auch lokal in einer digital signierten und verschlüsselten Datei zwischenspeichern, um Platz im Hauptspeicher zu sparen.

## 5.6 Bewertung

In diesem Abschnitt führe ich einige Leistungstests durch und bewerte die Leistung der Implementierung der Sicherheitsarchitektur im Kontext der beiden Beispielszenarien.

### 5.6.1 Update-Nachrichten

In diesem Abschnitt wird der Einfluss kryptographischer Operationen auf den Versand von Update-Nachrichten quantifiziert.

Wie in Abschnitt 3.4.4 erläutert, werden in der DveSec-Architektur Filterobjekte vor die Kommunikationskanäle geschaltet, die jeweils eine kryptographische Operation implementieren. Jeder Filter verlängert die Verarbeitungslatenz auf Sender- und Empfängerseite und erhöht die Länge der Nachricht, wenn er zusätzliche Informationen an jedes Paket anfügt.

Untersucht wurden folgende vier Filterkonfigurationen:

**Direkt:** Aggregator auf der Senderseite, Splitter auf der Empfängerseite.

**MAC:** Aggregator und MacWriter auf der Senderseite, MacReader und Splitter auf der Empfängerseite.

**Enc:** Aggregator und Encryptor auf der Senderseite, Decryptor und Splitter auf der Empfängerseite.

**TS, Enc, MAC:** Aggregator, TimeStampWriter, Encryptor und MacWriter auf der Senderseite und MacReader, Decryptor, TimeStampReader und Splitter auf der Empfängerseite.

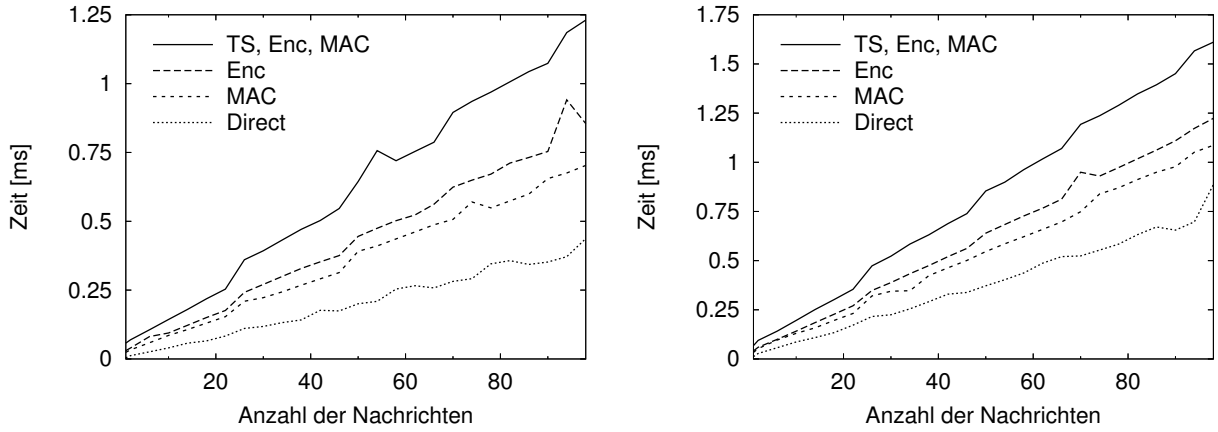



Abbildung 5.14: Verarbeitungsaufwand der kryptographischen Nachrichtenfilter

**Sign:** Aggregator und Signer auf der Senderseite, SignatureVerifier und Splitter auf der Empfängerseite.

Sender und Empfänger wurden über einen MBQueueChannel verbunden, der die gesendeten serialisierten Nachrichten in einer Warteschlange speichert, die vom Empfänger abgearbeitet wird.

Da Update-Nachrichten in ihrer serialisierten Form sehr klein sind, ist es ratsam, auf Senderseite alle Einzelnachrichten an denselben Empfänger mithilfe eines Aggregator-Objekts in möglichst großen Paketen zu aggregieren, bevor sie in die Filter-Pipeline eingespeist werden. Dadurch werden auch seltener aufwändige kryptographische Operationen ausgeführt und die Nachrichtenexpansion wird reduziert.

Sei $l_{MTU}$ die maximale Paketgröße der Verbindung in Bytes, $l_{msg}$ die Länge einer einzelnen Nachricht und $l_{hdr}$ die Länge aller Header, die die Filter einer Konfiguration erzeugen. Die maximale Anzahl $p$ der pro Paket aggregierten Nachrichten ist dann

$$p = \lfloor \frac{l_{MTU} - l_{hdr}}{l_{msg}} \rfloor .$$

Im Folgenden gehe ich von der Ethernet MTU von $l_{MTU} = 1500$ Byte aus. Die Länge einer Update-Nachricht setze ich auf $l_{msg} = 64$ Byte, da die Position und Orientierung in einer Entität im dreidimensionalen Raum als reelle $3 \times 4$-Transformationsmatrix dargestellt werden kann, also 48 Byte in einfacher Genauigkeit, und die restlichen 16 Byte beispielsweise Entitäten-ID, und Rundenzahl aufnehmen können.

Die maximale Anzahl von Updates pro Paket $p$ ist dann mit den Header-Längen aus Tabelle 3.10 für die ersten drei Kombinationen 23, für „TS, Enc, MAC“ 22 und für „Sign“ 20.

Um den zeitlichen Mehraufwand der Filter zu messen, wurden Zeitmessungen für vier verschiedene Filterkonfigurationen auf dem in Abschnitt B.7 beschriebenen Rechner vorgenommen. In jedem Testlauf wurden insgesamt 500.000 Update-Nachrichten versendet, von denen jeweils $n$ auf einmal an den Aggregator weitergereicht wurden. Dieser zerteilt die Übertragung in mehrere maximal 1500 Byte große Pakete. Die Messergebnisse sind in Abbildung 5.14 dargestellt. Die linke Abbildung zeigt dabei die Serialisierungszeit und die rechte die der Deserialisierung. Auf der $x$-Achse ist jeweils die Anzahl der Nachrichten pro Übertragung aufgetragen und auf der $y$-Achse die Verarbeitungszeit einer Übertragung in Millisekunden. Der Verlauf der Kurven ist im Großen und Ganzen linear. Bei der Deserialisierung fällt die Zeit zur Konstruktion neuer Update-Objekte

stärker ins Gewicht, sodass die reine filterlose Deserialisierungszeit daher etwa doppelt so lang wie die reine Serialisierungszeit ausfällt. Die anderen Filter verhalten sich im Vergleich dazu auf beiden Seiten etwa gleich. Die Berechnung eines MAC ist immer ein wenig schneller als die Chiffrierung, während die Kombination aus Zeitstempel, Chiffrierung und MAC etwas länger dauert als die reine Serialisierungszeit plus dem jeweiligen Overhead von MAC- und Chiffre-Filter.

Die Messungen der Signatur-Filter liegen deutlich höher und wurden daher nicht abgebildet. Für jedes Paket muss man auf der Senderseite etwa 20 ms veranschlagen und beim Empfänger weitere 1,5 ms. Im Falle der Unicast-Übertragung verursachen sie damit schon bei wenigen Teilnehmern untolerierbare Latenzen, und selbst bei Multicast-Versand ist es unwahrscheinlich, dass in jeder Runde 20 Millisekunden allein für die Serialisierung der Update-Nachrichten verwendet werden können. Der Einsatz digitaler Signaturen für die Update-Nachrichten kommt somit nicht infrage und wird daher im Folgenden auch nicht weiter berücksichtigt.

In einem DVE sendet jeder Teilnehmer pro Runde eine Update-Nachricht an den Verwalter bzw. Kontrolleur, der Kollisionen auflöst und die maßgeblichen bereinigten Updates an alle zurücksendet. Wir nehmen an, dass alle Kommunikationskanäle dieselbe Filterkonfiguration benutzen. Im Folgenden sei $n$ die Anzahl der Teilnehmer, $T_{send}(q)$ die Zeit, die die Filter auf der Senderseite zur Verarbeitung einer Übertragung mit $q$ Nachrichten benötigen, und $T_{recv}(q)$ die Zeit auf der Empfängerseite.

Wir betrachten zuerst die Verarbeitungszeit bei Multicast-Übertragung. Mit obiger Schreibweise berechnet die maximale Filterverarbeitungszeit $T_{filter}$ als

$$T_{filter} = T_{send}(1) + n \cdot T_{recv}(1) + T_{send}(n) + T_{recv}(n).$$

Die maximale benötigte Bandbreite $b_{filter}(n)$ der Update-Nachrichten einer Runde bei Multicast-Übertragung mit $n$ Teilnehmern berechnet sich dann als

$$b(n) = n(l_{msg} + l_{hdr}) + nl_{msg} + \lceil \frac{n}{p} \rceil \cdot l_{hdr}$$

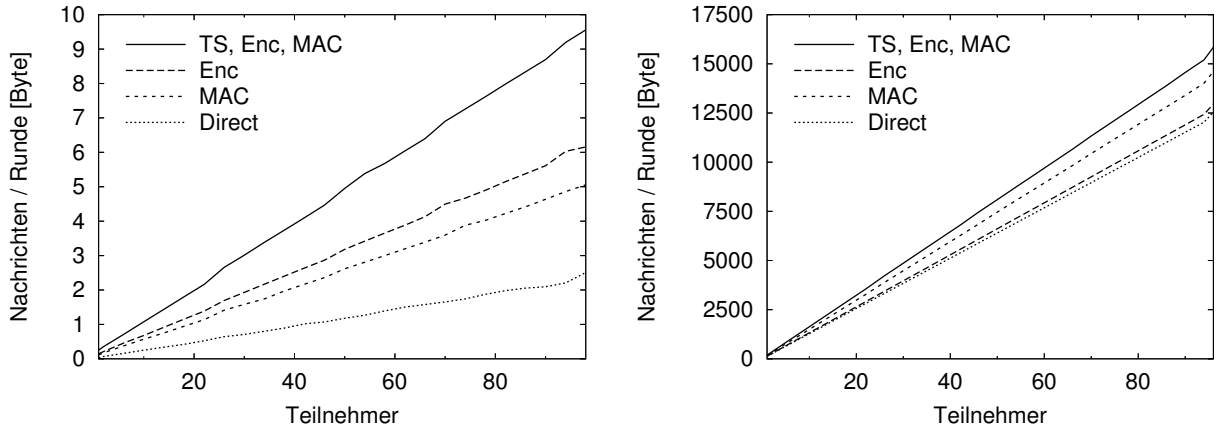
Bei Unicast-Übertragung müssen die maßgeblichen Updates einzeln versendet werden. Die Variable $m$ beschreibt die maximale Anzahl der Entitäten im Sichtbereich eines Teilnehmers. Wird vom Verwalter bzw. vom Koordinator keine Sichtbereichsberechnung ausgeführt, ist $m = n$. Es folgt

$$T_{filter} = T_{send}(1) + n \cdot T_{recv}(1) + n \cdot T_{send}(m) + T_{recv}(m).$$

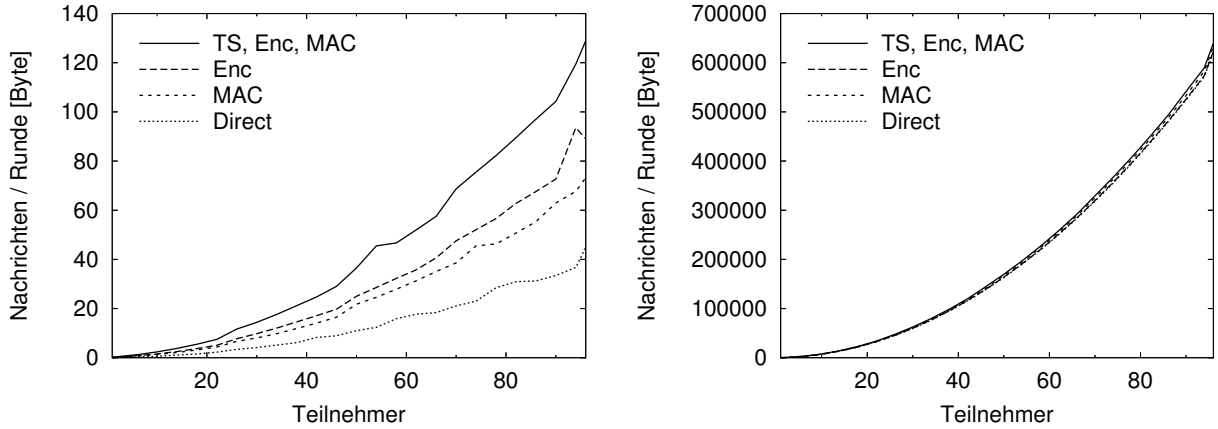
Für die maximale benötigte Bandbreite erhält man

$$b(n) = n(l_{msg} + l_{hdr}) + n\left(m \cdot l_{msg} + \lceil \frac{m}{p} \rceil \cdot l_{hdr}\right).$$

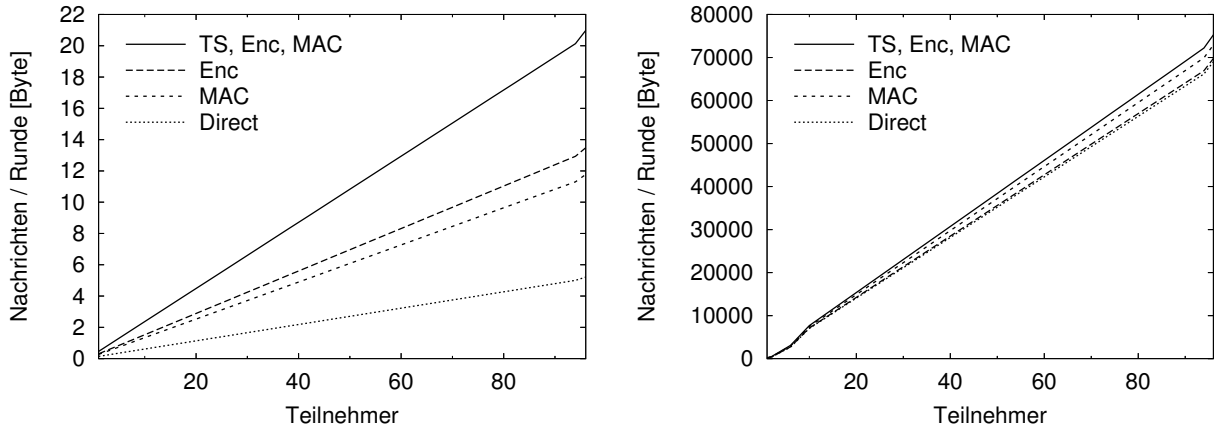
Abbildung 5.15 zeigt die Verarbeitungszeit und die benötigte Bandbreite in Abhängigkeit von der Teilnehmerzahl für die vier Filterkonfigurationen. Multicasting skaliert linear in Verarbeitungszeit und Bandbreite. Authentisiert man alle Update-Nachrichten mit einem MAC, so verdoppelt sich in etwa die Verarbeitungszeit durch die Filter-Pipeline, Verschlüsselung verlängert die Zeit etwa um den Faktor 2,5 und Zeitstempel, Verschlüsselung und MAC-Authentisierung zusammen etwa um den Faktor 3,8. Selbst bei 100 Teilnehmern liegt die gesamte Verarbeitungszeit aber noch unter 10 ms. Die benötigte Bandbreite liegt erwartungsgemäß sehr niedrig. Verschlüsselung produziert nur etwa 3% mehr Netzwerkvolumen, ein MAC 16% und die lange Filterkombination 26%. Angesichts der absoluten Zahlen sind alle diese Werte tolerierbar.




(a) Multicast-Übertragung




(b) Unicast-Übertragung




(c) Unicast-Übertragung mit Sichtbereichsberechnung

Abbildung 5.15: Verarbeitungszeit und Bandbreite für unterschiedliche Filterkombinationen

Unicasting ohne Sichtbereichseinschränkung skaliert im Vergleich viel schlechter. Sowohl die Verarbeitungszeit als auch das Nachrichtenvolumen steigen mit dem Quadrat der Teilnehmeranzahl. Die Verarbeitungszeit steigt durch einen MAC im Mittel um 90%, durch Verschlüsselung um 125% und für Zeitstempel, Verschlüsselung und MAC um 230%.

Beim Unicasting sind wir davon ausgegangen, dass dieselben Nachrichtenpakete für jeden Teilnehmer neu serialisiert werden müssen. Ist wie im Innenarchitektur-Szenario keine individuelle Verschlüsselung und Authentisierung notwendig, kann der Kontrolleur einen Multi-Session-Kanal (vgl. Abschnitt 3.4.5) benutzen, sodass jede Übertragung wie im Multicasting-Fall nur einmal serialisiert werden muss und erst danach an alle Teilnehmer versendet wird. Die Verarbeitungszeit im Filter ist dann exakt die gleiche wie bei Multicast-Übertragung. Das immense Netzwerkvolumen bleibt allerdings, wobei der Anteil der Header für alle Filter unwesentlich mit weniger als 3% zu Buche schlägt.

Unter der Voraussetzung, dass die Sichtbereichsberechnung nur maximal 10 Entitäten pro Teilnehmer liefert, skaliert auch Unicasting linear. Alle Verarbeitungszeiten verdoppeln sich im Vergleich zum Multicasting-Fall. Das Nachrichtenvolumen ist zwar etwa fünfmal so hoch wie bei Multicasting, liegt aber immer noch sehr niedrig. Bei einer Rundenzeit von 100ms genügt immer noch eine ISDN-Leitung, um alle Updates zu übertragen. Verschlüsselung erzeugt 1% mehr Nachrichtenvolumen, ein MAC 6% und die lange Filterkombination 9%.

Im Fantasy-Rollenspiel-Szenario sollte der Datenverkehr zwischen Teilnehmer und Kontrolleur mit einem MAC authentisiert werden, um Angriffe durch Veränderung der Nachricht (5.3.1), den Zugriff auf fremde Entitäten (5.3.3) und Privilegienbruch (5.3.3) auszuschließen. Die Messwerte lassen die Vorhersage zu, dass sich, wenn zusätzlich der Sichtbereich jedes Teilnehmers berücksichtigt wird, die Verarbeitungszeit jeder Nachricht in etwa verdoppelt. Da das Abhören eines Kommunikationskanals großes Expertenwissen und viel kriminelle Energie benötigt, sollte man auf Verschlüsselung verzichten, um eine weitere Verdopplung der Filterverarbeitungszeit zu vermeiden. Die absoluten Zahlen aus Abbildung 5.15 (c) zeigen, dass selbst bei 100 Teilnehmern pro Kontrolleur die Verarbeitungszeit durch die Filter noch unter 12 ms liegt. Bei einer Rundenzeit von 50 ms bleiben pro Runde immer noch 38 ms für die übrigen Aufgaben wie Kollisionsdetektion, Sichtbereichsberechnung etc. Diese Zeit sollte durchaus ausreichen, insbesondere angesichts der Tatsache, dass zukünftige Spiele-Server mit Sicherheit größer dimensioniert sind als der Testrechner. Ohne Sichtbereichsberechnung wird dieser Wert allerdings bereits bei 38 Teilnehmern erreicht.

Im Innenarchitektur-Szenario muss, um das DVE vor Abhören zu schützen, die Nachrichtenübertragung verschlüsselt und authentifiziert werden, sobald ein öffentliches Netzwerk benutzt wird. Die Nachrichtenverarbeitung in der Filterpipeline wird dann bei Multicasting und Unicasting mit Sichtbereichsberechnung vervierfacht, und für reines Unicasting verdreifacht. In absoluten Zahlen bedeutet das bei 20 Teilnehmern für Multicasting 2 ms statt 0,5 ms, für Unicasting 6,5 ms statt 2,1 ms und unter Berücksichtigung des Sichtbereichs 4,5 ms statt 1,1 ms. Angesichts dieser Zahlen dürfte die Verzögerung durch die kryptographischen Operationen für Benutzer nicht bemerkbar sein.

### 5.6.2 Weitere Empfehlungen

Generell sollte für jede Nachrichtenklasse (vgl. Abschnitt 3.4.1) ein separater Kommunikationskanal verwendet werden. Für die schnellen favorisiere ich einen UDP-basierten und für die zuverlässigen einen TCP/IP-basierten Kanal (vgl. Abschnitt 3.4.1).

Für die Datenablage empfehle ich eine SQL-Datenbank. Alle Daten sollten mit einem Zeitstem-

pel oder einer Versionsnummer abgelegt werden, sodass neue Versionen alte nicht überschreiben. Auf diese Weise kann ein früherer Zustand nach einem später erkannten Angriff wiederhergestellt werden. Selbstverständlich können überholte Daten nach Ablauf einer genügend großen Zeitspanne gelöscht werden. Da nur die Koordinatoren direkt auf die Datenbank zugreifen, sollten keine Verklemmungen auftreten.

Um den Übergriff des DVE auf andere Daten, Programme oder sonstige Ressourcen des ausführenden Rechners zu verhindern, sollten alle DVE-Programme innerhalb einer *Java-sandbox* (Abschnitt B.1 gestartet werden. So können die Zugangsrechte auf ein Minimum beschränkt werden. Zu den benötigten Privilegien gehören die Benutzung von Socket-Verbindungen, beschränkt auf ein Minimum von Zielrechnern und Ports, und die Möglichkeit, einen Logger zu benutzen. Je nach Anwendung können weitere Privilegien notwendig sein, z.B. zum Lesen und Schreiben bestimmter Verzeichnisse oder zum Laden von Klassenbibliotheken.

## 5.7 Zusammenfassung und Ausblick

In dieser Arbeit habe ich die Sicherheitsaspekte von verteilten virtuellen Umgebungen analysiert und daraus eine Sicherheitsarchitektur entwickelt, die die meisten Sicherheitsprobleme löst. Neben dem konzeptionellen Entwurf habe ich weite Teile des Systems implementiert und getestet. Da DVEs die vorhandenen Ressourcen ohnehin stark auslasten, wurde dabei großer Wert auf Schnelligkeit und Effizienz des resultierenden Systems gelegt, und trotzdem leichte Benutzbarkeit durch Abschirmung komplexer Implementierungsdetails gewährleistet. Außerdem wurden Standardtechnologien wie LDAP und X.509 eingesetzt.

Im ersten Kapitel wurde der Begriff der verteilten virtuellen Umgebungen beleuchtet, indem ich zunächst eine Definition (Abschnitt 2.1) und Beispiele (2.2) gegeben habe, danach die speziellen Anforderungen (2.3) untersucht habe und schließlich typische Entwurfsmuster (2.4 und 2.5) vorgestellt habe.

In der Analyse hat sich die Kommunikation unter den Teilnehmern als besondere Herausforderung herausgestellt. Bei der Wahl des Kommunikationsprotokolls muss ein Kompromiss zwischen Echtzeitfähigkeit, Skalierbarkeit und Verlässlichkeit gefunden werden, da nicht alle drei Eigenschaften gleichzeitig optimiert werden können (Abschnitt 2.3.3). Zu berücksichtigen ist auch die Dienstqualität des Netzwerks (3.2), insbesondere wenn das DVE über ein heterogenes Netzwerk betrieben wird.

Nicht alle Nachrichten in einem DVE unterliegen denselben Anforderungen. Einige Nachrichtentypen müssen in der Hauptsache schnell übertragen werden, während andere vor allem zuverlässig ausgeliefert werden müssen (Abschnitt 3.4.1). Daher kommen unterschiedliche Transportprotokolle zum Einsatz: Während IP-Multicasting (3.3.2) die effizienteste Art der schnellen *Viele-an-Viele-Kommunikation* kleiner Nachrichten darstellt, ist TCP/IP (3.3.4) ideal für die verlässliche, nicht zeitkritische Übertragung großer Datenmengen.

Um die Verarbeitungslatenz der Netzwerknachrichten niedrig zu halten, habe ich eine hochperformante Serialisierungsbibliothek entworfen und sowohl puffer- als auch datenstrombasiert implementiert (3.4.2). Die serialisierten Nachrichten sind deutlich kleiner als in der klassischen Java-IO, und die Geschwindigkeit ist besonders in der Pufferimplementierung deutlich höher.

Die für DVEs optimierte Kommunikationsbibliothek aus Abschnitt 3.4.3 abstrahiert den Kommunikationskanal, sodass unterschiedlichste Transportmedien auf die gleiche Art und Weise benutzt werden können. Wegen des hohen Aufkommens der schnell auszuliefernden kleinen Entitäten-Update-Nachrichten ist die Kanalabstraktion paketbasiert. Jede Nachricht wird nur einmal se-

rialisiert und deserialisiert. Für eine Vielzahl von Transportmedien wurden Kanäle implementiert, wobei die speziellen Vorteile, wie die Selektoren der Java-NIO, genutzt wurden.

Durch den Einsatz von Filtern (Abschnitt 3.4.4) können alle übertragenen Nachrichten effizient verschlüsselt, authentisiert, digital signiert oder mit einem Zeitstempel versehen werden. Ein Aggregator minimiert die benötigte Bandbreite und die Anzahl der notwendigen Filteroperationen. Der Programmierer kann selbst entscheidet, ob die Ergebnisse einer Filteroperation in der weiteren Anwendung relevant sind. Die Filterarchitektur befreit die Anwendung außerdem von den Einschränkungen monolithischer Sicherheitsprotokolle wie TLS.

Eine Anwendung kann dank der direkt in die Kanäle integrierten Sitzungsverwaltung (3.4.5) kontrolliert über einen einzigen Kanal mit mehreren Teilnehmern kommunizieren. Für UDP-Sockets wird mit der Sitzungsverwaltung das Java-Sicherheitskonzept der gebundenen Sockets auf mehrere Teilnehmer verallgemeinert. Multi-Session-Kanäle (3.4.5) abstrahieren Multicasting. Sie ermöglichen Multicast-Emulation auf der Basis mehrerer Unicast-Verbindungen, bei der die einmal serialisierte Nachricht nacheinander an alle registrierten Teilnehmer gesendet wird und Nachrichten über alle Unicast-Verbindungen empfangen werden können. Über die gleiche Schnittstelle können *application-layer-multicast-Protokolle* (3.3.5) und IP-Multicasting (3.3.2) benutzt werden. Dadurch können Multicast-Anwendungen implementiert werden, die mit entsprechenden Leistungseinbußen auch in Umgebungen lauffähig sind, in denen IP-Multicasting nicht zur Verfügung steht.

Die Rollen der verschiedenen Akteure eines DVE und deren Sicherheitsinteressen sind durchaus unterschiedlich und teilweise sogar konträr (Abschnitte 5.1.1, 5.1.2 und 5.1.3). Die Sicherheitsrichtlinie (4.3.4) muss dies berücksichtigen, einen fairen Kompromiss zwischen den Einzelinteressen schließen und vertraglich festhalten (5.4.4). Alle Beteiligten müssen die Richtlinie in verbindlicher Weise akzeptieren.

Die Sicherheit eines DVE muss bereits in der Grundarchitektur (Abschnitt 3.1) eines Systems verankert werden. Auch wenn eine Sicherheitsarchitektur für ein zentralisiertes Client-Server-System einfacher zu implementieren ist, ist es nicht unbedingt sicherer, da der Server gleichzeitig einen kritischen Ausfallpunkt und ein klares Angriffsziel darstellt, und somit der Sicherheitsaspekt der Verfügbarkeit des Systems gefährdet ist. Je stärker die Aufgaben verteilt werden, desto mehr Angriffspunkte enthält das System. Das System muss so konzipiert sein, dass der Schaden, der durch einen kompromittierten Teilnehmer entsteht, begrenzt ist, und sich der Koordinationsaufwand der Teilnehmer innerhalb vernünftiger Schranken bewegt. Allein aus den grundsätzlichen Sicherheitsanforderungen ist also keine Art der Aufgabenverteilung deutlich vorzuziehen.

Daher muss man sich stärker an der speziellen Anwendung des DVE orientieren. In Abschnitt 2.2 wurden eine Reihe Beispiele vorgestellt, die die Vielfalt der Anwendungen von DVEs demonstrieren. Eine Klassifizierung nach Sicherheitsaspekten erfolgte in Abschnitt 5.2. In durch *cheating* gefährdeten kompetitiven DVEs müssen alle maßgeblichen Entscheidungen von einer neutralen Entscheidungsinstanz getroffen werden. Für das Fantasy-Rollenspiel-Szenario (5.2.1) wurde deshalb eine verteilte Server-Architektur gewählt. Kollaborative DVEs wie die Innenarchitektur-Simulation (5.2.2) erlauben eine weniger zentralisierte Peer-to-Peer-Architektur.

Mithilfe von Zertifikaten (Abschnitt 5.4.1) lassen sich Benutzerverwaltung und Authentisierung dezentral organisieren. Durch die Typisierung von Zertifikaten (5.5.4) wird für jede Rolle eines Prinzipals im System der Verwendungszweck seines Zertifikats definiert. Mit dem LDAP-basierten Ankündigungs- und Verzeichnisdienst (5.5.5) können autorisierten Benutzern Zertifikate und Server-Dienste in verteilten DVEs angeboten werden. Dadurch wird auch eine Erweiterung des Systems zur Laufzeit um zusätzliche Server erleichtert. Die speziell angepasste effiziente Public-Key-Infrastruktur (5.5.6) minimiert den Aufwand der Zertifikatsprüfung und der Übertragung von

Zertifikaten, und erhöht die Skalierbarkeit der Zertifikatserzeugung, die sowohl über das Netzwerk zur Laufzeit als auch – um den Zertifizierungsschlüssel zu schützen – extern im Voraus erfolgen kann.

Die PKI und die Kanalarchitektur liefern weiterhin die Basis für alle zugangskontrollierten Dienste des Systems (5.5.7) mit einer dauerhaften Verbindung. Die verwendeten starken Authentisierungs- und Schlüsseletablierungsprotokolle können ausgetauscht werden. Ergebnis der erfolgreichen Anmeldung ist ein authentisierter und vertraulicher Kommunikationskanal.

Die vorgeschlagenen Authentisierungs- bzw. Verschlüsselungsoperationen für die Update-Nachrichten schaden der Skalierbarkeit und Geschwindigkeit des DVE in den beiden Beispielszenarien nicht wesentlich (5.6.1). Auch alle anderen Systembausteine sind so konzipiert, dass die Anzahl aufwändiger Operationen begrenzt werden kann und zentrale Bausteine repliziert werden können oder nur in seltenen Situationen in Aktion treten und die Funktion der restlichen Teilnehmer ansonsten nicht davon abhängt. Sichere DVEs sind also mit vertretbarem Aufwand realisierbar.

Die Sicherheitsarchitektur ist an vielen Stellen erweiterbar: Dank der klaren Schnittstellen können neue Nachrichtenklassen, Kanäle, Filter und Authentisierungsprotokolle einfach implementiert und sogar zur Laufzeit eingebunden werden.

Für die Zukunft wäre es interessant, die vorgeschlagene Sicherheitsarchitektur in ein DVE zu integrieren. Ein solcher Versuch schlug mit NPSNET-V leider fehl, da dieses zu keiner Zeit in einer stabilen Version vorlag. Die *Java.Net Community* arbeitet an diversen Projekten für eine Softwarearchitektur für Spiele in Java [JG]. Sobald diese eine gewisse Stabilität erreicht haben, kämen sie ebenfalls als Einsatzbereich infrage.

Da für *Massive Multiplayer On-line Role Play Games* häufig Server-Cluster eingesetzt werden, überschneiden sich die Themen Parallelverarbeitung und DVEs zunehmend. Es wäre beispielsweise interessant, wie sich die dort verfügbaren schnellen Kommunikationsverbindungen oder *shared-memory-Abstraktionen* für DVEs nutzen lassen, und welche Konsequenzen sich dadurch für die Sicherheit ergeben. Außerdem zeichnet sich ein Trend zum Einsatz von *Computational Grids* für DVEs ab (vgl. [Butt]).

Online-Computerspiele und vernetzte Simulationssysteme sind nicht zuletzt wegen der innerhalb der letzten Jahre massiv gestiegenen verfügbaren Netzwerkbandbreite auf dem Vormarsch. Die immer größeren Systeme werden auch in Zukunft immer wieder neue Sicherheitsprobleme aufwerfen, die sich zu erforschen lohnen.

# Literaturverzeichnis

Stand aller angegebenen WWW-Links ist der 8. März 2005.


[3DS] Discreet Inc.: *3D Studio MAX*.
http://www.discreet.com/products/3dsmax.

[AES] *AES: Advanced Encryption Standard*. National Institute of Standards and Technology, November 2001.
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[And01] Ross Anderson: *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley (New York), 2001.

[BC] The Legion of the Bouncy Castle: *Bouncy Castle JCE*.
http://www.bouncycastle.org.

[BL01] Nathaniel Baughman und Brian N. Levine: *Cheat-Proof Playout for Centralized and Distributed Online Games.* In *Proceedings of IEEE INFOCOM 2001*, pp 104–113, Anchorage, April 2001.
http://www.ieee-infocom.org/2001/paper/808.ps.

[Blx] Blaxxun Interactive Inc.: *Blaxxun Community*.
http://www.blaxxun.com/en/products/platform/index.html.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad und Michael Stal: *A System of Patterns*. Wiley (Chichester), 1996.

[BN00] Mihir Bellare und Chanathip Namprempre: *Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm*. In *Advances in Cryptology – Proceedings of ASIACRYPT 2000*, pp 531–545, 2000.
http://www.cs.ucsd.edu/users/mihir/papers/oem.pdf.

[BNet] Blizzard Entertainment: *Battle.Net*.
http://www.battle.net.

[Bri99] Bob Briscoe: *MARKS: Zero Side Effect Multicast Key Management Using Arbitrarily Revealed Key Sequences*. In *Proceedings of the First International COST264 Workshop on Networked Group Communication*, pp 301–320, Pisa, November 1999.

[Bru97] Don Brutzman: *Graphics Internetworking: Bottlenecks and Breakthroughs*. In Clark Dodsworth (Editor), *Digital Illusions*, pp 61–97. Addison-Wesley (Reading), 1997.
http://www.stl.nps.navy.mil/~brutzman/vrml/breakthroughs.html.

[BSI03a] Bundesamt für Sicherheit und Informationstechnik: *Leitfaden IT-Sicherheit*. *BSI Schriftenreihe zur IT-Sicherheit*, 2003.
http://www.bsi.de/gshb/Leitfaden/GS-Leitfaden.pdf.

[BSI03b] Bundesamt für Sicherheit und Informationstechnik: *E-Government-Handbuch*. *BSI Schriftenreihe zur IT-Sicherheit*, Band 11, 2003.
http://www.bsi.bund.de/fachthem/egov/6.htm.

[Butt] Butterfly.net: *Butterfly.net: The scalable, reliable and high performance online game platform.*
http://www.butterfly.net.

[Che] Stuart Cheshire: *Latency Survey Results*.
http://www.stuartcheshire.org/rants/LatencyResults.html.

[CN94] J.M. Carroll und S. Nudiati: *On Weak Keys and Weak Data: Foiling the Two Nemeses*. *Cryptologia*, 16(23), pp 253–280, Juli 1994.

[COM] Microsoft Corporation: *COM: Component Object Model*.
http://www.microsoft.com/com.

[Cos99] Greg Costikyan: *The Future of Online Games*. Creative Good, Inc. (New York), 1999.
http://www.costik.com.

[CRZ00] Yang-Hua Chu, Sanjay G. Rao und Hui Zhang: *A case for end system multicast*. In *Measurement and Modeling of Computer Systems*, pp 1–12, 2000.
http://www.cs.cmu.edu/~sanjay/Papers/jsac-2001.ps.gz.

[Del3D] The MOVES Institute: *Delta3D – Open Source Gaming and Simulation Engine*.
http://www.delta3d.org.

[Dep85] Department of Defense: *Department of Defense Trusted Computer System Evaluation Criteria*. Dec 1985. DoD 5200.28-STD.

[DES] National Institute of Standards und Technology (NIST): *DES: Data Encryption Standard*. FIPS 46-3, Oktober 1999.
http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[DH76] Whitfield Diffie und Martin E. Hellman: *New directions in cryptography*. *IEEE Transactions on Information Theory*, IT-22(6), pp 644–654, 1976.

[Dib] Julian Dibbell: *A Rape in Cyberspace*.
http://www.juliandibbell.com/texts/bungle_vv.html.

[DIS] IEEE Standards Association: *IEEE Standard for Distributed Interactive Simulation*. IEEE Standard 1278.1-1278.4, 1993-1998.

[Dive] Swedish Institute of Computer Science: *DIVE: Distributed Interactive Virtual Environment*. Kista.
http://www.sics.se/dive.

[DiX] Microsoft Corporation: *DirectX*.
http://www.microsoft.com/directx.

[DJV] DIS-Java-VRML Working Group: *DIS-Java-VRML*.
http://web.nps.navy.mil/~brutzman/vrtp/dis-java-vrml.

[DLL+00] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassem und Doug Balensiefen: *Deployment issues for the IP multicast service and architecture*. *IEEE Network*, 14(1), pp 78–88, 2000.
http://signl.cs.umass.edu/pubs/brian.ieeenetwork00.ps.gz.

[Dob96] Hans Dobbertin: *The Status of MD5 After a Recent Attack*. *CryptoBytes*, 2(2), pp 1–7, 1996.
ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf.

[Doom] IDGames: *Doom*.
http://www.idsoftware.com/games/doom/doom-final.

[DS] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang und W. Weiss: *An Architecture for Differentiated Services*. Internet Engineering Task Force. RFC 2475, Dezember 1998.
http://www.ietf.org/rfc/rfc2475.txt.

[DSS] National Institute of Standards and Technology (NIST): *Digital Signature Standard (DSS)*. U.S. Department Of Commerce. FIPS PUB 186, Mai 1994.
http://www.itl.nist.gov/fipspubs/fip186.htm.

[DvOW92] Whitfield Diffie, Paul C. van Oorschot und Michael J. Wiener: *Authentication and Authenticated Key Exchanges*. *Designs, Codes and Cryptography*, 2, pp 107–125, 1992.
http://www3.sympatico.ca/wienerfamily/Michael/MichaelPapers/STS.pdf.

[EQ] Sony Online Entertainment: *EverQuest*.
http://everquest.station.sony.com.

[GDOI] M. Baugher, T. Hardjono und H. Harney: *The Group Domain of Interpretation*. Internet Engineering Task Force. RFC 3547, Version 6, Juli 2004.
http://www.ietf.org/rfc/rfc3547.txt.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides: *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley (New York), 1995.

[Gon99] Li Gong: *Inside Java 2 Platform Security*. The Java Series. Addison-Wesley (Reading), 1st edition, 1999.

[Gre97] Richard Greenhill: *Diablo, and Online Multiplayer Game's Future*. *Games Domain Review*, Mai 1997.
http://www.gamesdomain.com/gdreview/zones/shareware/may97.html.

[GSAKMP] H. Harney, U. Meth, A. Colegrove und G. Gross: *GSAKMP*. Internet Engineering Task Force. Internet-Draft, Version 6, Juni 2004.
http://www.ietf.org/internet-drafts/draft-ietf-msec-gsakmp-sec-06.txt.

[HLA] IEEE Standards Association: *IEEE Standard for Modeling and Simulation High Level Architecture*. IEEE Standard 1516.1-1278.2, 2000.

[HLTV] Martin Otten: *HLTV: Half-Life TV*.
http://www.slipgate.de/hltv/hltv.html.

[HM03] Knut Håkon und Tolleshaug Mørch: *Cheating in Online Games – Threats and Solutions*. Technical Report DART/01/03, Norwegian Computing Center, Januar 2003.
www.nr.no/dart/projects/gisa/download/Cheating%20in%20Online%20Games.pdf.

[HMAC] Hugo Krawczyk, Mihir Bellare und Ran Canetti: *HMAC: Keyed-Hashing for Message Authentication*. Internet Engineering Task Force. RFC 2104, 1997.
http://www.ietf.org/rfc/rfc2104.txt.

[HT00] Thomas Hardjono und Gene Tsudik: *IP Multicast Security: Issues and Directions*. *Annales de Telecom*, pp 324–340, Juli 2000.
http://www.ics.uci.edu/~gts/paps/ht99.ps.gz.

[IAIK] Institute for Applied Information Processing and Communication: *IAIK-JCE*. TU Graz.
http://jce.iaik.tugraz.at/products/index.php.

[IP] Defense Advanced Research Projects Agency (DARPA): *DoD Standard Internet Protocol*. Internet Engineering Task Force. RFC 0760, Januar 1980.
http://www.ietf.org/rfc/rfc0760.txt.

[ISAKMP] D. Maughan, M. Schertler, M. Schneider und J. Turner: *Internet Security Association and Key Management Protocol (ISAKMP)*. Internet Engineering Task Force. RFC 2408, November 1998.
http://www.ietf.org/rfc/rfc2408.txt.

[J3D] Sun Microsystems: *Java3D API*.
http://java.sun.com/products/java-media/3D.

[JADE] University College London: *JADE: Java Adaptive Dynamic Environment*.
http://www.cs.ucl.ac.uk/staff/m.oliveira/research/jade.htm.

[Java] Sun Microsystems: *The Java 2 Platform*.
http://java.sun.com.

[JB] Sun Microsystems: *Java Beans*.
http://java.sun.com/products/javabeans.

[JG] Java.Net Community: *Java Games*.
http://community.java.net/games.

[JGJ+00] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek und James W. O'Toole, Jr.: *Overcast: Reliable Multicasting with an Overlay Network*. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, pp 197–212, Oktober 2000.
http://www.cs.brown.edu/~jj/papers/overcast-osdi00.pdf.

[JS] Sun Microsystems: *Java Servlet Technology*.
http://java.sun.com/products/servlet.

[JSP] Sun Microsystems: *Java Server Pages*.
http://java.sun.com/products/jsp/index.jsp.

[Kap03] Andrzej Kapolka: *The Extensible Run-Time Infrastructure (XRTI): An Experimental Implementation of Proposed Improvements to the High Level Architecture*. Master's thesis, Naval Postgraduate School (Monterrey), Dezember 2003.
http://www.movesinstitute.org/~kapolka/kapolka.pdf.

[KB03] Gu-In Kwon und John Byers: *ROMA: Reliable Overlay Multicast with Loosely Coupled TCP Connections*. Technical Report 2003-015, Boston University, Computer Science Department, Juli 2003.
http://www.cs.bu.edu/techreports/pdf/2003-015-roma.pdf.

[Ker83] Auguste Kerckhoffs: *La cryptographie militaire*. *Journal des sciences militaires*, IX, pp 5–38, 161–191, 1883.

[KMC02] Andrzej Kapolka, Don McGregor und Michael Capps: *A Unified Framework for Dynamically Extensible Virtual Environments*. In *Proceedings of the Fourth International Conference on Collaborative Virtual Environments*, Bonn, September 2002.
http://www.movesinstitute.org/Publications/unifiedcomponentframe.pdf.

[Kos] Raph Koster: *Raph Koster's Website*.
http://www.legendmud.org/raph/gaming.

[Kos98] Dave Kosiur: *IP Multicasting*. Wiley (New York), 1998.

[Kra03] Hugo Krawczyk: *SIGMA: the 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and its Use in the IKE Protocols*. WWW, Kurzversion des Papers in Proceedings of CRYPTO'03, Juni 2003.
http://www.ee.technion.ac.il/~hugo/sigma.ps.

[Kum96] Vinay Kumar: *MBone: Interactive Multimedia on the Internet*. New Riders Publishing (Indianapolis), 1996.

[KW00] Jan Köhnlein und Helmut Weberpals: *Softwarearchitektur verteilter 3D-Simulationssysteme*. *Mitteilungen – Gesellschaft für Informatik e.V., Parallel-Algorithmen und Rechnerstrukturen*, 18, pp 60–69, November 2000.

[KWD99] Frederick Kuhl, Richard Weatherly und Judith Dahmann: *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall (Upper Saddle River), 1999.

[Köh02] Jan Köhnlein: *Secure Multicasting for Distributed Virtual Environments*. In *Parallel Computing: Grids and Applications*, pp 107–121, Ghent, 2002.

[Köh04] Jan Köhnlein: *Security in a Peer-to-Peer Distributed Virtual Environment*. In *Parallel Computing: Software Technology, Algorithms, Architectures & Applications*, *Advances in Parallel Computing*, 13, pp 597–598, Elsevier (Amsterdam), 2004.

[Lau03] Jens Ove Lauf: *Vergleich verschiedener Algorithmen zur Sender-Authentifizierung von Multicast-Multimedia-Datenströmen*. Diplomarbeit an der Technischen Universität Hamburg-Harburg, Oktober 2003.
http://www.sva.tu-harburg.de/lauf/diplom.

[LDAP] W. Yeong, T. Howes und S. Kille: *LDAP: The Lightweight Directory Access Protocol*. Internet Engineering Task Force. RFC 1777, 1995.
http://www.ietf.org/rfc/rfc1777.txt.

[LKH] D. Wallner, E. Harder und R. Agee: *Key Management for Multicast: Issues and Architecture*. Internet Engineering Task Force. RFC 2627, 1999.
http://www.ietf.org/rfc/rfc2627.txt.

[LV01] Arjen K. Lenstra und Eric R. Verheul: *Selecting Cryptographic Key Sizes*. *Journal of Cryptology*, 14(4), pp 255–293, 2001.

[Mav] Advanced Interfaces Group, University of Manchester: *MAVERIK: MAnchester Virtual EnviRonment Interface Kernel*.
http://aig.cs.man.ac.uk/maverik.

[Maya] Alias Wavefront Inc.: *Maya*.
http://www.alias.com.

[MFW02] Martin Mauve, Stefan Fischer und Jörg Widmer: *A Generic Proxy System for Networked Computer Games*. In *Proceedings of NetGames2002*, Braunschweig, April 2002.
http://portal.acm.org/citation.cfm?id=566504.

[MIKEY] J. Arkko, E. Carrara, F. Lindholm, M. Naslund und K. Norrman: *MIKEY: Multimedia Internet KEYing*. Internet Engineering Task Force. RFC 3820, August 2004.
http://www.ietf.org/rfc/rfc3830.txt.

[MIKEY2] M. Euchner: *HMAC-authenticated Diffie-Hellman for MIKEY*. Internet Engineering Task Force. Internet-Draft, Version 7, Oktober 2004.
http://www.ietf.org/internet-drafts/draft-ietf-msec-mikey-dhhmac-07.txt.

[MN03] Rebecca T. Mercuri und Peter G. Neumann: *Security by Obscurity*. *Communications of the ACM*, 46(11), pp 160, November 2003.

[MOVES] *The MOVES Institute: Modeling, Virtual Environments and Simulation*. Naval Postgraduate School, Monterey.
http://www.nps.navy.mil.

[MPI2] Message Passing Interface Forum: *MPI-2: Extensions to the Message-Passing Interface*. 1997.
http://www.mpi-forum.org/docs/mpi-20.ps.

[MS03] David A. McGrew und Alan T. Sherman: *Key Establishment in Large Dynamic Groups Using One-Way Function Trees*. *IEEE Transactions on Software Engineering*, 29, pp 444–458, Mai 2003.
http://portal.acm.org/citation.cfm?id=776811#IndexTerms.

[MSEC] Internet Engineering Task Force: *IETF Working Group Multicast SECurity (MSEC)*.
http://www.securemulticast.org/msec-index.htm.

[MT95] D.C. Miller und J.A. Thorpe: *SIMNET: The Advent of Simulator Networking*. *Proceedings of the IEEE*, 83(8), pp 1114–1123, August 1995.
http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=400452.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot und Scott A. Vanstone: *Handbook of Applied Cryptography*. CRC Press (Boca Raton), 5th edition, Oktober 1997.
http://www.cacr.math.uwaterloo.ca/hac.

[NAT] K. Egevang und P. Francis: *The IP Network Address Translator (NAT)*. Internet Engineering Task Force. RFC 1631, Mai 1994.
http://www.ietf.org/rfc/rfc1631.txt.

[NPSNET] MOVES Institute: *NPSNET-V*. Naval Postgraduate School, Monterey.
http://www.npsnet.org/~npsnet/v.

[NTP] David L. Mills: *Network Time Protocol: Specification, Implementation and Analysis*. Internet Engineering Task Force. RFC 1305, Version 3, 1992.
http://www.ietf.org/rfc/rfc1305.txt.

[Oak01] Scott Oaks: *Java Security*. O'Reilly (Sebastopol), 2nd edition, 2001.

[OCSP] M. Myers, R. Ankney, A. Malpani, S. Galperin und C. Adams: *X.509 Internet Public Key Infrastructure – Online Certificate Status Protocol – OCSP*. Internet Engineering Task Force. RFC 2560, Juni 1999.
http://www.ietf.org/rfc/rfc2560.txt.

[OGL] *OpenGL: Open Graphics Library*.
http://www.opengl.org.

[Oli02] Manuel Oliveira: *Causing Mayhem on the Internet with Virtual Environments*. In *Proceedings of the Eleventh International World Wide Web Conference*, Honolulu, Mai 2002.
http://www2002.org/CDROM/alternate/344/index.html.

[Pan] *Pandora Anticheat*.
http://www.pandora-reloaded.de.

[Para] Stanford University: *PARADISE: Performance ARchitecture for Advanced Distributed Interactive Simulation Environments*.
http://www-dsg.stanford.edu/paradise.html.

[PB] Even Balance, Inc.: *PunkBuster Online Countermeasures*.
http://www.evenbalance.com.

[PCST01] Adrian Perrig, Ran Canetti, Dawn Song und J. D. Tygar: *Efficient and secure source authentication for multicast*. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2001)*, pp 35–46. Internet Society, Februar 2001.
http://www.isoc.org/isoc/conferences/ndss/01/2001/papers/perrig.pdf.

[PCTS00] Adrian Perrig, Ran Canetti, J. Doug Tygar und Dawn Xiaodong Song: *Efficient Authentication and Signing of Multicast Streams over Lossy Channels*. In *IEEE Symposium on Security and Privacy*, pp 56–73, 2000.
http://www.ece.cmu.edu/~adrian/projects/stream/stream.pdf.

[Per01] Adrian Perrig: *The BiBa one-time signature and broadcast authentication protocol*. In *Proceedings of the Eighth ACM Conference on Computer and Communication Security*, pp 28–37, November 2001.
http://portal.acm.org/citation.cfm?id=501988.

[PKCS1] RSA Laboratories: *PKCS #1 v2.1: RSA Cryptography Standard*. Juni 2002.
ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf.

[PR00] Erez Petrank und Charles Rackoff: *CBC MAC for Real-Time Data Sources*. *Journal of Cryptology*, 13(3), pp 315–338, 2000.
http://dimacs.rutgers.edu/TechnicalReports/abstracts/1997/97-26.html.

[Pri00] Matt Pritchard: *How to Hurt the Hackers: The Scoop on Internet Cheating and How You Can Combat It*. *Game Developer*, Juni 2000.
http://www.gamasutra.com/features/20000724/pritchard_01.htm.

[PSVW01] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma und Marcel Waldvogel: *ALMI: An Application Level Multicast Infrastructure*. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, pp 49–60, 2001.
http://www.arl.wustl.edu/~sherlia/papers/almi.ps.

[Qua] IDGames: *Quake*.
ftp://ftp.idsoftware.com/idstuff/source/q1source.zip.

[RJB99] James Rumbaugh, Ivar Jacobson und Grady Booch: *The Unified Modeling Language Reference Manual*. Addison-Wesley (Reading), 1999.

[RL96] Ronald L. Rivest und Butler Lampson: *SDSI – A Simple Distributed Security Infrastructure*. Presented at CRYPTO'96, April 1996.
http://theory.lcs.mit.edu/~rivest/sdsi10.ps.

[RNG] D. Eastlake, S. Crocker und J. Schiller: *Randomness Recommendations for Security*. RFC 1750, Dezember 1994.
http://www.ietf.org/rfc/rfc1750.txt.

[RSA78] Ronald L. Rivest, Adi Shamir und Leonard Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. *Communications of the ACM*, 21(2), pp 120–126, Februar 1978.
http://theory.lcs.mit.edu/~cis/pubs/rivest/rsapaper.ps.

[San00] Derek Sanderson: *Online Justice Systems*. *Gamasutra*, März 2000.
http://www.gamasutra.com/features/20000321/sanderson_01.htm.

[SASL] J. Myers: *Simple Authentication and Security Layer (SASL)*. Internet Engineering Task Force. RFC 2222, Oktober 1997.
http://www.ietf.org/rfc/rfc2222.txt.

[SC94] Sandeep K. Singhal und D.R. Cheriton: *Using a Position History-Based Protocol for Distributed Object Visualization*. Technical Report STAN-CS-TR-94-1505, Stanford University, Februar 1994.

[Sch96] Bruce Schneier: *Applied Cryptography*. Wiley (New York), 2nd edition, 1996.

[Sch00] Bruce Schneier: *Secrets and Lies: Digital Security in a Networked World*. Wiley (New York), 2nd edition, 2000.

[Sen02] D. Senie: *Network Address Translator (NAT)-Friendly Application Design Guidelines*. Internet Engineering Task Force. RFC 3235, Januar 2002.
http://www.ietf.org/rfc/rfc3235.txt.

[SKJH00] Sanjeev Setia, Samir Koussih, Sushil Jajodia und Eric Harder: *Kronos: A Scalable Group Re-Keying Approach for Secure Multicast*. In *IEEE Symposium on Security and Privacy*, pp 215–228, Oakland, Mai 2000.
http://www.cs.gmu.edu/~setia/kronos.pdf.

[SZ99] Sandeep Singhal und Michael Zyda: *Networked Virtual Environments: Design and Implementation*. Addison-Wesley (New York), 1999.

[Szy97] Clemens Szyperski: *Component Software*. Addison-Wesley (New York), 1997.

[TC96] W. J. Teahan und John G. Cleary: *The Entropy of English Using PPM-based Models*. In *Data Compression Conference*, pp 53–62, 1996.

[TCP] Defense Advanced Research Projects Agency: *Transmission Control Protocol*. Internet Engineering Task Force. RFC 793, September 1981.
http://www.ietf.org/rfc/rfc0793.txt.

[TLS] T. Dierks und C. Allen: *The TLS Protocol*. Internet Engineering Task Force. RFC 2246, Version 1.0, 1999.
http://www.ietf.org/rfc/rfc2246.txt.

[UDP] J. Postel: *User Datagram Protocol*. Internet Engineering Task Force. RFC 768, 1980.
http://www.ietf.org/rfc/rfc0768.txt.

[UO] Electronic Arts: *Ultima Online*.
http://www.uo.com.

[Vau02] Serge Vaudenay: *Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS*. In *Advances in Cryptology, EUROCRYPT'02*, *Lecture Notes in Computer Science*, 2332, pp 534–545, Springer (Amsterdam), 2002.

[VNET] Stephen White und Jeff Sonstein: *VNET*.
http://directory.fsf.org/graphics/anim/vnet.html.

[VRML] *Information technology – Computer graphics and image processing – The Virtual Reality Modeling Language*. The Web3D Consortium. International Standard ISO/IEC 14772-1:1997, 2:2002, 1997.
http://www.web3d.org/x3d/specifications/vrml.

[Wat01] Michael S. Wathen: *Dynamic Scalable Network Area of Interest Management for Virtual Worlds*. Master's thesis, Naval Postgraduate School (Monterey), September 2001.
http://www.npsnet.org/~npsnet/v/theses/wathen.pdf.

[WCS+99] Marcel Waldvogel, Germano Caronni, Dan Sun, Nathalie Weiler und Bernhard Plattner: *The VersaKey Framework: Versatile Group Key Management*. *IEEE Journal on Selected Areas in Communications, Special Issue on Middleware*, 8(17), August 1999.
http://www.tik.ee.ethz.ch/~versakey/JSAC-VersaKey.pdf.

[WP02] Lars Wolf und Lothar Pantel: *On the Suitability of Dead Reckoning Schemes for Games*. In *Proceedings of NetGames2002*, pp 79–84, Braunschweig, April 2002.
http://www.ibr.cs.tu-bs.de/events/netgames2002/presentations/wolf.pdf.

[WZ99] Ralph Wittmann und Martina Zitterbart: *Multicast: Protokolle und Anwendungen*. dpunkt Verlag (Heidelberg), 1999.

[X3D] *Information technology – Computer graphics and image processing – Extensible 3D (X3D)*. The Web3D Consortium. International Standard, Final Draft, ISO/IEC 19775:200x, 2002.
http://www.web3d.org/x3d/specifications/index.html.

[X509] Russell Housley, W. Ford, Tim Polk und D. Solo: *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. Internet Engineering Task Force. RFC 2459, Januar 1999.
http://www.ietf.org/rfc/rfc2459.txt.

[XRTI] The MOVES Institute: *XRTI: Extensible Run-Time Infrastructure*. Naval Postgraduate School, Monterey.
http://www.npsnet.org/~npsnet/xrti.

[YC02] Jianxin Jeff Yan und Hyun-Jin Choi: *Security Issues in Online Games*. *The Electronic Library: International Journal for the Application of Technology in Information Environments*, 20(2), pp 125–133, 2002.

# Anhang A

# Kryptographische Grundlagen

In diesem Teil des Anhangs werden die kryptographischen Grundlagen für die in Kapitel 5 entwickelte Sicherheitsarchitektur beschrieben. Eine generelle Einführung in die Kryptographie sowie eine ausführlichere Beschreibung der Grundlagen findet man beispielsweise in den Büchern von Schneier [Sch96] und Menezes, van Oorschot und Vanstone [MvOV97].

## A.1 Notation

In der Beschreibung von Protokollen und Algorithmen wird folgende Notation verwendet:

| | |
|---|---|
| $0x0123abcd$ | Binärstring in hexadezimaler Schreibweise |
| $\|k\|$ | Länge des Binärstrings $k$ |
| $k\|0x00$ | Verkettung der Binärstrings $k$ und $0x00$ |
| $x \otimes y$ | Exklusiv-Oder-Verknüpfung (XOR) der Binärstrings $x$ und $y$ |
| $E(K,m)$ | Chiffrierung des Klartexts $m$ mit dem Schlüssel $K$ |
| $D(K,c)$ | Dechiffrierung des Chiffretextes $c$ mit dem Schlüssel $K$ |
| $Mac_K(m)$ | Message Authentication Code über die Nachricht $m$ mittels Schlüssel $K$ |
| $Mac_K$ | Message Authentication Code mit $K$ über die bisherige Nachricht |
| $ID_x$ | Identität des Prinzipals $x$ |
| $K_x^+$ | Der öffentliche Schlüssel des Prinzipals $x$ |
| $K_x^-$ | Der private Schlüssel des Prinzipals $x$ |
| $Cert_x$ | Zertifikat des Prinzipals $x$ |
| $Sig_x(m)$ | Digitale Signatur der Nachricht $m$ mit dem privaten Schlüssel von $x$ |
| $Sig_x$ | Digitale Signatur über die bisherige Nachricht mit dem privaten Schlüssel von $x$. |
| $N$ | Nur einmal benutzte Zufallszahl (englisch *nonce*) |
| $T$ | Aktueller Zeitstempel |
| $[x]$ | Optionaler Bestandteil einer Nachricht |
| $m \in \mathbf{N}$ | Eine Nachricht $m$ in serialisierter Form als natürliche Zahl interpretiert |
| $\mathbf{N}_l$ | Die Menge aller als natürliche Zahlen interpretierten Binärstrings der Länge $l$, also $\{0,\ldots,2^l-1\}$ |

## A.2 Einweg- und Hashfunktionen

Sei $l \in \mathbf{N}$ die *Ausgabelänge*. Eine *Einwegfunktion* $F : \mathbf{N} \rightarrow \mathbf{N}_l$ ist eine einfach berechenbare Funktion mit der Eigenschaft, dass es zu jedem Bild $b \in \mathbf{N}_l$ rechnerisch unausführbar (englisch *computationally unfeasible*) ist, ein Urbild $m \in \mathbf{N}$ zu finden, sodass

$$F(m) = b.$$

Eine *Einweg-Hashfunktion H* besitzt zusätzlich *schwache Kollisionsresistenz*, d.h. es ist zu gegebenem $m \in \mathbf{N}$ und $h = H(m)$ rechnerisch unausführbar, ein $m' \in \mathbf{N}$ zu finden, sodass

$$H(m') = H(m) = h.$$

Eine *kryptographische Hashfunktion H* besitzt schließlich zusätzlich *starke Kollisionsresistenz*, d.h. es ist rechnerisch unausführbar $m$ und $m' \in \mathbf{N}$ zu finden, sodass

$$H(m) = H(m').$$

Die meisten Einweg-Hashfunktionen arbeiten die Nachricht $m$ intern iterativ in Blöcken einer festen Länge, der *Blocklänge* ab. Einweg-Hashfunktionen eignen sich unter anderem zum Integritätsschutz von Nachrichten und zur Erzeugung von Pseudo-Zufallszahlen. In beiden Fällen benötigt man Funktionen, die auf kleine Änderungen des Arguments mit möglichst großen Änderungen des Funktionswertes reagieren. Beispiele für gängige Einweg-Hashfunktionen sind *Message Digest 5* (MD5), *Secure Hash Algorithm* (SHA-1) und *RACE Integrity Primitives Evaluation Message Digest* (RIPEMD). Von der Ausgabelänge einer Hashfunktion hängt deren Kollisionsresistenz ab. MD5 mit einer Ausgabelänge von 128 Bit gilt inzwischen nicht mehr als kryptographische Hashfunktion, da Kollisionen gefunden wurden [Dob96]. Für sicherheitsrelevante Anwendungen wird daher eine Ausgabelänge von mindestens 160 Bit empfohlen, wie sie SHA-1 und RIPEMD-160 bieten.

## A.3 Zufallszahlen

In vielen kryptographischen Anwendungen benötigen wir Zufallszahlen. Schneier definiert salopp in [Sch96]: Eine Funktion

$$G : \mathbf{N} \rightarrow \{0,1\}$$

heißt *Pseudo-Zufallszahlgenerator*, wenn sie alle bekannten statistischen Tests auf Zufälligkeit besteht.

Ein *kryptographisch sicherer Pseudo-Zufallszahlgenerator* ist zusätzlich *praktisch nicht voraussagbar*, d.h. bei vollständiger Kenntnis des Algorithmus, der benutzten Hardware und aller bisher erzeugten Bits ist die Vorhersage des nächsten Bits *praktisch unberechenbar*. Dies wird durch zusätzliche geheime Eingabe- bzw. Zustandsdaten erreicht. Kryptographisch sichere Pseudo-Zufallszahlen benutzt man, wenn die Zufallszahlenfolge auf mehreren Rechnern synchronisiert werden soll. Der Generator wird jeweils mit dem geheim gehaltenen Schlüssel $k$ initialisiert, und produziert dann deterministisch auf jedem Rechner dieselbe Pseudo-Zufallszahlen-Folge. Der Generator muss allerdings so konzipiert sein, dass sich anhand der Pseudo-Zufallszahlen-Folge nicht

auf den Schlüssel $k$ schließen lässt. Ein solcher Generator lässt sich z.B. mithilfe einer Einweg-Hashfunktion $H$ konzipieren:

$$\begin{aligned} P(0) &= H(k) \\ P(i) &= H(k|P(i-1)), \quad \forall i \in \mathbf{N} \setminus \{0\}. \end{aligned}$$

wobei hier die Zufallszahlen nicht Bits sondern Wörter der Hash-Länge von $H$ sind. Ohne Kenntnis des Schlüssels $k$ kann ein Angreifer wegen der Einweg-Eigenschaft der Funktion $H$ nicht dieselbe Pseudo-Zufallszahlen-Folge konstruieren.

Ein *echter Zufallszahlgenerator* kann zusätzlich nicht verlässlich reproduziert werden: Bei zweimaliger Verwendung exakt derselben Eingabedaten werden völlig unabhängige Zufallszahlen generiert.

### Entropie

Was bedeutet Zufälligkeit und woher kommt sie? Für eine etwas genauere Betrachtung sei $Z$ eine diskrete Zufallsvariable

$$\begin{aligned} Z : \mathbf{N} &\to \mathbf{N}_n \\ m &\mapsto G(nm)|G(nm+1)|G(nm+2)|\ldots|G(n(m+1)-1), \end{aligned}$$

die den Zufallszahlgenerator $G$ benutzt, um bitweise $n$-Bit-Zufallszahlen zu generieren. Sei weiterhin

$$p_k = P(Z = k), \quad \forall k \in \mathbf{N}_n, \tag{A.1}$$

die Wahrscheinlichkeit, dass $Z$ die Zahl $k$ liefert. Die *Entropie* $H$ von $Z$ ist dann definiert als

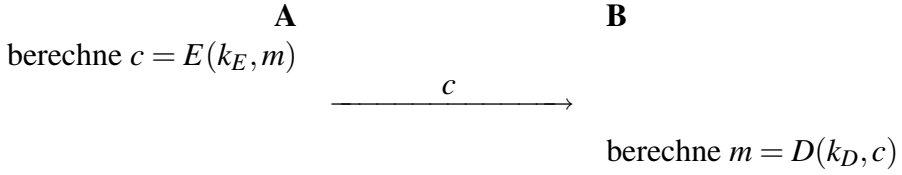
$$H(Z) = -\sum_{i=0}^{2^n-1} p_i \log_2 p_i. \tag{A.2}$$

Die Entropie von $Z$ misst die *Ungewissheit* (englisch *uncertainty*) und damit die Zufälligkeit der Funktionswerte: In unserem Fall kann die Entropie einen Wert zwischen 0 und $\log n$ einnehmen. Wenn $Z$ immer nur die gleiche Zufallszahl $k_0$ liefert, gibt es keinerlei Unsicherheit im Ergebnis, und die Entropie ist folglich 0. Wenn alle möglichen Ergebnisse $k \in \mathbf{N}_n$ gleich wahrscheinlich sind, erreicht die Entropie ihr Maximum von $\log n$.

Entropie kann nicht durch deterministische Berechnungen erzeugt oder erhöht werden. Werden Schlüssel mittels Hash-Funktionen aus Passwörtern erzeugt, enthalten diese nicht mehr Entropie als die Passwörter selbst. Daher werden Zufallszahlen unter Einbeziehung von Messwerten scheinbar zufälliger äußerer Einflüsse generiert, z.B. Temperatur der CPU, Variationen der Drehzahl der Festplatte durch Turbulenzen, zeitlicher Abstand zwischen Tastendrücken des Benutzers oder Abweichungen in der Mausbewegung, oft ohne die signifikantesten Stellen der Messwerte. In diesem Fall spricht man auch von *geheimer Entropie*, da scheinbare Entropie auf der Geheimhaltung der zusätzlichen Eingabeparameter beruht. Ein echter Zufallszahlgenerator benötigt eine echte Entropiequelle, z.B. eine Messung von Quanteneffekten. Andere Entropiequellen werden in [RNG] beschrieben.

## A.4 Verschlüsselung

Daten, insbesondere Nachrichten in einem sicheren System müssen oft vertraulich behandelt werden, und dürfen daher nicht im Klartext über öffentliche Datenleitungen verschickt werden. Um die Vertraulichkeit zu gewährleisten, wird der *Klartext* $m \in \mathbf{N}$ der Nachricht vom Sender vor der Übertragung mittels eines *Chiffrier-Schlüssels* $k_E \in \mathbf{N}_l$ und einer *Chiffrierfunktion* $E : \mathbf{N}_l \times \mathbf{N} \to \mathbf{N}$ *verschlüsselt*. Dabei bezeichnet $l \in \mathbf{N}$ die *Schlüssellänge*. Der entstehende *Chiffretext* $c$ kann von unautorisierten Dritten nicht entziffert werden, und kann daher vorbehaltlos über einen öffentlichen Kanal übertragen werden. Der Empfänger *entschlüsselt* den Chiffretext mithilfe des zu $k_E$ passenden *Dechiffrier-Schlüssels* $k_D \in \mathbf{N}_l$ und der zu $E$ gehörigen *Dechiffrierfunktion* $D : \mathbf{N}_l \times \mathbf{N} \to \mathbf{N}$ und erhält somit wieder den Klartext der Nachricht.




### A.4.1 Symmetrische Chiffrierung

In der *symmetrischen Chiffrierung* sind die Schlüssel für die Ver- und die Entschlüsselung identisch, d.h.

$$k_E = k_D = k$$

und

$$D(k, E(k, m)) = m, \quad \forall k \in \mathbf{N}_l \text{ und } \forall m \in \mathbf{N}.$$

Man unterscheidet *Stromchiffrierung* und *Blockchiffrierung*.

In einer Stromchiffrierung liefert ein Schlüsselgenerator eine unendliche Folge von Bits[1], die mit einer *Mischfunktion*, z.B. „exklusiv-oder-Verknüpfung“ oder „Addition modulo $n$“, sukzessive mit dem Klartext verknüpft wird. Der Empfänger besitzt denselben Schlüsselgenerator und kann durch Anwenden der *inversen Mischfunktion*, also „exklusiv-oder-Verknüpfung“ oder „Subtraktion modulo $n$“, den Klartext rekonstruieren. Dazu müssen allerdings beide Schlüsselgeneratoren zu Beginn der Übertragung mit demselben Chiffrierschlüssel initialisiert und synchronisiert werden. Der chiffrierte Text hat bei Stromchiffrierung die gleiche Länge wie der Klartext.

Im Gegensatz dazu verschlüsselt eine Blockchiffrierung die Nachricht blockweise. Aus Hardware-Gründen sind typische Blocklängen Zweierpotenzen, also 64, 128 oder 256 Bit. Nachrichten werden gegebenenfalls mit einem regelmäßigen Bitmuster auf ein Vielfaches der Blocklänge aufgefüllt (englisch *padding*). Dadurch wird der chiffrierte Text länger als der Klartext. Prominente Beispiele für Blockchiffrierungen sind der *Data Encryption Standard* (DES) und *DES Encryption Decryption Encryption* (DES-EDE bzw. 3DES, [DES]), *IDEA*, *Blowfish*, *RC1-6*, *SKIPJACK*, *Twofish*, *Serpent* und der neue *Advanced Encryption Standard* (AES, [AES]). Letzterer löste DES als offiziellen Chiffrieralgorithmus der US-Behörden ab.

Blockchiffrierungen können in bestimmten Betriebsmodi arbeiten. Im einfachsten Modus, dem *ECB-Modus* (englisch *electronic code book*), wird bei Verwendung desselben Schlüssels ein gegebener Klartextblock immer in denselben Chiffretextblock übersetzt. Dadurch kann ein Lauscher

[1] aus Effizienzgründen heute eher Bytes oder Speicherwörter

bei vielen ähnlichen Nachrichten bereits einige Informationen über die Konversation erhalten. Außerdem könnte er Chiffretext-Pakete untereinander austauschen, ohne dass der Empfänger das bemerkt. Diese Schwäche behebt der *CBC-Modus* (englisch *cipher block chaining*). Seien $P_i$ die eventuell aufgefüllten Klartextblöcke, $C_i$ die zugehörigen Chiffretextblöcke und $E$ und $D$ die Chiffrier- bzw. Dechiffrierfunktion zum Schlüssel $k$, dann funktioniert der CBC-Modus wie folgt:

$$\begin{aligned} C_i &= E(k, P_i \otimes C_{i-1}) \quad \text{Verschlüsselung} \\ P_i &= P_{i-1} \otimes D(k, C_i) \quad \text{Entschlüsselung} \\ &\forall i \in \mathbf{N} \setminus \{0\}. \end{aligned}$$

Zusätzlich wird ein *Initialisierungsvektor* $IV = C_0$ benötigt, mit dem Sender und Empfänger den Algorithmus synchronisieren. Der Initialisierungsvektor muss nicht geheim gehalten werden. Man sollte allerdings nie denselben Initialisierungsvektor mehrmals in Kombination mit demselben Schlüssel verwenden, da sonst der Vorteil gegenüber dem ECB-Modus verloren geht.

Blockchiffrierungen können auch zur Erzeugung von Stromchiffrierungen benutzt werden, indem man sie im *OFB-Modus* (englisch *output feedback*) oder *CFB-Modus* (englisch *cipher feedback*) betreibt. In beiden Fällen wird nur die Chiffrierfunktion benutzt.

Im OFB-Modus wird ein internes Schieberegister der Länge eines Blockes der Blockchiffrierung zu Beginn mit dem Initialisierungsvektor $IV \in \mathbf{N}_l$ geladen. Der Inhalt des Schieberegisters wird dann mit der Chiffrierfunktion $E$ und mit dem Schlüssel $k$ verschlüsselt. Die höchstwertigen $n$ Bits des entstehenden Chiffretextes bilden den Schlüsselstrom für die Stromchiffrierung, d.h. die nächsten $n$ Bits des Klartext werden unter Benutzung der Mischfunktion mit diesen Stromschlüsselbits verknüpft. Gleichzeitig werden diese Schlüsselbits von rechts in das Schieberegister eingeschoben, welches dann wieder verschlüsselt wird, usf. Verschlüsselung und Entschlüsselung arbeiten im OFB-Modus identisch. Da der Schlüsselstrom im OFB-Modus nicht vom Klar- und vom Chiffretext abhängt, kann er im Voraus berechnet werden (englisch *offline cipher*).

Der CFB-Modus funktioniert ähnlich. Chiffrierung und Dechiffrierung arbeiten allerdings asymmetrisch: Zur Chiffrierung werden statt der entstehenden Schlüsselbits die Bits des gerade berechneten Chiffretextes in das Schieberegister geladen, und für die Dechiffrierung sind es entsprechend die aktuellen zu dechiffrierenden $n$ Bits des Chiffretextes. Eine *offline-Berechnung* des Schlüsselstroms ist nicht mehr möglich.

Der letzte hier beschriebene Modus ist der *CTR-Modus* (englisch *counter mode*). Hier wird ein Schlüsselstrom $s_i$ mithilfe eines *Zählers* $n_i$ und der Chiffrierfunktion $E$ erstellt.

$$\begin{aligned} n_0 &= IV \\ n_i &= inc(n_{i-1}), \forall i \in \mathbf{N} \setminus \{0\} \\ s_i &= E(k, n_i) \\ C_i &= s_i \otimes P_i, \end{aligned}$$

wobei *inc* die *Inkrementfunktion* des Zählers ist. Im CTR-Modus kann man wahlfrei beliebige Einzelblöcke des Chiffretextes entschlüsseln, da der Stromschlüssel nur vom Zähler und nicht von der Nachricht selbst abhängt. Daher wird auch kein Padding benötigt.

Generell sollten alle verschlüsselten Nachrichten zusätzlich authentisiert werden, z.B. mit einem MAC. Insbesondere kann ein Angreifer, der den Klartext kennt, bei Stromchiffren, deren Schlüsselstrom weder vom Klar- noch vom Chiffretext abhängt, den Chiffretext so verändern, dass er zu jeder beliebigen Nachricht der gleichen Länge entschlüsselt wird (englisch *malleability*).

Weitere Modi für Blockchiffren und Sicherheits- und Effizienzvergleiche findet man in [Sch96].

### A.4.2 Asymmetrische Chiffrierung

Im Gegensatz zu der symmetrischen Chiffrierung benutzt man bei der asymmetrischen Chiffrierung zwei unterschiedliche Schlüssel für die Ver- und Entschlüsselung. Jeder Teilnehmer eines verschlüsselten Nachrichtenaustauschs benötigt ein eigenes Schlüsselpaar $(K^+, K^-)$. Da der Dechiffrierungsschlüssel $K^-$ geheim gehalten werden muss, heißt er auch *privater Schlüssel*, während der Chiffrierungsschlüssel $K^+$ der *öffentliche Schlüssel* genannt wird. Die vertrauliche Nachricht muss mit dem öffentlichen Schlüssel des Empfängers chiffriert werden. Da der Empfänger als einziger über den dazupassenden privaten Schlüssel verfügt, ist die Vertraulichkeit gewährleistet.

In einem bidirektionalen Nachrichtenaustausch muss jeder Teilnehmer den öffentlichen Schlüssel des anderen kennen. Bei der Verbreitung der öffentlichen Schlüssel kommen häufig *Zertifikate* zum Einsatz (vgl. Abschnitt 4.4.6).

Die Sicherheit asymmetrischer Verfahren beruht auf einer *Einwegfunktion mit Falltür*, d.h. einer Funktion, die einfach ausgewertet werden kann, deren Inverse allerdings rechnerisch extrem aufwändig zu berechnen ist, falls man das notwendige Geheimnis – die Falltür – nicht kennt. Ein Beispiel hierfür ist die Berechnung des Logarithmus in einer endlichen multiplikativen Gruppe $(G, *)$ mit großem Modulus $n \in \mathbf{N}$ oder das Faktorisieren großer Primzahlen.

Der bekannteste asymmetrische Chiffrieralgorithmus ist RSA [RSA78], nach seinen Erfindern Rivest, Shamir und Adleman benannt. Seine Sicherheit basiert auf der Faktorisierung großer Primzahlen. Die „Falltür“ von RSA wird durch Eulers Verallgemeinerung des kleinen Satzes von Fermat geöffnet.

Asymmetrische Verschlüsselung ist oft um ein Vielfaches langsamer als symmetrische Verschlüsselung (vgl. Abschnitt B.6). Sie bietet allerdings den Vorteil, dass keine geheimen Parameter vorab vereinbart sein müssen. Sie wird daher häufig nur bei der Etablierung einer Sicherheitsassoziation verwendet, sodass die weitere Kommunikation mithilfe schneller symmetrischer Algorithmen geschützt werden kann.

Weitere Algorithmen und spezifischere Details finden sich in [MvOV97].

## A.5 Diffie-Hellman-Schlüsselverhandlung

Die Diffie-Hellman-Schlüsselverhandlung (DH, [DH76]) dient der Verhandlung eines gemeinsamen geheimen Schlüssels zwischen zwei Kommunikationspartnern ohne vorher vereinbarte Geheimnisse.

Gegeben ist eine Primzahl $n \in \mathbf{N}$, die dazugehörige endliche multiplikative Gruppe $(\mathbf{Z}_n, * \bmod n)$ und ein Generator $g$ der Gruppe. Jeder Teilnehmer wählt einen privaten Schlüssel $x \in \mathbf{Z}_n$ bzw. $y \in \mathbf{Z}_n$ und berechnet daraus seinen öffentlichen Schlüssel $g^x \pmod n$, bzw. $g^y \pmod n$. Die öffentlichen Schlüssel werden ausgetauscht, woraufhin jeder Teilnehmer den gemeinsamen geheimen Schlüssel $g^{xy}$ berechnen kann. DH ist in Abbildung A.1 dargestellt.

Ein Angreifer, der den gesamten Nachrichtenverkehr abhört, kennt zwar die öffentlichen Schlüssel, ist aber nicht in der Lage, $g^{xy}$ zu berechnen, da er dazu entweder $x$ oder $y$ benötigen würde. Die Sicherheit des Protokolls wird also durch die Komplexität der Berechnung des diskreten Logarithmus in $(\mathbf{Z}_n, *)$ geliefert.

In der hier dargestellten Form stellt DH allerdings noch kein sicheres Protokoll dar, da *man-in-the-middle-Angriffe* möglich sind. Um dem vorzubeugen, sind zusätzliche Authentisierungsmechanismen notwendig. DH bildet die Basis vieler Schlüsseletablierungsprotokolle, beispielsweise im *Station-To-Station-Protokoll* (STS, [DvOW92]) oder in den in Abschnitt 4.6.2 beschriebenen

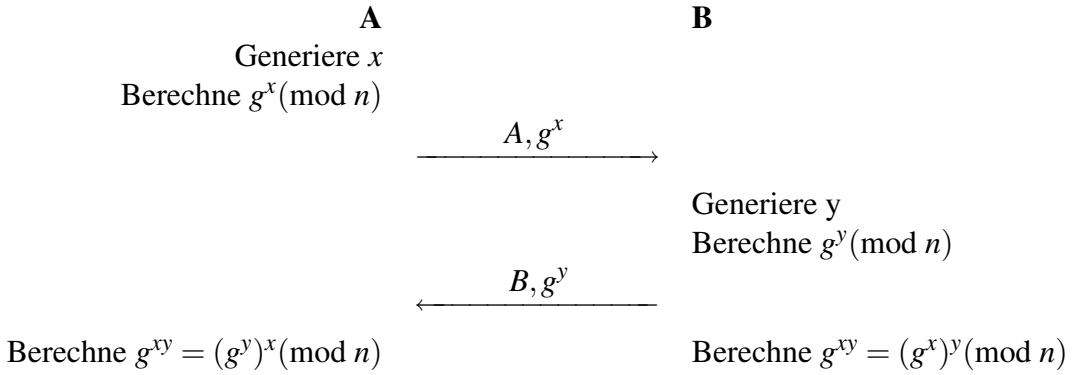



Abbildung A.1: Diffie-Hellman-Schlüsselverhandlung

Protokollen.

DH bietet *perfekte Vorwärtsgeheimhaltung*, wenn alle Schlüssel in jeder Sitzung neu generiert werden. Da die Erzeugung von $g^x$ jedoch rechnerisch aufwändig ist, werden z.B. in Server-Anwendungen oft private Exponenten (und damit auch die öffentlichen Schlüssel) für mehrere Sitzungen verwendet.

DH setzt voraus, dass beide Verhandlungsteilnehmer Interesse an der Etablierung eines gemeinsamen Geheimnisses haben. Wählt $B$ beispielsweise einen kleinen geheimen Exponenten $y$, sinkt der Aufwand eines *brute-force-Angriffs* zur Berechnung von $y$ zu einem abgehörten $g^y$ erheblich. Wählt $B$ den privaten Exponenten $y = 0$, so erhält der Lauscher den geheimen Schlüssel sogar direkt, denn $(g^0)^x = 1^x = 1$. $A$ hat keine Möglichkeit zu überprüfen, ob $y$ genügend groß gewählt wurde, außer selbst einen *brute-force-Angriff* durchzuführen.

DH kann auch auf mehr als zwei Teilnehmer verallgemeinert werden. Außerdem können andere endliche multiplikative Gruppen verwendet werden.

## A.6 Authentisierung von Nachrichten

Oft ist es nötig, die Urheberschaft einer Nachricht nachzuweisen. Dazu benutzt man entweder *Message Authentication Codes* oder *digitale Signaturen*.

Technisch gesehen weist die sich authentisierende Entität durch diese Mechanismen nur nach, dass sie einen bestimmten Schlüssel besitzt. In welcher Form sie durch die Authentisierung Verantwortung für den Inhalt der Nachricht übernimmt, muss in der Sicherheitsrichtlinie (vgl. Abschnitt 4.3.4) des Systems definiert werden. Die Sicherheitsrichtlinie sollte des Weiteren Regeln zur Bindung des Authentisierungsschlüssels an die Entität festlegen, z.B. die Notwendigkeit eines konventionellen Identitätsnachweises einer Person vor dem Ausstellen eines digitalen Zertifikats (vgl. Abschnitt 4.4.6). Da die Authentisierungsmechanismen selbst wie der DH-Schlüsselaustausch darauf basieren, dass die Geheimhaltung der Schlüssel im Interesse aller Beteiligten liegt, muss die Sicherheitsrichtlinie außerdem alle Benutzer zur Geheimhaltung der verwendeten Schlüssel verpflichten und eventuelle Strafmaßnahmen bei deren unerlaubter Weitergabe festlegen.

Da die Berechnung des Authentisierungscodes auch die Nachricht selbst einschließt, schützt er auch deren Integrität: Versucht ein unbefugter Dritter den Text der Nachricht zu verändern, ohne dass er den Code mitverändert, schlägt die Authentifizierung mit höchster Wahrscheinlichkeit fehl. Da der Authentisierungsschlüssel geheim gehalten wird, kann der Angreifer keine gültigen neuen Authentisierungscodes generieren.

### A.6.1 Message Authentication Codes

Ein *Message Authentication Code* (MAC) dient, wie der Name bereits sagt, der Authentisierung einer Nachricht, also dem Nachweis des Verfassers und der Integrität. MACs sind symmetrische Algorithmen, d.h. dass der Erzeuger einer Nachricht und der Überprüfende denselben Schlüssel $k$ benutzen. Es kann daher nicht unterschieden werden, wer von beiden die Nachricht gesendet hat. Außerdem kann ein außenstehender Dritter den MAC nicht verifizieren. Die Hauptanwendung von MACs liegt daher in der Authentisierung von Nachrichten innerhalb einer bereits ausgehandelten Sicherheitsassoziation mit gemeinsamem Geheimnis sowie innerhalb einer Gruppe, deren Mitglieder den geheimen Schlüssel $k$ teilen.

Sehr verbreitet sind die *Hash-basierten MACs* (HMAC, [HMAC]), üblicherweise definiert als

$$
\begin{aligned}
pad_i &= 0x3636\ldots \quad \text{(Länge b Bytes)} \\
pad_o &= 0x5c5c\ldots \quad \text{(Länge b Bytes)} \\
k_{temp} &= \begin{cases} H(k), & \text{falls |k| > b} \\ k, & \text{falls |k|=b} \\ k|0x00\ldots \text{(Länge b Bytes)}, & \text{falls |k|<b} \end{cases} \\
k_i &= pad_i \otimes k_{temp} \\
k_o &= pad_o \otimes k_{temp} \\
HMAC(k,m) &= H(k_o|H(k_i|m))
\end{aligned}
$$

mit einer Hashfunktion $H$ der Blocklänge $b$ und einem Signaturschlüssel $k$. Es wird empfohlen, dass der Schlüssel mindestens die Ausgabelänge der Hashfunktion $H$ hat.

Weitere MACs basieren auf symmetrischen Chiffren, siehe z.B. [PR00].

### A.6.2 Digitale Signaturen

Mit den im vorangegangenen Abschnitt beschriebenen MACs können Nachrichten auf Gruppenbasis authentisiert und verifiziert werden. Eine Unterscheidung der einzelnen Gruppenmitglieder ist allerdings ebenso ausgeschlossen wie die Verifizierung ohne den geheimen Signaturschlüssel. In beiden Fällen bietet sich die Verwendung von asymmetrischen Algorithmen zum Signieren an.

*Digitale Signaturen* dienen wie konventionelle Unterschriften zum Nachweis der Urheberschaft einer Nachricht. Der Verfasser einer Nachricht *signiert* diese, indem er eine *Signatur* aus der Nachricht selbst und einem *Signaturschlüssel* berechnet und diese der Nachricht anfügt. Der Empfänger kann diese Signatur unter Zuhilfenahme eines *Verifikationsschlüssels* mit einem entsprechenden Algorithmus *verifizieren*. Die Sicherheit der Signatur hängt von der Geheimhaltung des Signaturschlüssels ab.

Digitale Signaturen können auch verwendet werden, wenn ausgeschlossen werden soll, dass der Verfasser einer Nachricht später seine Urheberschaft abstreitet (englisch *non-repudiation*). Außerdem werden oft zusätzliche Daten, z.B. ein Zeitstempel zur Festlegung des Erstellungsdatums, mitsigniert.

Bei digitalen Signaturen wird nicht die Nachricht selbst signiert, sondern deren Hashwert. Da die Erzeugung der Signatur wesentlich aufwändiger ist als die Berechnung des Hashwerts, spielt die Länge der Nachricht für die Signierzeit nur eine geringe Rolle. Der Empfänger einer digital signierten Nachricht muss ebenfalls zuerst den Hashwert berechnen.

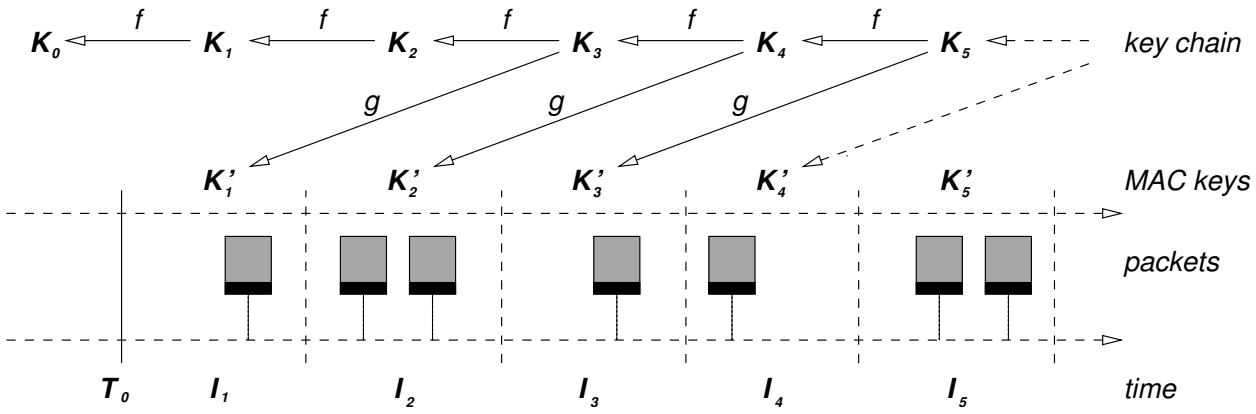



Abbildung A.2: Funktionsweise von TESLA

Die am häufigsten verwendeten digitalen Signaturalgorithmen sind der *Digital Signature Standard* [DSS] und RSA [RSA78]. Weitere Algorithmen findet man beispielsweise in [Sch96] und [MvOV97].

### A.6.3 TESLA

*Timed Efficient Stream Loss-tolerant Authentication* (TESLA, [PCST01]) ist ein Verfahren zur individuellen Senderauthentisierung von Multicast-Nachrichten. TESLA basiert auf einer zunächst geheimen *Einwegfunktionsschlüsselkette* (englisch *one-way function key chain*), die schrittweise aufgedeckt wird, und MACs für die Authentisierung der Nachrichten. Eine solche Schlüsselkette $(K_i)$, $i \in \{0,\ldots,n\}$ der Länge $n \in \mathbf{N}$ wird mithilfe einer Einwegfunktion $f : N_l \to \mathbf{N}_l$ rekursiv aus einem zufälligen Initialschlüssel $K_n \in \mathbf{N}_l$ generiert:

$$K_i = f(K_{i+1}) = f^{n-i}(K_n), \quad \forall i = 0,\ldots,n-1.$$

Aufgrund der Einwegfunktion kann man $K_{i+1}$ nicht aus $K_i$ berechnen, sehr wohl aber andersherum. In TESLA besitzt nun jeder Sender eine solche Kette. Die Sendezeit wird in $n-d$ gleich große Intervalle $(I_i)$, $i \in \{1,\ldots,n-d\}$ der Länge $T_{int}$ unterteilt, wobei die *Authentisierungsverzögerung* $d \in \mathbf{N}$ (englisch *authentication delay*) die Anzahl von Intervallen zwischen Benutzung und Aufdeckung eines Schlüssels $K_i$ bezeichnet. Jede Nachricht, die im Intervall $I_i$ gesendet wird, wird vom Sender mit einem MAC aus dem Schlüssel $K_{i+d}$ geschützt und deckt außerdem den Schlüssel $K_i$ auf. Jeder Empfänger speichert eine empfangene Nachricht so lange, bis der korrespondierende Schlüssel aufgedeckt wird. Er verifiziert durch Anwendung der Einwegfunktion, dass der aufgedeckte Schlüssel zur Schlüsselkette gehört. Wenn ja, kann er die Nachricht authentifizieren, indem er den MAC prüft. Da der aufgedeckte Schlüssel zum Empfangszeitpunkt noch geheim war, kann niemand die Nachricht gefälscht haben, und da der Schlüssel zur bis dahin geheimen Schlüsselkette gehört, stammt der MAC eindeutig vom Sender.

Bevor der Sender mit der Übertragung beginnt, schickt er eine digital signierte Nachricht, die den ersten Schlüssel $K_0$, die Länge der Schlüsselkette $n$, das Zeitinterval $T_{int}$, die Authentisierungsverzögerung $d$ und den Zeitpunkt des Übertragungsbeginns enthält. Dies ist die einzige asymmetrische kryptographische Operation während des gesamten Zeitintervalls $I$. Empfänger prüfen die Signatur und warten bis zum Übertragungsbeginn.

Die Authentisierung mit TESLA ist in Abbildung A.2 skizziert. Um eine Nachricht zu authentisieren, ermittelt der Sender das aktuelle Zeitintervall $I_i$ und den dazugehörigen Schlüssel $K_{i+d}$

aus der Schlüsselkette. Von diesem leitet er mit einer weiteren Einwegfunktion $g$ den Schlüssel $K'_i = g(K_{i+d})$ für die Berechnung des MAC ab. Er fügt der Nachricht einen Zeitstempel hinzu, berechnet den MAC und hängt ihn ebenfalls an die Nachricht an. Zum Schluss wird noch der Schlüssel $K_i$ angehängt und damit den Empfängern offenbart.

Ein Empfänger prüft bei jeder eingehenden Nachricht zunächst die *Sicherheitsbedingung* (englisch *security condition*):

> Eine Nachricht wurde sicher empfangen, wenn der Empfänger sicher ist, dass der Sender sich noch nicht in dem Zeitinterval befindet, in dem der zugehörige Schlüssel offen gelegt wird.

Dadurch wird garantiert, dass der zugehörige Schlüssel noch niemandem offenbart worden sein kann, der ihn zum Fälschen dieser Nachricht benutzt haben könnte. Nachrichten, die die Sicherheitsbedingung verletzen, werden sofort gelöscht. Alle anderen werden gepuffert. Danach wird überprüft, ob der in der Nachricht aufgedeckte Schlüssel $K_i$ in die Schlüsselkette gehört, indem die Einwegfunktion $f$ angewendet wird. Entspricht das Ergebnis von $f^{i-j}(K_i)$ dem zuletzt aufgedeckten Schlüssel $K_j$, können die MACs aller gepufferten Nachrichten, die in den Intervallen $(I_k)$, $k = j-d+1,\ldots,i-d$ gesendet wurden, nach zusätzlicher Anwendung von $g$ verifiziert werden.

TESLA ist resistent gegenüber beliebigem Paketverlust, da fehlende Schlüssel aus später aufgedeckten Schlüsseln rekonstruiert werden können. Empfänger müssen jeweils nur den letzten offen gelegten Schlüssel speichern. Verglichen mit digitalen Signaturen und anderen Authentisierungsmechanismen für Multicasting sind sowohl Arbeitsaufwand als auch Nachrichtenexpansion von TESLA sehr gering.

Zwei große Nachteile hat TESLA allerdings auch. Der erste ist, dass die Empfänger Nachrichten puffern müssen, bis der korrespondierende Schlüssel aufgedeckt wird. Bei großem Sendevolumen oder langen Authentisierungsverzögerungen kann hier sehr viel Speicher vonnöten sein. Der andere Nachteil ist die Verzögerung der Authentisierung, und damit eine Erhöhung der Gesamtlatenz.

In [Köh04] habe ich bereits den Einsatz von TESLA in DVEs untersucht. Damit die Sicherheitsbedingung bei allen Empfängern erfüllt werden kann, muss die Authentisierungslatenz $dT_{int}$ mindestens das Doppelte der Netzwerklatenz zwischen dem Sender und dem am weitesten entfernten Empfänger betragen. Der Empfänger mit der größten Übertragungslatenz bremst damit alle anderen aus. Um stark unterschiedliche Latenzen besser zu meistern, kann man auch mehrere TESLA-Instanzen $\tau_k, \quad k = 1,\ldots,m$ mit unterschiedlichen Authentisierungsverzögerungen $d_k$ gleichzeitig verwenden [PCST01]. Dabei teilen alle Instanzen zwar dieselbe Schlüsselkette $(K_i)$ , benutzen aber eine individuelle Einwegfunktion $g_k$, um die MAC-Schlüssel $K_i^k$ aus den Schlüsseln

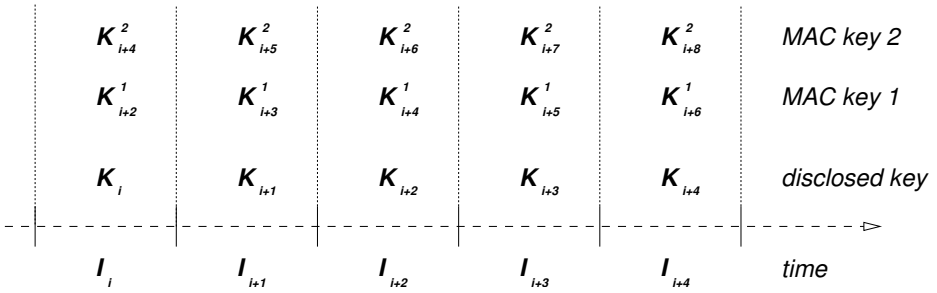



Abbildung A.3: Mehrere TESLA-Instanzen für unterschiedliche Authentisierungslatenzen

der Schlüsselkette als

$$K_i^k = g_k(K_{i+d_k}), \quad k = 1,\ldots,m \quad \text{und} \quad i = 1,\ldots,n - \max_{l=1,\ldots,m} d_l \tag{A.3}$$

abzuleiten. Jede TESLA-Instanz erzeugt einen eigenen MAC. Empfänger können dann die Instanz mit der niedrigsten Authentisierungslatenz wählen, die noch die Sicherheitsbedingung erfüllt. Das erweiterte TESLA-Schema ist in Abbildung A.3 abgebildet.

# Anhang B

# Sicherheit in Java

## B.1 Java 2 Sicherheitsmodell

Das Sicherheitsmodell der Java-2-Plattform [Java] stellt in der Hauptsache zwei Dienste bereit:

- Eine sichere, fertig-gebaute Laufzeitumgebung, auf der Java-Anwendungen auf sichere Weise ausgeführt werden können.
- In Java implementierte Sicherheitswerkzeuge und -dienste zur Erstellung anderer sicherer Anwendungen, beispielsweise in der Unternehmenswelt.

Ein Java-Programm kann in einer *sandbox* gestartet werden, die einschränkt, welche Dienste das Programm auf dem ausführenden Rechner nutzen darf, bzw. welche Operationen es ausführen darf. In Java 2 kann die *sandbox* über eine *policy*-Datei fein-granular in so genannte *protection domains* unterteilt werden, welche unterschiedlichen Programmpaketen je nach Code-Herkunft, ausführendem Benutzer oder Ersteller verschiedene Rechte für sicherheitsrelevante Dienste und Operationen zuweisen. Auf diese Weise kann der Benutzer den Schaden, den ein Programm anrichten kann, effektiv begrenzen. Um Modifikationen von Programmen durch Dritte auszuschließen, können Programmpakete außerdem digital signiert werden.

Voraussetzung dafür ist die sichere Laufzeitumgebung der Java-2-Plattform. Sie enthält z.B. einen *bytecode verifer*, der kompilierte Programme vor der Ausführung inspiziert. Außerdem ist Java typsicher, erlaubt die Kontrolle des Zugriffs auf Methoden und Attribute von Klassen, und kann Klassen, Methoden und Attribute als final, also nicht vererbbar oder überschreibbar definieren.

Das gesamte Sicherheitsmodell von Java ist ausführlich in [Gon99] und [Oak01] beschrieben.

## B.2 Java Cryptographic Architecture

Die *Java Cryptographic Architecture* (JCA) definiert Klassen und Interfaces für

- Die Verwaltung verschiedener Dienstanbieter von kryptographischen Algorithmen.
- Die Erzeugung von Hashes und kryptographisch sicheren Zufallszahlengeneratoren.
- Die Erzeugung, Repräsentation und Serialisierung kryptographischer Schlüssel.
- Die Verwendung digitaler Signaturen.

- Den Einsatz bereits erstellter X.509-Zertifikate.

Ein Provider ist die Schnittstelle einer Kryptographiebibliothek. Suns Java SDK wird von Haus aus mit einigen Providern ausgeliefert, nämlich SUN für die Basisklassen der JCA und DSA-Signaturen, SunRsaSign für RSA-Signaturen, SunJSSE für TLS-Verbindungen sowie SunJCE für symmetrische Chiffrierung und HMACs. Das Singleton-Objekt Security verwaltet die Provider. Alle Provider müssen digital signiert sein, wobei ein spezielles Zertifikat von Sun beantragt werden muss. Da die Signatur zur Laufzeit geprüft wird, wird die Erstellung eines eigenen Providers extrem aufwändig. Dieser Mechanismus dient dem Schutz von Unternehmensanwendungen, da die Integrität der kryptographischen Algorithmen ein absolutes Muss ist, entmutigt aber Programmierer, neue Algorithmen zu integrieren oder vorhandene effizienter zu implementieren.

Kryptographische Algorithmen werden über so genannte *engine classes* zur Verfügung gestellt. Diese besitzen die statische Methode getInstance(String, Provider), der der Name des gewünschten Algorithmus und eventuell der Provider übergeben wird, und die das gewünschte Objekt liefert. Ein Objekt zur Generierung von MD5-Hashes erhält man beispielsweise mit

```
java.security.MessageDigest.getInstance("MD5", new Provider("SUN")).
```

Die Erzeugung von Zertifikaten ist leider nur über das Kommandozeilenprogramm keytool möglich. Diese Entscheidung ist etwas merkwürdig, da das keytool selbst in Java geschrieben ist, und die Klassen zur Erzeugung von Zertifikaten daher vorhanden, aber nicht öffentlich zugänglich sind. Das Fehlen der Generierungsklassen ist der erste und oft einzige Grund, einen zusätzlichen externen Provider zu installieren. Da die Generierung von Zertifikaten von Sun nicht einmal als Schnittstelle definiert ist, entstehen hier im Programm Abhängigkeiten von speziellen Providern.

Wegen der Importbeschränkungen einiger Länder für kryptographische Algorithmen wird Java 2 mit „starker aber beschränkter Sicherheit“ ausgeliefert, sodass die kryptographischen Algorithmen nur mit relativ kurzen Schlüssellängen verwendet werden können. Um diese Algorithmen in voller Stärke zu benutzen, muss man die *unlimited strength jurisdiction policy* nachinstallieren.

## B.3 Java Secure Socket Extension

Für SSL- bzw. TLS-Verbindungen stellt Java die *Java Secure Socket Extension* (JSSE) bereit. Durch Erweiterung der Klasse java.net.Socket zur Klasse javax.net.ssl.SSLSocket können vorhandene TCP/IP-Verbindungen einfach durch TLS-Verbindungen ersetzt werden. Lediglich der Verbindungsaufbau unterscheidet sich leicht.

Zwei weitere Klassen spielen in der Authentisierungsphase eine Rolle: Ein Objekt der Klasse javax.net.ssl.KeyManager speichert die eigenen Schlüssel und Zertifikate. Die Entscheidung, ob das Zertifikat des Gegenübers vertrauenswürdig ist, übernimmt ein javax.net.ssl.TrustManager. Diese Aufgaben werden in der DveSec-Architektur vom DVEKeyManager bzw. DVETrustManager übernommen.

## B.4 Java Cryptographic Extension

Die *Java Cryptographic Extension* (JCE) definiert die Klassen und Schnittstellen für symmetrische Chiffrierung und Dechiffrierung, für MACs sowie für die Diffie-Hellman-Schlüsselverhandlung. Seit Version 1.4.2 ist endlich auch der AES-Algorithmus enthalten.

## B.5 Security-Provider anderer Anbieter

Eine Reihe anderer Anbieter hat ebenfalls Provider-Klassen implementiert, die die Standardprovider von Sun erweitern oder ersetzen.

Der wohl umfangreichste frei verfügbare Provider ist BouncyCastle [BC]. Er enthält u.a.

- eine leichtgewichtige API für Kryptographie in Java,
- einen Provider für JCA und JCE,
- eine völlige Neuimplementierung der JCE,
- eine Bibliothek zum Lesen und Schreiben von ASN.1-kodierten Objekten,
- Generatoren für X.509-Zertifikate und Widerrufslisten,
- Klassen zur Verarbeitung von S/MIME, CMS, OCSP und OpenPGP,
- eine digital signierte Version, die sich in Suns JCA einbinden lässt.

IAIK [IAIK] ist eine Kryptographiebibliothek der Technischen Universität Graz, die ähnlich umfangreich ist wie BouncyCastle. Für den akademischen Einsatz ist auch sie frei.

## B.6 Kryptographische Leistungstests

Um konkrete Werte für den rechnerischen Aufwand verschiedener kryptographischer Operationen zu erhalten, wurden für die in dieser Arbeit verwendeten Algorithmen Leistungstests durchgeführt.

Getestet wurden die im Java-SDK 1.4.2 enthaltenen Standard-Provider von Sun, SUN, SunJCE und SunRsaSign, BouncyCastle [BC] (Version 1.18) und IAIK [IAIK] (Version 3). Für alle Tests wurde die in Abschnitt B.7 beschriebene Testumgebung verwendet.

### B.6.1 Symmetrische Algorithmen

Zunächst untersuchen wir unterschiedliche symmetrische Algorithmen. Abbildung B.1 zeigt die Ergebnisse des HMAC-Tests. Als Schlüsselgröße wurde entsprechend der Ausgabegröße 128 Bit für MD5 und 160 Bit für SHA-1 gewählt. Der HMAC wurde für Nachrichten unterschiedlicher Länge berechnet, da er, zumindest im Bereich der hier behandelten kleinen Eingabedaten, nicht linear zur Nachrichtengröße ist[1]. Die Ergebnisse sind jeweils über 10000 Iterationen gemittelt. SUN ist bei MD5 langsamer, BouncyCastle bei SHA-1. IAIK arbeitet beinahe doppelt so schnell wie der jeweils langsamste Provider.

Auf einen direkten Test der Hash-Algorithmen MD5 und SHA-1 wurde verzichtet. Da ein HMAC in der Hauptsache aus zwei Hash-Operationen besteht (vgl. Abschnitt A.6.1), liegen die Zeiten etwa bei der Hälfte der Werte für den entsprechenden HMAC. Da kein zusätzlicher Schlüssel benötigt wird, ist ein zur Nachrichtenlänge proportionales Verhalten auch bei kleinen Nachrichten zu erwarten.

Als nächstes wurden unterschiedliche symmetrische Chiffrieralgorithmen untersucht. Alle Chiffren arbeiten im ECB-Modus. Gemessen wurden jeweils die Zeit zum Ver- wie zum Entschlüsseln

[1] Das Verhältnis der Schlüssellänge zur Länge der Gesamtnachricht ist hier sehr ungünstig.

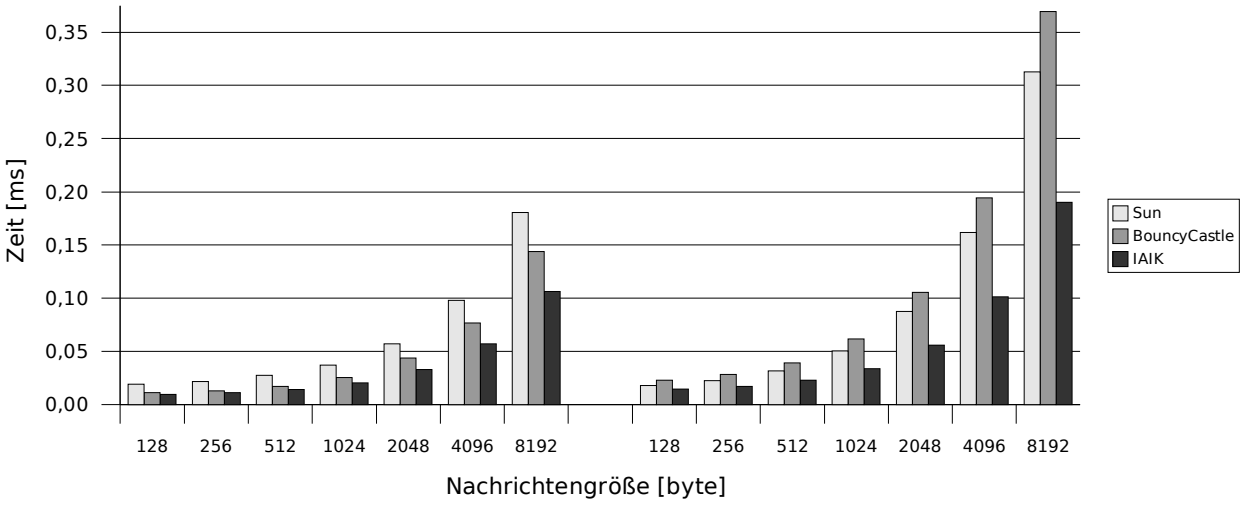



Abbildung B.1: HMAC Leistungstests

von 10000 Nachrichten von jeweils unterschiedlicher Länge. Da die jeweiligen Ver- und Entschlüsselungszeiten erwartungsgemäß fast gleich waren, ist hier nur deren Summe abgebildet. Außerdem haben sich die Bearbeitungszeiten ab einer Nachrichtenlänge von ca. 512 Byte als linear erwiesen, sodass wir hier nur die Rate in Mikrosekunden pro Byte sehen. Abbildung B.2 zeigt die Ergebnisse. Außer bei 3DES, bei dem Sun fast doppelt so lange braucht wie die anderen Provider, verhalten sich die verschiedenen Provider fast gleich. Auch große Schlüssellänge machen bei AES und Blowfish kaum einen Unterschied. IAIK hat die leicht bessere Blowfish-Implementierung.

## B.6.2 Asymmetrische Verfahren

Bei den asymmetrischen Verfahren wurde zunächst die Generierung von Schlüsselpaaren gemessen. Für RSA wurde der öffentliche Exponent 65537 gewählt und mit verschiedenen Modulus-

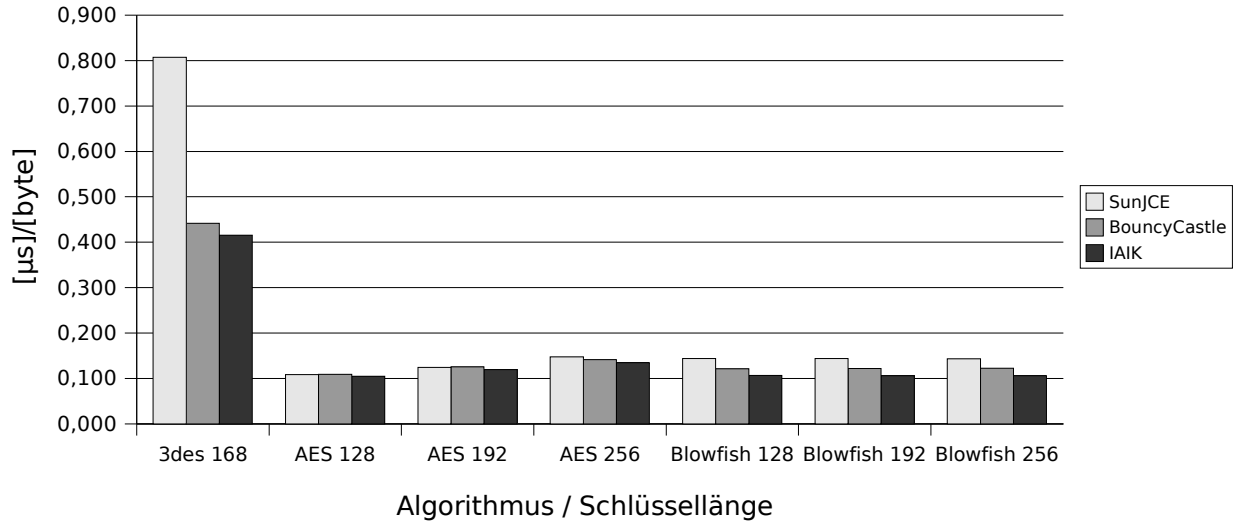



Abbildung B.2: Symmetrische Ver- und Entschlüsselungszeit

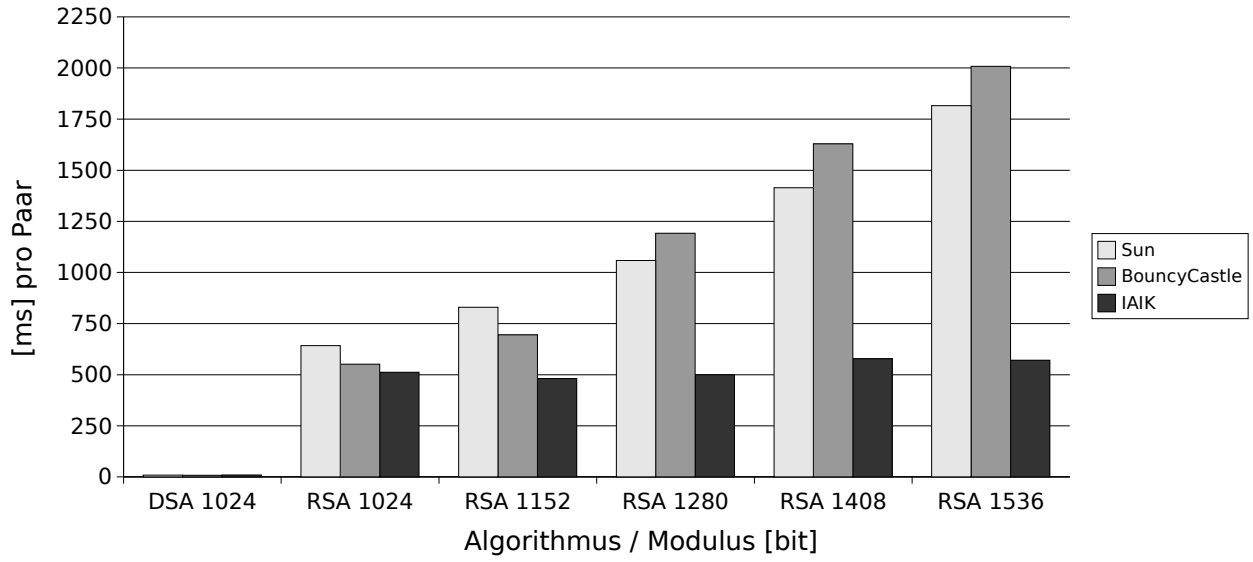



Abbildung B.3: Asymmetrische Schlüsselgenerierung

Größen experimentiert. Abbildung B.3 zeigt die aus jeweils 100 Versuchen gemittelten Ergebnisse. Die verschiedenen Provider zu vergleichen, ergibt hier wenig Sinn, da stochastische Primzahltests mit unterschiedlichen Wahrscheinlichkeiten und damit unterschiedlichen Verarbeitungszeiten verwendet werden. Wichtig ist allerdings die Größenordnung der Generierungszeit: Bei DSA liegt sie für alle Provider unter 30 ms, also sehr deutlich unter den Zeiten für RSA. In zeitkritischen oder häufig benutzten Programmabschnitten sollten daher möglichst keine RSA-Schlüsselpaare generiert werden. In der Regel benötigt man aber auch nur ein Schlüsselpaar pro Benutzer, das offline generiert werden kann.

Zu guter Letzt wurden die Signaturalgorithmen untersucht. Dabei wurden jeweils 1000 Daten-

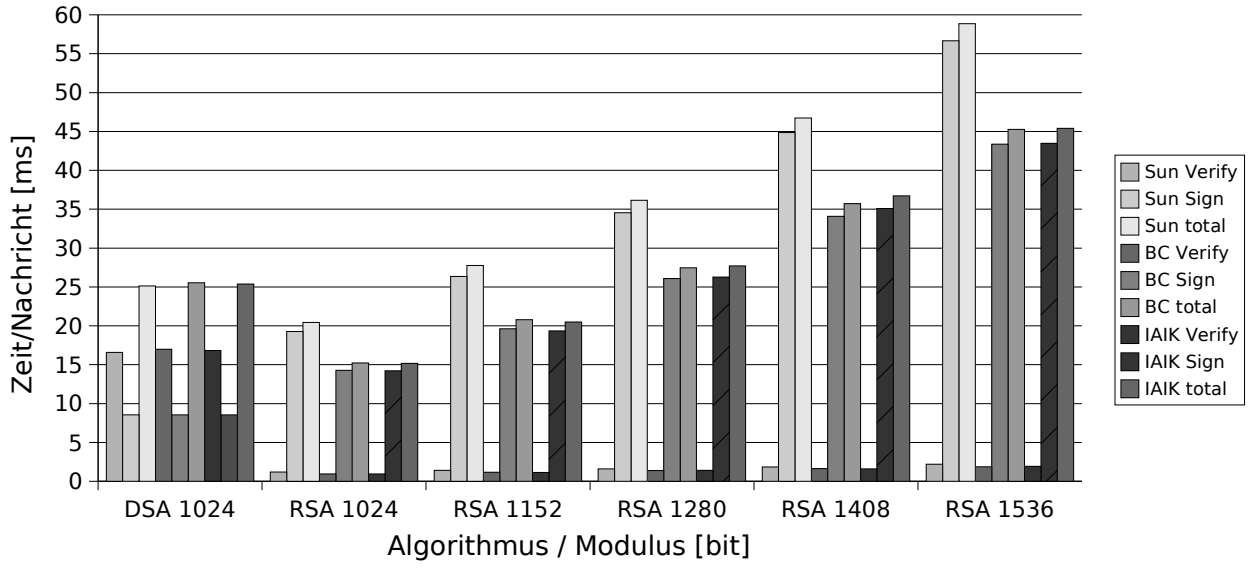



Abbildung B.4: Erzeugung und Verifizierung digitaler Signaturen

pakete von jeweils 4096 Byte signiert und die Signatur verifiziert. Als Hash-Algorithmus wurde SHA-1 eingesetzt. Größere Datenpakete bringen wegen des sehr kleinen Anteils der Hash-Operation an der gesamten Berechnungszeit nur geringfügige Veränderungen. Die Schlüssel wurden mit denselben Parametern generiert wie im vorherigen Experiment. Die Ergebnisse sind in Abbildung B.4 dargestellt. DSA hat in der Standardeinstellung ein besseres Gleichgewicht zwischen Erzeugung und Verifizierung als RSA. Das ist nicht verwunderlich, da bei RSA ein vergleichsweise kleiner öffentlicher Exponent vorgewählt wurde. Ansonsten ist DSA langsamer als RSA mit gleicher Moduluslänge. Für RSA-Operationen benötigt Sun generell etwa ein Viertel mehr Zeit als die beiden anderen Provider, die sich mit dem bloßen Auge kaum erkennbar unterscheiden.

Die auf elliptischen Kurven basierenden Algorithmen des IAIK-Providers waren übrigens deutlich langsamer und wurden daher hier nicht dargestellt.

### B.6.3 Fazit

Es ist keine Neuigkeit, dass asymmetrische Verfahren deutlich langsamer sind als symmetrische. Die obigen Tests liefern die dazugehörige Größenordnung in der Java-Implementierung. Die Authentisierung einer 4096 Byte großen Nachricht mittels HMAC-SHA1 dauert maximal $2 \cdot 0,2\text{ms} = 0,4\text{ms}$, während der Einsatz einer digitalen Signatur mit DSA-1024 etwa 25 ms und RSA-1024 mindestens 15 ms benötigt. Das ergibt einen Geschwindigkeitsfaktor von 62,5 für DSA und 37,5 für RSA. Asymmetrische Verfahren sollten also nach Möglichkeit in zeitkritischen sowie in häufig benutzten Programmteilen vermieden werden. Insbesondere die RSA-Schlüsselgenerierung dauert mindestens eine halbe Sekunde.

Auf 3DES sollte auch aus Geschwindigkeitsgründen verzichtet werden. Der maximale Durchsatz eines mit AES oder Blowfish verschlüsselten Kommunikationskanals liegt auf dem Testrechner je nach Algorithmus, Provider und Schlüssellänge zwischen 6,6 GByte/s (0,15 $\mu$s/Byte) und 10 GByte/s (0,1 $\mu$s/Byte).

Die Tests haben gezeigt, dass einige kryptographische Primitive von externen Providern effizienter implementiert worden sind. Insbesondere IAIK bietet erstaunliche Leistungsvorteile gegenüber den Standard-Providern. Leider erlaubt seine Lizenz keine generell freie Weitergabe. Wird häufig RSA eingesetzt, empfiehlt sich auch BouncyCastle.

## B.7 Testumgebung

Für alle Leistungstests wurde ein Rechner mit der folgenden Konfiguration verwendet:

- AMD Athlon (Barton) 2600 MHz
- 333MHz front side bus
- 128KB L1-Cache, 512KB L2-Cache
- 1GB RAM
- Linux Fedora Core 2
- Java SE SDK Version 1.4.2_02

## Anhang C

# Englische Fachbegriffe

Die zitierte Fachliteratur ist fast ausschließlich in englischer Sprache verfasst. Um eine schwer lesbare Mischung aus Deutsch und Englisch zu vermeiden, wurden an vielen Stellen weniger gebräuchliche deutsche Übersetzungen benutzt. Diese sind in der nachfolgenden Tabelle aufgeführt.

| Deutsch | Englisch |
|---|---|
| Beobachter-Entwurfsmuster | *observer pattern* |
| Berechtigungsnachweis | *credential* |
| durchgehende Sicherheit | *end-to-end security* |
| durchgehende Verbindungsfähigkeit | *end-to-end connectivity* |
| Interessensbereich | *area of interest* |
| kritischer Ausfallpunkt | *single point of failure* |
| Modell-Sicht-Kontrolleur-Entwurfsmuster | *model-view-controller pattern* |
| neue Nachricht | *fresh message* |
| Nicht-Abstreitbarkeit | *non-repudiation* |
| Passphrase | *passphrase* |
| perfekte Vorwärtsgeheimhaltung | *perfect forward secrecy* |
| praktisch unmöglicher Angriff | *computationally unfeasible attack* |
| Revisionsaufzeichnung | *audit log* |
| Schlüsselwechsel | *re-keying* |
| schwacher Schlüssel | *weak key* |
| Sicherheitstechnik | *security engineering* |
| Sicherheitsrichtlinie | *security policy* |
| soziale Manipulation | *social engineering* |
| Stellvertreter | *proxy* |
| Stellvertreter-Entwurfsmuster | *proxy pattern* |
| überkreuzte Zertifikate | *cross certificates* |
| Verleger-Abonnent-Entwurfsmuster | *publisher subscriber pattern* |
| Verschleierung | *obscurity* |
| Verteilte virtuelle Umgebung | *distributed virtual environment* |
| Zertifizierungsantrag | *certificate signing request* |
| Zugangsausweis | *access token* |
| Zugangspunkt | *rendezvous point* |
| zuverlässiges Zertifikat | *trusted certificate* |

# Abbildungsverzeichnis

# Lebenslauf

| | | |
|---|---|---|
| **Name** | | Jan Köhnlein |
| **Geburtsdatum** | | 26. Februar 1973 |
| **Geburtsort** | | München |
| **Schulische Ausbildung** | 1979-1980 | Nadischule (München) |
| | 1980-1983 | Mühlbachhofschule (Stuttgart) |
| | 1983-1984 | Eberhard-Ludwigs-Gymnasium (Stuttgart) |
| | 1984-1992 | Wilhelm-Gymnasium (Hamburg) |
| | 1992 | Abitur |
| **Studium** | 10/1992 - 07/1999 | Mathematikstudium an der Universität Hamburg mit Anwendungsfach Informatik |
| | 10/1996 - 03/1997 | Auslandssemester in Valladolid (Spanien) |
| | 07/1999 | Diplom |
| **Berufliche Tätigkeit** | 10/1999 - 03/2004 | Wissenschaftlicher Mitarbeiter an der TU Hamburg-Harburg. |