LEHRSTUHL FÜR BETRIEBSSYSTEME

Univ.-Prof. Dr. habil. Thomas Bemmerl

Entwurf und Implementierung einer sicheren Gruppen-Kommunikationsschicht für Peer-to-Peer Systeme

Ruben Niederhagen
Matrikelnummer: 235545

Diplomarbeit
an der
Rheinisch-Westfälischen Technischen Hochschule Aachen
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Betriebssysteme

RWTH AACHEN UNIVERSITY

Erstgutachter:	Univ.-Prof. Dr. habil. Thomas Bemmerl
Zweitgutachter:	Univ.-Prof. Dr.-Ing. Klaus Wehrle
Betreuer:	Dr. rer. nat. Stefan Lankes

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht als Prüfungsarbeit eingereicht worden.

Aachen, den 27. März 2007

Inhaltsverzeichnis

Abbildungsverzeichnis **vii**

1 Einleitung **1**

2 Netzsicherheit **5**
2.1 Benutzer und Identität 6
2.2 Angriffsszenarien 6
2.3 Sicherheitsziele 7
2.4 Kryptographische Grundlagen 8
2.4.1 Kryptosysteme 8
2.4.2 Hash-Funktionen 10
2.5 Kryptographie in der Netzsicherheit 10
2.5.1 Signaturen 10
2.5.2 Verschlüsselung 12
2.5.3 Schlüsselverteilung 14
2.5.4 Gültigkeitsdauer von Schlüsseln 17

3 Peer-to-Peer Systeme **19**
3.1 Unstrukturierte Peer-to-Peer-Systeme 20
3.1.1 Reine Peer-to-Peer-Systeme 20
3.1.2 Hybride Peer-to-Peer-Systeme 22
3.2 Strukturierte Peer-to-Peer-Systeme 22
3.2.1 Zentralisierte Peer-to-Peer-Systeme 23
3.2.2 DHT-basierte Peer-to-Peer-Systeme 23
3.3 Peer-to-Peer Programmierschnittstellen 25
3.3.1 JXTA - It's all about protocols. 26

4 Sichere Gruppenkommunikation **29**
4.1 Sichere Gruppenkommunikation in Peer-to-Peer-Systemen 29

5 Entwurf **33**
5.1 Systembeschreibung 33
5.2 Systemarchitektur 34
5.3 Kommunikationskanäle 35
5.4 Nachrichten 36
5.5 Gruppen 36
5.5.1 Gruppenrechte 37

5.5.2 Gruppenoperationen 38
5.6 Sicherheitsarchitektur 43
5.6.1 Erzeugung und Schutz der privaten Schlüssel 44
5.6.2 Verteilung öffentlicher Schlüssel 45
5.6.3 Symmetrischer Schlüssel des `ServiceChannels` 46
5.6.4 Symmetrischer Schlüssel des `GroupChannels` 47
5.6.5 Verbindlichkeit des `ServiceChannels` 49
5.6.6 Wiedereinspielung 49
5.6.7 Sicherheitsschicht 49
5.6.8 Erneuerung der symmetrischen Schlüssel 51
5.6.9 Nachrichtenformate der Sicherheitsschicht 52

6 Implementierung 55
6.1 Kryptographische Algorithmen 55
6.1.1 Symmetrische Schlüssel 55
6.1.2 Asymmetrische Schlüssel 56
6.1.3 Schlüsselbund 57
6.1.4 Diffie-Hellman-Schlüsseltausch 58
6.1.5 Hash-Algorithmus 58
6.2 Nachrichten 58
6.3 Sicherheitsschicht 62
6.3.1 Sicherheitsschicht der Peers im Detail 62
6.3.2 Sicherheitsschicht des `MembershipService` im Detail 65
6.3.3 `MessageHandler` der asynchronen Kommunikation 67
6.3.4 `MessageHandler` der synchronen Kommunikation 69
6.3.5 Synchronisierer 71
6.3.6 Einbettung der Sicherheitsschicht 72
6.4 Credential 73
6.5 JXTA Abstraktion 75
6.6 Ausführung der Gruppenoperationen 77
6.6.1 Datenbankstruktur 77
6.6.2 `MessageHandler` des `MembershipService` 78
6.7 Sicherheitshinweise 80

7 Zusammenfassung 83

Abkürzungsverzeichnis 87

Index 89

Literaturverzeichnis 93

Abbildungsverzeichnis

2.1 Nachrichtenfluss durch die Abstraktionsschichten 5

2.2 Schema der Erstellung und Verifizierung einer Signatur 11
2.3 Lage der Sicherheitsschicht im TCP/IP-Referenzmodell 12
2.4 Man-in-the-Middle-Angriff auf den Diffie-Hellman-Schlüsseltausch . . 16

3.1 Rechnernetz und Overlay-Graph 20
3.2 Unstrukturierte P2P-Systeme 21
3.3 Strukturierte P2P-Systeme 23
3.4 Beispiele für Chord-Ringe 24
3.5 JXTA-Rendezvous-Overlay und JXTA-Relay-Overlay 27

5.1 Systemarchitektur 34
5.2 Hierarchie der Gruppenrechte. 38
5.3 Gruppenoperationen und erforderliche Rechte 43
5.4 Erzeugung des Gruppen-Schlüssels 47
5.5 Kommunikationskanäle und -schichten 50

6.1 Hierarchie der abstrakten Nachrichtenklassen 58
6.2 Hierarchie der Nachrichtenformate Teil 1 60
6.3 Hierarchie der Nachrichtenformate Teil 2 61
6.4 Peer-seitige Implementierung der Sicherheitsschicht 63
6.5 `MembershipService`-Implementierung der Sicherheitsschicht 65
6.6 Komponenten eines asynchronen `MessageHandlers` 67
6.7 Verbindung der Komponenten `Inside` und `Outside` 69
6.8 Synchrone Komponenten der Sicherheitsschicht im Detail 70
6.9 `SynchronousRequester` und `SynchronousResponder` im Detail . 71
6.10 `Credential`-Klassenhierarchie 76
6.11 Datenbankstruktur 78

1 Einleitung

Im Jahr 1969 wurde das Advanced Research Projects Agency Network (ARPANET) in Betrieb genommen. Es verband die vier Forschungseinrichtungen Stanford Research Institute, University of Utah sowie die Universities of California in Los Angeles und Santa Barbara. Viele Netzdienste, die sich zu Beginn der 70er Jahre im ARPANET etablierten, finden auch heute noch Anwendung - wie beispielsweise eMail, das File Transfer Protocol und Telnet.

In den 1980er Jahren wurde das bis dahin reine Forschungsnetz auch für den gewerblichen und den privaten Gebrauch geöffnet; die Bezeichnung Internet etablierte sich. Mit der Einführung des World Wide Web (WWW) im Jahr 1989 wurde das Internet auch für die breite Öffentlichkeit interessant; seit den 1990er Jahren wächst die Zahl der Internet-Präsenzen und Web-Seiten und damit das Volumen an Web-Diensten und Informationen exponentiell.

Während das ARPANET aus einer überschaubaren Anzahl von gleichberechtigten Computern bestand, entwickelte sich das Internet in den 1990er Jahren durch die wachsende Zahl von Nutzern und die Einführung zentralisierter Dienste zu einem Client-Server-System, in dem eine vergleichsweise geringe Zahl von Servern eine Vielzahl von Clients mit Informationen versorgt.

Seit dem Beginn des 21. Jahrhunderts ist aber einhergehend mit fortschrittlichen Technologien eine neue Entwicklung im Verhalten der Benutzer des Internets feststellbar:

- Die Verbreitung von Heim-Computern und die private Nutzung des Internets nimmt zu.
- Die Rechenleistung von Heim- und Bürorechnern wächst stetig.
- Immer mehr Haushalte sind mit Breitband-Verbindungen an das Internet angeschlossen.

In Folge dessen treten die Endbenutzer nicht mehr als bloße Konsumenten von Informationen oder Internet-Dienstleistungen in Erscheinung, sondern beginnen aktiv, den Inhalt des Internets mitzugestalten. Beispiele sind Web-Angebote wie Wikipedia[1], eBay[2], YouTube[3], alle Arten von Blogs und Newsgroups, aber auch Filesharing-

[1] `www.wikipedia.org`
[2] `www.ebay.com`
[3] `www.youtube.com`

Anwendungen wie eMule oder BitTorrent[4].

Diese Entwicklung bleibt nicht auf den privaten Konsum beschränkt, Forschungsprojekte wie SETI@home[5] oder ZetaGrid[6] zeigen, dass in Büros und Privaträumen eine gewaltige Rechenleistung schlummert, die wirtschaftlich oder gemeinschaftlich genutzt werden kann.

Die am Internet beteiligten Rechner können somit nicht mehr streng als Server- oder Client-Rechner kategorisiert werden, sondern treten mehr und mehr einander gleichberechtigt auf. Das Internet wandelt sich von einem Client-Server-Netz zu einem Netz Gleichgestellter; ein solches Netz wird in der Literatur als Peer-to-Peer (P2P)-Netz (engl. *peer*: Gleichgestellter) bezeichnet.

Auch wenn bereits seit einigen Jahren wissenschaftlich an P2P-Systemen geforscht wird und P2P-Anwendungen wie eMule und BitTorrent von der Internet-Gemeinschaft verwendet werden, steht die kommerzielle Nutzung dieses neuen Paradigmas noch an ihrem Anfang - eine junge Anwendung ist beispielsweise Skype[7].

Der kommerziellen Nutzung von P2P-Systemen steht entgegen, dass den P2P-Anwendungen ein Image des Illegalen anhaftet. Die ersten populären P2P-Anwendungen dienten der meist nicht legalen Verbreitung von urheberrechtlich geschützten Medien wie Musikstücken oder Filmen. Unterstützt wurde dies durch die weitgehende Anonymität, die das Internet dem Benutzer bietet.

Die kommerzielle Nutzung von P2P-Anwendungen setzt dagegen eine Authentifizierung der Benutzer voraus, so dass angebotene Dienste nur von autorisierten (und zahlenden) Benutzern in Anspruch genommen werden können.

Die Firma SYNCING.NET[8] entwickelt eine P2P-Anwendung, die es einer begrenzten, autorisierten Gruppe von Benutzern ermöglichen soll, Kalenderdaten und andere Informationen miteinander zu teilen. Da es sich bei diesen Informationen auch um sensible Daten wie Geschäftsgeheimnisse handeln kann, muss für eine vertrauliche Übertragung der Daten gesorgt werden.

Dafür wird eine plattformunabhängige Bibliothek benötigt, die folgende Funktionen anbietet:

- **Gruppenkommunikation:** Benutzer können Nachrichten an die Mitbenutzer ihrer Gruppe senden.
- **Vertraulichkeit:** Nur berechtigte Benutzer haben Zugriff auf die Nachrichten.
- **Authentifizierung:** Die Benutzer müssen ihre Identität nachweisen, um an der Gruppenkommunikation teilzunehmen.

[4] `www.bittorrent.com`
[5] `www.setiathome.berkeley.edu`
[6] `www.zetagrid.net`
[7] `www.skype.com`
[8] `www.syncing.net`

- **Zugriffskontrolle:** Der Zugang zur Gruppe ist auf berechtigte Benutzer eingeschränkt.
- **Gruppenverwaltung:** Berechtigte Benutzer können folgende Verwaltungsoperationen in der Gruppe durchführen:
  - Benutzer in eine Gruppe einladen
  - Mitglieder aus einer Gruppe ausschließen
  - neue Gruppen erstellen
  - Gruppen auflösen
- **Rechteverwaltung:** Den Benutzern werden feingranular Rechte zum Durchführen dieser Verwaltungsoperationen zugewiesen.

In dieser Diplomarbeit wird die Programmbibliothek „Secure Communication Group“ (SCG) entwickelt und implementiert, die flexibel auf der P2P-Programmbibliothek JXTA aufbauend diese Anforderungen erfüllt. Die Bibliothek ist für die .NET-Laufzeitumgebung geschrieben und damit auf verschiedenen Plattformen einsetzbar.

Gliederung

In den folgenden drei Kapiteln werden die im Rahmen dieser Diplomarbeit verwendeten Begriffe und Mechanismen vorgestellt und definiert. Kapitel 2 geht hierbei auf Netzsicherheit und kryptographische Grundlagen ein, Kapitel 3 führt in P2P-Systeme ein und Kapitel 4 stellt aktuelle Forschungsarbeiten im Bereich der sicheren Gruppenkommunikation vor.

In Kapitel 5 wird der Entwurf der im Rahmen dieser Diplomarbeit entwickelten Programmbibliothek Secure Communication Group (SCG) vorgestellt und in Kapitel 6 näher auf ihre Implementierung eingegangen.

Kapitel 7 enthält eine Zusammenfassung der vorliegenden Diplomarbeit.

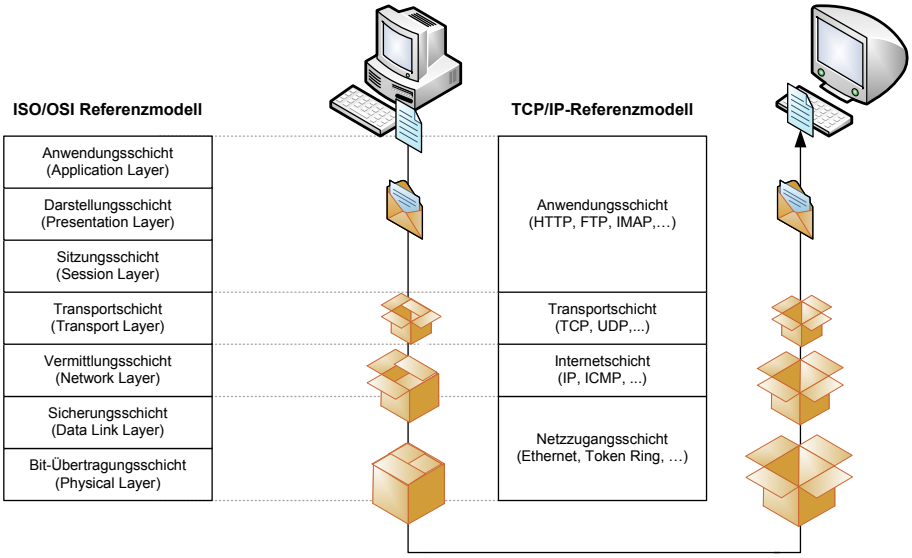


Abbildung 2.1: Nachrichtenfluss durch die Abstraktionsschichten

2 Netzsicherheit

Als **Kommunikation** wird das Austauschen von strukturierten Datenpaketen, so genannten **Nachrichten**, zwischen zwei **Instanzen** in einem Rechnernetz bezeichnet. Die Nachrichten werden von einer Instanz, dem **Absender** zu einer oder mehreren Instanzen, den **Empfängern** übertragen.

Üblicherweise durchlaufen die Nachrichten dabei verschiedene **Abstraktionsschichten**, die sich am ISO-OSI-Referenzmodell orientieren. Jede Nachricht einer höheren Schicht wird beim Senden in eine Nachricht der darauf folgenden Schicht verpackt und beim Empfänger Schicht für Schicht wieder entpackt. Abbildung 2.1 illustriert diesen Vorgang.

Zwischen gleichen Schichten auf Sender- und Empfängerseite besteht eine logische **Kommunikationsverbindung** oder kurz **Verbindung**. Jede Schicht besitzt ein **Protokoll**, welches die Datenstruktur der Nachricht sowie die Regeln der Kommunikation zwischen Sender und Empfänger auf dieser Schicht definiert.

Der Weg, den eine Nachricht durch die Schichten und über das Rechnernetz zwischen Absender und Empfänger beschreitet, wird als **Nachrichtenkanal** bezeichnet.

Eine **Punkt-zu-Punkt**- oder **Unicast-Verbindung** ist eine Kommunikationsverbindung zwischen genau zwei Instanzen. Sendet bei einer Unicast-Verbindung genau

einer der Kommunikationsteilnehmer, während der andere nur empfängt, handelt es sich um eine **unidirektionale** Unicast-Verbindung; treten beide Teilnehmer sowohl als Absender wie als Empfänger auf, ist die Verbindung **bidirektional**.

Sind mehr als zwei Instanzen an der Kommunikation beteiligt, wird ihre Kommunikationsverbindung als **Mehrpunktverbindung** oder **Multicast-Verbindung** bezeichnet. Die Kommunikationsteilnehmer bilden eine **Gruppe**, jedes Mitglied der Gruppe kann Nachrichten senden, die von allen Mitgliedern empfangen werden.

Wenn ein Sender pausiert, bis er eine Antwort des Empfängers erhalten hat, wird die Kommunikation als **synchron** bezeichnet. Ist das Sendern und Empfangen von Nachrichten unabhängig voneinander, so ist die Kommunikation **asynchron**.

2.1 Benutzer und Identität

Ein **Benutzer** (engl.: *user*) ist eine natürliche Person, die ein System verwendet. Er wird durch eine Instanz im System repräsentiert.

Jede Instanz in einem System besitzt mindestens eine **Identität**. An die Identität sind die Rechte gebunden, die der zugehörigen Instanz unter dieser Identität eingeräumt werden.

Die Überprüfung der Identität einer Instanz wird als **Authentifizierung** (engl.: *authentication*) bezeichnet; eine **Authentisierung** ist der Vorgang des Nachweises der eigenen Identität.

Nach erfolgreicher Authentifizierung ist eine Instanz **autorisiert**, im Rahmen ihrer Rechte Operationen im System durchzuführen. Die Überprüfung und Zuweisung von Rechten wird als **Autorisierung** (engl.: *authorization*) bezeichnet.

2.2 Angriffsszenarien

In einem Rechnernetz wie dem Internet ist eine Vielzahl von Angriffen auf die Kommunikation zwischen Rechnersystemen und auf die Rechner selbst möglich. Im Folgenden werden die wichtigsten Angriffszenarien erläutert [4]:

- Durch das **Abhören** einer Kommunikationsverbindung kann ein Angreifer in den Besitz sensibler Daten wie Kontonummern oder Vertragsgeheimnisse gelangen. Ein Abhören von Nachrichten ist beispielsweise in unverschlüsselten Funknetzen besonders einfach.
- Bei einer **Man-in-the-Middle-Attacke** leitet der Angreifer den Kommunikationskanal zwischen den Instanzen so um, dass er die volle Kontrolle über die übertragenen Daten hat und Informationen abhören, aber auch einfügen, löschen oder verändern kann.

- Das Vortäuschen einer fremden Identität wird als **Maskerade** bezeichnet. Ein Angreifer kann sich beispielsweise als Webserver einer Bank ausgeben und auf diese Weise an sensible Kundendaten gelangen.
- Bei **Replay-Attacken** (engl. für: Wiedereinspielung) werden Nachrichten zu einem späteren Zeitpunkt ein weiteres Mal eingespielt. Wird beispielsweise die Transaktions-Nachricht einer Bank-Überweisung mehrfach wieder eingespielt, kann sich ein Angreifer unbemerkt bereichern.
- **Denial-Of-Service-Attacken** (DoS) zielen darauf ab, einen von einem Rechner angebotenen Dienst unerreichbar zu machen, indem der Rechner mit Anfragen überschwemmt und dadurch überlastet wird.

Angriffe auf Rechnernetze müssen nicht einmal über das Netz selbst erfolgen. Oft ist es möglich, unter Anwendung **physischer Gewalt** (wie das Eindringen in ein Rechenzentrum), Zugriff auf Information zu erlangen oder mit Hilfe von **Social Engineering** (z.B. durch einen freundlichen Anruf und höfliches Nachfragen) in den Besitz von Passwörtern zu gelangen. Des Weiteren können **Implementierungsfehler** von Protokollen oder Anwendungen ausgenutzt werden, um Zugriff auf gesamte Rechnersysteme zu erhalten.

Diese Arbeit wird sich auf die Sicherung der Kommunikation konzentrieren und nicht auf weiter reichende Sicherheitsaspekte eingehen.

2.3 Sicherheitsziele

Im Rahmen dieser Arbeit werden für Kommunikationsverbindungen folgende Sicherheitsziele betrachtet [4]:

Vertraulichkeit (engl.: *confidentiality*): Der Inhalt einer Nachricht darf nur autorisierten Instanzen zugänglich sein.

Integrität (engl.: *integrity*): Eine Veränderung von Nachrichten während der Übertragung muss vom Empfänger bemerkt werden.

Authentizität (engl.: *authenticity*): Der Absender einer empfangenen Nachricht muss zweifelsfrei feststellbar sein. Authentizität setzt die Integrität der Nachricht sowie die Authentifizierung des Absenders voraus.

Verbindlichkeit (engl.: *non-repudiability*): Der Absender einer Nachricht darf zu einem späteren Zeitpunkt nicht bestreiten können, die Nachricht gesendet zu haben. Die Verbindlichkeit setzt die Authentizität der Nachricht voraus.

2.4 Kryptographische Grundlagen

Um die in Abschnitt 2.3 genannten Sicherheitsziele zu erreichen, werden in der Netzsicherheit Methoden aus der Kryptographie verwendet. In den folgenden beiden Abschnitten werden einige kryptographische Grundlagen vorgestellt.

2.4.1 Kryptosysteme

Kryptosysteme können dazu eingesetzt werden, sensible Nachrichten oder Daten zu verschlüsseln. Um Verschlüsselung formal definieren zu können, werden im Folgenden einige Termini eingeführt; die Definitionen orientieren sich am *Handbook of Applied Cryptography* [10].

Definition 2.1: Alphabet und Wort
*Ein **Alphabet** $\mathcal{A}$ ist eine endliche Menge von Symbolen.*

Beispiel 2.1.1: $\mathcal{A}_{binary} = \{1, 0\}$, $\mathcal{A}_{alphabet} = \{A...Z\}$

*Ein **Wort** w über einem Alphabet $\mathcal{A}$ ist eine endliche Folge von Symbolen aus $\mathcal{A}$, $|w|$ bezeichnet die Länge, d.h. die Anzahl der Symbole des Wortes w.*

Beispiel 2.1.2: *Ein Satz der deutschen Sprache kann als Wort über einem Alphabet $\mathcal{A}_{de} = \{a...z, A...Z, ä, ö, ü, Ä, Ö, Ü, ß, ␣, [,], [.], [:], [;]\}$ betrachtet werden; ␣ bezeichnet hierbei das Leerzeichen.*

Definition 2.2: Klartextraum, Geheimtextraum und Schlüsselraum
*Ein **Klartextraum** $\mathcal{M}$ (engl.: message space) ist eine Menge von Wörtern über einem Alphabet $\mathcal{A}_{\mathcal{M}}$. Ein Element von $\mathcal{M}$ wird **Klartext** (eng.: plaintext) oder **Nachricht** genannt.*

*Ebenso ist ein **Geheimtextraum** $\mathcal{C}$ (engl.: cipher space) eine Menge von Wörtern über einem Alphabet $\mathcal{A}_{\mathcal{C}}$; die Alphabete $\mathcal{A}_{\mathcal{M}}$ und $\mathcal{A}_{\mathcal{C}}$ können identisch sein. Ein Element von $\mathcal{C}$ wird **Geheimtext** (eng.: ciphertext) oder **Chiffre** genannt.*

*$\mathcal{K}$ bezeichnet eine Menge mit dem Namen **Schlüsselraum** (engl.: key space); die Elemente von $\mathcal{K}$ heißen **Schlüssel**.*

Definition 2.3: Verschlüsselung und Entschlüsselung
*Jedes $e \in \mathcal{K}$ definiert eindeutig eine bijektive Abbildung $E_e : \mathcal{M} \rightarrow \mathcal{C}$, die einen Klartext $m \in \mathcal{M}$ auf einen Geheimtext $c \in \mathcal{C}$ abbildet. E_e wird **Verschlüsselungsfunktion** oder **-transformation** genannt.*

*Das Anwenden der Transformation E_e auf einen Klartext wird als **Verschlüsselung** oder Chiffrierung (engl.:* encryption*) des Klartextes bezeichnet.*

*Für jedes $d \in \mathcal{K}$ bezeichnet $D_d : \mathcal{C} \rightarrow \mathcal{M}$ eine bijektive Abbildung vom Geheimtextraum in den Klartextraum. D_d wird **Entschlüsselungsfunktion** oder **-transformation** genannt.*

Das Anwenden der Transformation D_d auf einen Geheimtext wird als ***Entschlüsselung*** *oder Dechiffrierung (engl.:* decryption*) des Geheimtextes bezeichnet.*

Definition 2.4: Kryptosystem
Ein ***Kryptosystem*** *besteht aus einer Menge $\{E_e : e \in \mathcal{K}\}$ von Verschlüsselungsfunktionen und einer Menge $\{D_d : d \in \mathcal{K}\}$ von Entschlüsselungsfunktionen mit der Eigenschaft, dass für jeden Schlüssel $e \in \mathcal{K}$ ein eindeutiger Schlüssel $d \in \mathcal{K}$ existiert, so dass gilt: $D_d = {E_e}^{-1}$. Das heißt, es gilt $D_d(E_e(m)) = m$ für alle $m \in \mathcal{M}$.*

Ein Kryptosystem ermöglicht es also, einen Klartext mit einem Schlüssel e zu einer Chiffre zu verschlüsseln, so dass diese Chiffre mit einem eindeutigen Schlüssel d wieder zu dem ursprünglichen Klartext entschlüsselt werden kann.

Es werden zwei Arten von Kryptosystemen unterschieden: symmetrische sowie asymmetrische Kryptosysteme.

Definition 2.5: Symmetrische und asymmetrische Kryptosysteme
Bei ***symmetrischen Kryptosystemen*** *sind die Schlüssel e und d identisch, ein einziger Schlüssel dient sowohl der Ver- wie auch der Entschlüsselung einer Nachricht.*

Bei ***asymmetrischen Kryptosystemen*** *wird ein* ***Schlüsselpaar*** *von unterschiedlichen Schlüsseln e und d verwendet.*

Public-Key-Kryptosysteme sind ein Sonderfall der asymmetrischen Kryptosysteme:

Definition 2.6: Public-Key-Kryptosystem
Ein ***Public-Key-Kryptosystem*** *(public key engl. für: öffentlicher Schlüssel) ist ein asymmetrisches Kryptosystem, dessen Klartext- und Geheimtextraum identisch sind.*

Bei einem Public-Key-Kryptosystem ist jedes Schlüsselpaar an eine Identität gebunden. Der Dechiffrierungsschlüssel ist nur der Instanz bekannt, die diese Identität besitzt, und wird geheim gehalten. Dieser Schlüssel wird auch als ***privater*** *oder* ***geheimer*** *Schlüssel bezeichnet. Der Chiffrierungs-Schlüssel ist allgemein bekannt, er wird daher auch* ***öffentlicher*** *Schlüssel (engl.: public key) genannt.*

Da Klartext- und Geheimtextraum identisch sind, ist eine Verschlüsselung sowohl mit dem öffentlichen als auch mit dem privaten Schlüssel möglich. Für die Entschlüsselung muss jeweils der andere Schlüssel verwendet werden.

Zu beachten ist, dass Kryptosysteme im Allgemeinen nicht perfekt sicher sind (einzige Ausnahme: One-Time-Pads [10]). Beispielsweise sind viele mathematische Kryptosystem durch einen so genannten Brute-Force-Angriff (*brute*: brutal; *force*: Kraft) kompromittierbar. Bei diesem Angriff wird systematisch jeder mögliche Schlüssel durchprobiert, bis der korrekte Schlüssel gefunden wurde.

Kryptosysteme werden daher anwendungsspezifisch so ausgewählt, dass sie eine „hinreichende“ Sicherheit bieten. Angriffe auf ein Kryptosystem sind mit Kosten ver-

bunden (z.B.: Zeit, Hardware, Rechenpower, Bereitschaft zur Anwendung physischer Gewalt...). Wenn die Kosten für den Angriff größer sind als der Wert des Klartextes, ist davon auszugehen, dass der Klartext durch das Kryptosystem hinreichend gesichert ist.

2.4.2 Hash-Funktionen

Neben den Kryptosystemen werden im Bereich der Netzsicherheit mathematische Einwegfunktionen, so genannte **Hash-Funktionen** verwendet.

Definition 2.7: Hash-Funktion
Sei $\mathcal{A}$ ein Alphabet und $\mathcal{S}$ die Menge aller Wörter über $\mathcal{A}$. Ferner sei $\mathcal{S}_n = \{w \in \mathcal{S} \mid |w| = n\}, n \in \mathbb{N}$ die Menge aller Wörter über $\mathcal{A}$ der Länge n.

*Eine **Hash-Funktion** ist eine Abbildung $h : \mathcal{S} \to \mathcal{S}_n, n \in \mathbb{N}$, die Wörter beliebiger Länge auf Wörter der Länge n abbildet.*

*Für ein gegebenes Wort $m \in \mathcal{S}$ nennt man $h(m)$ den **Hash-Wert** von m.*

Da die Zielmenge der Hash-Funktion endlich ist, die Definitionsmenge dagegen unendlich, ist eine Hash-Funktion nicht injektiv. Wenn mehrere Elemente der Definitionsmenge auf dasselbe Element der Zielmenge abgebildet werden, nennt man dies **Kollision**.

Kryptographische Hash-Funktionen werden so definiert, dass es in der Praxis weder möglich ist, zwei kollidierende Eingabewerte zu bestimmen (**Kollisionsresistenz**), noch zu einem gegebenen Hash-Wert $y \in \mathcal{S}_n$ einen zugehörigen Eingabewert $x \in \mathcal{S}$ zu finden, so dass gilt: $h(x) = y$ (**Urbildresistenz**).

2.5 Kryptographie in der Netzsicherheit

Kryptographie wird in der Netzsicherheit dazu eingesetzt, Kommunikationspartner zu authentifizieren und die Sicherheitsziele für Kommunikationsverbindungen zu erreichen.

2.5.1 Signaturen

Um die Authentizität, Verbindlichkeit und Integrität einer Kommunikationsverbindung zwischen zwei Kommunikationsinstanzen *Alice* und *Bob*[1] zu gewährleisten,

[1] Alice und Bob sind in der Kryptographie häufig verwendete Synonyme für Kommunikationspartner.

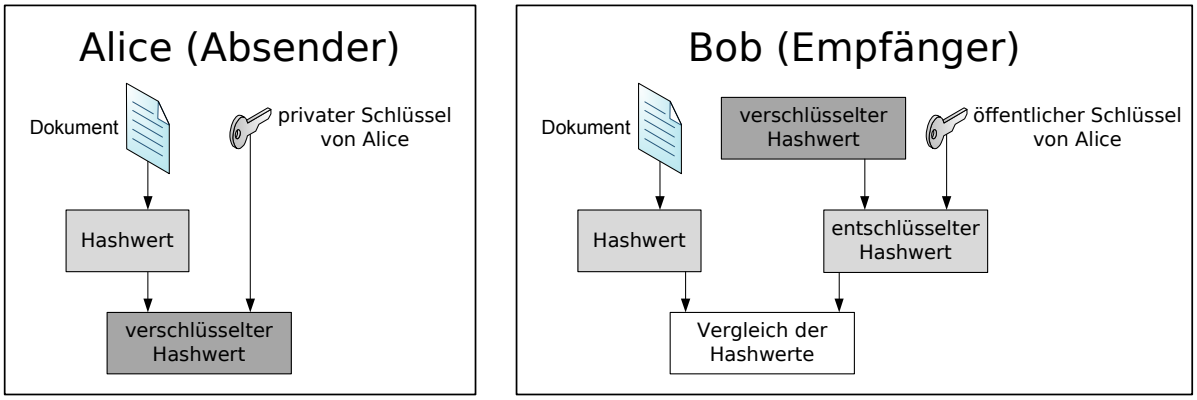


Abbildung 2.2: Schema der Erstellung und Verifizierung einer Signatur

können **Signaturen** eingesetzt werden. Zum Signieren von Nachrichten wird ein Public-Key-Kryptosystem benötigt.

Abbildung 2.2 zeigt das Schema der Erstellung und Verifizierung einer Signatur. Zunächst berechnet die Absenderin Alice mit Hilfe einer Hash-Funktion einen Hash-Wert für die Nachricht. Dieser Hash-Wert wird **Fingerprint** (Fingerabdruck) genannt. Den Fingerprint verschlüsselt sie mit ihrem privaten Schlüssel; dadurch erhält Alice eine **Signatur** der Nachricht. Nun wird die Signatur zusammen mit der Nachricht an den Empfänger Bob gesendet.

Bob berechnet seinerseits mit der Hash-Funktion den Fingerprint der Nachricht. Mit Hilfe des öffentlichen Schlüssels von Alice kann er nun die Signatur entschlüsseln und den berechneten Fingerprint mit dem entschlüsselten vergleichen.

Integrität, Authentizität und Verbindlichkeit der Nachricht können nun folgendermaßen verifiziert werden:

Integrität: Wenn die beiden Hash-Werte übereinstimmen, ist sichergestellt, dass die Nachricht während der Übertragung nicht verändert wurde.

Authentizität: Da nur Alice mit ihrem privaten Schlüssel den Hash-Wert verschlüsselt und damit die Signatur erstellt haben kann, ist nachgewiesen, dass die Nachricht von ihr stammt.

Verbindlichkeit: Aufgrund der Signatur kann Alice auch nicht abstreiten, die Nachricht gesendet zu haben. Um die Übertragung der Nachricht zu einem späteren Zeitpunkt nachweisen zu können, müssen Alice und Bob die Nachricht zusammen mit der Signatur so lange speichern, wie ein Nachweis erwünscht ist. Dazu kann ein Logging-Mechanismus verwendet werden, indem ein Log-Eintrag mit dem Namen des Absenders, dem Inhalt der Nachricht und der Signatur erstellt wird. Je nach verwendetem Logging-Mechanismus werden die Log-Einträge beispielsweise in einer Datenbank oder einer Datei gespeichert.

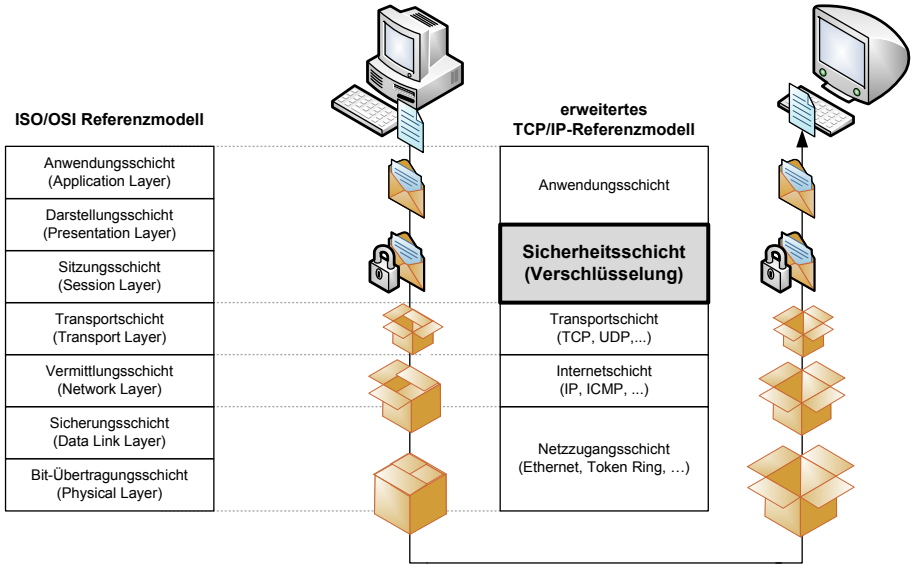


Abbildung 2.3: Lage der Sicherheitsschicht im TCP/IP-Referenzmodell

Authentifizierung

Signaturen können auch zur Authentifizierung eines Kommunikationspartners eingesetzt werden:

Soll Alice sich gegenüber Bob authentisieren, signiert sie einen zuvor abgesprochenen Klartext mit ihrem privaten Schlüssel und sendet die Signatur an Bob.

Bob entschlüsselt die Signatur mit dem öffentlichen Schlüssel von Alice. Wenn der entschlüsselte Klartext dem zuvor abgesprochenen Klartext entspricht, ist nachgewiesen, dass Alice im Besitzt des zu dem öffentlichen Schlüssel gehörenden privaten Schlüssel ist. Wenn davon ausgegangen werden kann, der dieser Schlüssel nur Alice bekannt ist, ist Alice authentifiziert.

2.5.2 Verschlüsselung

Die Vertraulichkeit einer Kommunikationsverbindung kann mit kryptographischen Mitteln erreicht werden, indem mit einem geeigneten Kryptosystem alle Nachrichten vor der Übertragung verschlüsselt und nach der Übertragung wieder entschlüsselt werden.

Zu diesem Zweck wird als weitere Netz-Abstraktionsschicht die **Sicherheitsschicht** eingeführt. Eine Nachricht, die von der nächst höheren Schicht an die Sicherheitsschicht weitergereicht wird, wird verschlüsselt und gemäß einem Protokoll mit Metainformationen versehen in eine Nachricht der Sicherheitsschicht eingebettet. Auf der Gegenseite wird die Nachricht entschlüsselt und an höhere Schichten weitergeleitet (siehe Abbildung 2.3).

Die Vertraulichkeit der Kommunikation ist gewährleistet, so lange die verwendeten Schlüssel nur den beteiligten Instanzen bekannt sind.

Für die Verschlüsselung der Kommunikationsverbindung können sowohl symmetrische als auch asymmetrische Kryptosysteme eingesetzt werden.

Symmetrische Verschlüsselung

- Zur symmetrischen Verschlüsselung einer *unidirektionalen Unicast-Verbindung* wird ein Schlüssel s verwendet, der beiden Teilnehmern bekannt ist. Der Absender verschlüsselt den Klartext mit dem Schlüssel s und überträgt die Chiffre an den Empfänger. Der Empfänger kann die Chiffre entschlüsseln und den Klartext wieder herstellen.
- Für die symmetrische Verschlüsselung einer *bidirektionalen Unicast-Verbindung* gibt es zwei Möglichkeiten:
  1. Die bidirektionale Verbindung wird wie zwei diametrale unidirektionale Verbindungen behandelt, für jede der beiden Verbindungen wird ein eigener Schlüssel verwendet.
  2. Es wird ein Schlüssel für beide Richtungen eingesetzt, alle Nachrichten werden mit demselben Schlüssel ver- und entschlüsselt.
- Bei *Multicast-Verbindungen* wird üblicherweise ein gemeinsamer, allen Mitgliedern der Multicast-Gruppe bekannter Schlüssel verwendet.

Asymmetrische Verschlüsselung

Für asymmetrische Verschlüsselung wird ein Public-Key-Kryptosystem verwendet.

- Zur Verschlüsselung einer *unidirektionalen Unicast-Verbindung* verschlüsselt der Absender die Nachricht mit dem öffentlichen Schlüssel des Empfängers; nur der Empfänger ist in der Lage, die Nachricht mit seinem privaten Schlüssel zu entschlüsseln.
- Eine *bidirektionale Unicast-Verbindung* wird wie zwei diametrale unidirektionale Verbindungen behandelt.
- *Multicast-Verbindungen* werden für gewöhnlich nicht mit dem Public-Key Verfahren verschlüsselt, da der Klartext für n Kommunikationsteilnehmer n-mal verschlüsselt werden müsste; der Berechnungsaufwand und der Umfang der zu übertragenden Daten würde unverhältnismäßig wachsen. In diesem Fall ist es effizienter, ein einzelnes Schlüsselpaar zu verwenden, welches allen Kommunikationsinstanzen bekannt ist. Das Schlüsselpaar ist dann nicht an eine einzelne Instanz, sonder an die Kommunikationsgruppe gebunden.

Da der Rechenaufwand der Ver- und Entschlüsselungsfunktionen bei asymmetrischen Verfahren deutlich höher ist als bei symmetrischen Verfahren, werden Datenkanäle, die dem Austausch umfangreicher Nachrichten dienen, üblicherweise mit einem symmetrischen Kryptosystem gesichert.

Salz

Wenn während der Kommunikation mehrfach derselbe Klartext mit demselben Schlüssel verschlüsselt wird, ist die Chiffre immer identisch. Ein Angreifer könnte daraus Rückschlüsse auf die Kommunikation ziehen. Um dem entgegen zu wirken, muss dafür gesorgt werden, dass identische Klartexte bei jeder Verschlüsselung zu unterschiedlichen Chiffren führen.

Dies wird erreicht, indem dem Klartext vor dem Verschlüsseln ein jedesmal wechselnder Zufallswert, ein so genanntes **Salz** (engl.: *salt*), vorangestellt wird. Die resultierenden Chiffren unterscheiden sich damit in jedem Fall auch bei identischem Klartext. Beim Entschlüsseln wird dieses Salz schlicht verworfen.

2.5.3 Schlüsselverteilung

Die Sicherheit eines verschlüsselten Kommunikationskanales und die Vertrauenswürdigkeit einer Signatur dürfen nicht höher eingeschätzt werden als die Sicherheit und Vertrauenswürdigkeit des Kanals, auf dem die zugehörigen Schlüssel ausgetauscht wurden.

Asymmetrische Schlüssel

Die öffentlichen Schlüssel eines Public-Key-Kryptosystems dienen meist der Authentifizierung der Kommunikationsinstanzen. Somit müssen sie auf einem vertrauenswürdigen Kanal ausgetauscht werden. Bei Schlüsseln, die Personen authentifizieren sollen, wird daher empfohlen, die Schlüssel persönlich unter Vorlage eines Lichtbildausweises auszutauschen.

Ist dies impraktikabel oder sollen nicht Personen, sondern Softwarekomponenten authentifiziert werden, können Vertrauenshierarchien von **Zertifikaten** eingesetzt werden; eine vertrauenswürdige Instanz zertifiziert dabei öffentliche Schlüssel einer niedrigeren Instanz. Über das Vertrauen in die höhere Instanz kann dann Vertrauen in eine zertifizierte Instanz hergeleitet werden.

Symmetrische Schlüssel

Symmetrische Schlüssel dienen der Verschlüsselung eines Kommunikationskanales. Sie werden bei Bedarf vereinbart und häufig gewechselt.

Symmetrische Schlüssel können übertragen werden, indem sie mit den öffentlichen Schlüsseln eines Public-Key-Kryptosystems chiffriert werden. Bei dieser Methode gibt es jedoch einen Nachteil: die Vertrauenswürdigkeit der symmetrischen Schlüssel ist an die Sicherheit der asymmetrischen Schlüssel gebunden. Wenn der private Schlüssel einer Instanz kompromittiert wird, kann der symmetrische Kommunikationsschlüssel entschlüsselt werden. Damit sind auch alle Informationen offengelegt, die mit den symmetrischen Schlüsseln chiffriert wurden.

Von symmetrisch verschlüsselten Verbindungen wird daher häufig gefordert, dass ihre Schlüssel von langlebigen, asymmetrischen Schlüsseln unabhängig sind; dies wird als **Perfect Forward Secrecy** (engl. für: perfekt fortgesetzte Geheimhaltung) bezeichnet. Im folgenden Abschnitt wird ein Verfahren vorgestellt, das die Eigenschaft der Perfect Forward Secrecy besitzt.

Diffie-Hellman-Schlüsseltausch

Der **Diffie-Hellman-Schlüsseltausch (DHS)** ermöglicht es, über eine ungesicherte Verbindung geheime Schlüssel auszutauschen. Das Verfahren wurde 1976 von Whitfield Diffie und Martin Hellman entwickelt [6].

Für die Erklärung des Diffie-Hellman-Schlüsseltausch werden folgende Definitionen benötigt:

Definition 2.8: Multiplikative Gruppe, Primitivwurzel
Sei $n \in \mathbb{N}$.

$\mathbb{Z}_n = \{\{\ldots,-n,0,n,2n,\ldots\},\{\ldots,-n+1,1,n+1,2n+1,\ldots\},\ldots,\{\ldots,-1,n-1,2n-1,\ldots\}\}$ *ist die Menge der Äquivalenzklassen von $\mathbb{Z} \bmod n$. Im Folgenden werden als Kurzschreibweise die Äquivalenzklassen mit dem betragsmäßig jeweils kleinsten nichtnegativen Repräsentant der Klasse identifiziert, also $\mathbb{Z}_n = \{0,1,2,\ldots,n-1\}$. In $\mathbb{Z}_n$ sind Addition und Multiplikation wie üblich (modulo n) definiert.*

*Die **multiplikative Gruppe** von $\mathbb{Z}_n$ ist $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \exists b \in \mathbb{Z}_n,\ ab \equiv ba \equiv 1 \pmod{n}\}$; falls n eine Primzahl ist, gilt: $\mathbb{Z}_n^* = \{a \in \mathbb{Z} \mid 1 \leq a \leq n-1\}$.*

*Eine **Primitivwurzel** der multiplikativen Gruppe $\mathbb{Z}_n^*$ ist ein $\alpha \in \mathbb{Z}_n^*$, für das gilt: $\mathbb{Z}_n^* = \{\alpha^i \bmod n \mid i \in \mathbb{Z}\}$.*

$\mathbb{Z}_n^*$ besitzt eine Primitivwurzel genau dann, wenn $n = 2, 4, p^k$ oder $2p^k$, wobei $p \neq 2$ eine Primzahl ist und $k \geq 1$ [10].

Im ersten Schritt des DHS einigen sich die Kommunikationspartner Alice und Bob auf eine große Primzahl p und eine Primitivwurzel g von $\mathbb{Z}_p^*$ mit $2 \leq g \leq p-2$.

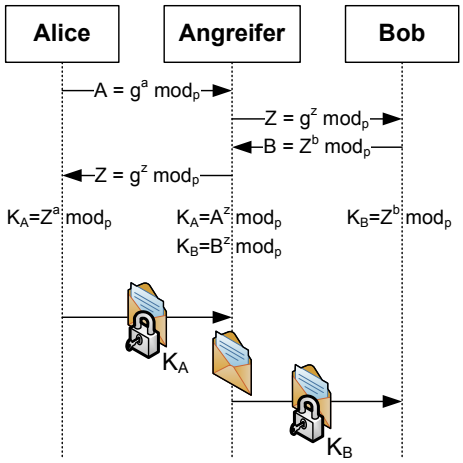


Abbildung 2.4: Man-in-the-Middle-Angriff auf den Diffie-Hellman-Schlüsseltausch

Diese beiden Zahlen können allgemein bekannt sein und daher auch über einen unverschlüsselten Kanal übertragen werden.

Da es meist aufwändig ist, zu einem p eine passende Primitivwurzel zu finden, werden häufig p und g nicht frei (innerhalb der Vorgaben) gewählt, sondern bekannte Paare von p und g verwendet.

Nachdem p und g definiert sind, wählt Alice zufällig ein $a \in \{2, 3, \ldots, p-2\}$, berechnet $A = g^a \mod p$ und sendet A unverschlüsselt an Bob. Bob wählt seinerseits zufällig ein $b \in \{2, 3, \ldots, p-2\}$, berechnet $B = g^b \mod p$ und sendet B an Alice. A und B werden im Rahmen dieser Arbeit **Diffie-Hellman-Geheimnisse** genannt.

Zuletzt berechnen Alice und Bob jeder für sich den gemeinsamen geheimen Schlüssel K:

$$K \equiv A^b \equiv g^{a^b} \equiv g^{ab} \equiv g^{b^a} \equiv B^a \pmod{p}$$

Die Sicherheit dieses Algorithmus basiert auf der Annahme, dass $K = g^{ab}$ unter Kenntnis von $A = g^a$ und $B = g^b$ nur mit hohem Aufwand berechnet werden kann. Gäbe es einen effizienten Algorithmus zur Berechnung des diskreten Logarithmus von A oder B, wäre dies ebenfalls effizient möglich. Für die Berechnung des diskreten Logarithmus von A oder B sind allerdings derzeit nur Algorithmen bekannt, deren Laufzeit bestenfalls subexponentiell zur Größe von p wächst. Ein Beweis für die Sicherheit des DHS steht jedoch noch aus.

Des Weiteren ist zu beachten, dass DHS durch eine Man-in-the-Middle-Attacke angreifbar ist. In diesem Fall lenkt der Angreifer den Kommunikationsfluss zwischen Alice und Bob derart um, dass alle Nachrichten, die Alice und Bob austauschen, zunächst zu ihm selbst gelangen. Der Angreifer kann die Nachrichten manipulieren und dann an den eigentlichen Adressaten zustellen. Abbildung 2.4 illustriert diesen Vorgang.

Während Alice und Bob während des Schlüsseltausches davon ausgehen, einen geheimen Schlüssel untereinander auszuhandeln, sorgt der Angreifer dafür, dass er nach dem Schlüsseltausch sowohl über einen gemeinsam Schlüssel mit Alice wie auch über einen weiteren mit Bob verfügt.

Tauschen Alice und Bob nach dem Schlüsseltausch verschlüsselt Nachrichten aus, werden diese zunächst vom Angreifer entschlüsselt, bei Bedarf manipuliert und dann mit dem anderen Schlüssel verschlüsselt und zugestellt. Der Angreifer hat Zugriff auf die gesamte vermeintlich vertrauliche Kommunikation.

Unter Verwendung eines Public-Key-Kryptosystems kann DHS zum **Station-To-Station-Protokoll** erweitert werden. Dieses Protokoll wurde von Diffie, Oorschot und Wiener in [7] vorgestellt. Das Station-To-Station-Protokoll bietet zusätzlich zum Schlüsseltausch eine Authentifizierung der Kommunikationsinstanzen und verhindert einen Man-in-the-Middle-Angriff.

2.5.4 Gültigkeitsdauer von Schlüsseln

Für den Einsatz in der Netzsicherheit ist jeder Schlüssel mit einer Gültigkeitsdauer verknüpft, die von der jeweiligen Verwendung des Schlüssels abhängt.

Die Gültigkeitsdauer der Schlüsselpaare eines Public-Key-Kryptosystems ist üblicherweise relativ lang (Wochen oder Monate), da der Aufwand der Schlüsselverteilung in diesem Fall vergleichsweise hoch ist.

Symmetrische Schlüssel sind dagegen eher kurzlebig (eine Sitzung; wenige Minuten) und werden nach ihrem Ablauf neu ausgehandelt oder zugestellt.

3 Peer-to-Peer Systeme [17]

Verteilte Systeme, deren Komponenten gleichberechtigt oder gleichartig sind, werden als **Peer-to-Peer-Systeme** bezeichnet.

Das englische Wort *peer* leitet sich aus dem lateinischen Wort *par* für „gleich“ ab und bezeichnet seit der ersten Hälfte des 14. Jahrhundert Mitglieder des englischen Hochadels (engl.: *peerage*). Zunächst waren alle Mitglieder des *peerage* gleichberechtigt; erst nach und nach entstand eine komplexe Hierarchie von verschiedenen Adelsrangstufen. Dennoch hat sich die ritterlich-romantische Vorstellung der *peers* als Gleichberechtigte bis heute gehalten.

Oram, Steinmetz und Wehrle definieren Peer-to-Peer Systeme in [12] und [16] wie folgt:

Definition 3.1: Peer-to-Peer System
*Ein **Peer-to-Peer System** ist ein sich selbst organisierendes System gleichberechtigter, autonomer Einheiten (Peers), das vorzugsweise ohne Nutzung zentraler Dienste auf der Basis eines Rechnernetzes mit dem Ziel der gegenseitigen Nutzung von Ressourcen operiert, also **ein System mit vollständig dezentraler Selbstorganisation und Ressourcennutzung**.*

Bei den angebotenen bzw. konsumierten *Ressourcen* kann es sich um Bandbreite, Speicherplatz, Rechenkapazität oder alle Arten von Informationen handeln. Der Zugriff auf die geteilten Ressourcen erfolgt direkt von Peer zu Peer, die Bereitstellung der Ressourcen wird nicht durch eine einzelne zentrale Komponente geleistet.

Die Hauptfunktion eines P2P-Systems ist es, den Peers einen Mechanismus zur Verfügung zu stellen, der es ermöglicht, vorhandene Ressourcen ausfindig zu machen. Zu diesem Zweck *organisieren* sich die Peers in einem **Overlay-Netz**, welches von der zugrundeliegenden Netzstruktur (beispielsweise dem Internet) abstrahiert.

Ein Overlay-Netz ist ein Graph, dessen Knoten den Peers entsprechen. Die Kanten des Overlay-Graphen können durch das P2P-System frei bestimmt werden; die einzige Bedingung ist, das eine Kommunikation zwischen zwei durch eine Kante verbundenen Knoten in dem zugrundeliegenden Rechnernetz möglich ist. Knoten, die durch eine Kante verbunden sind, heißen benachbart.

Zum Auffinden von Ressourcen werden Suchanfragen entlang den Kanten des Overlay-Graphen weitergeleitet. Nachdem eine Ressource gefunden wurde, tauschen der Peer, der die Ressource nutzen möchte, und der Peer, der sie anbietet, weitere Nachrichten unabhängig von den Kanten direkt untereinander aus (engl.: *out-of-*

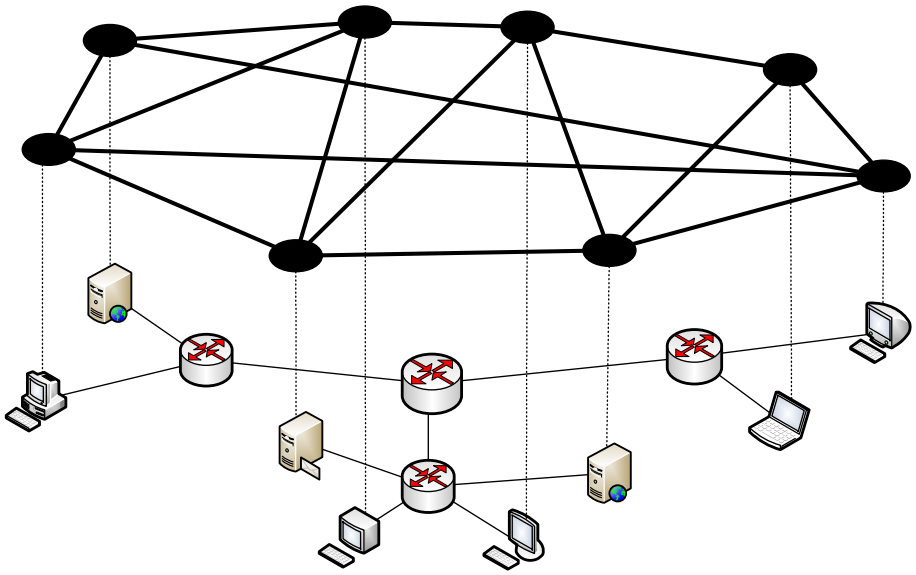


Abbildung 3.1: Rechnernetz und Overlay-Graph

band).

Abbildung 3.1 zeigt exemplarisch eine Rechnernetz-Topologie mit einem darüber liegenden Overlay-Graph. Die Knoten sind Rechnern des darunter liegenden Netzes zugeordnet, die Topologie des Overlay-Graphen ist aber davon unabhängig.

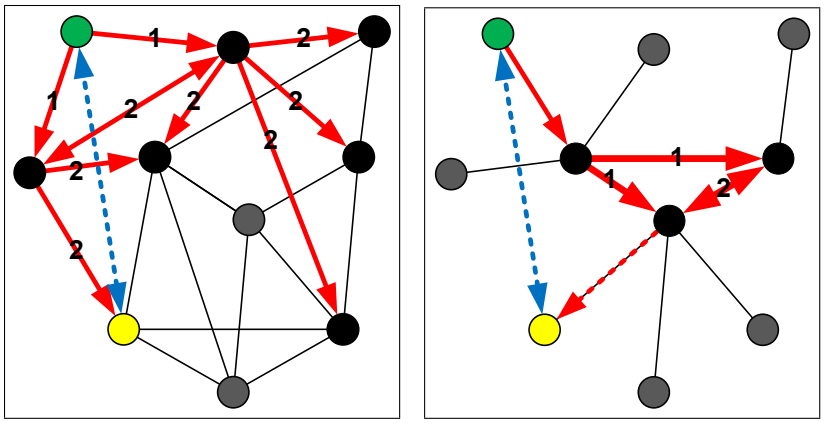
Wenn die Wahl der Kanten zwischen den Knoten zufällig erfolgt und keiner Strukturierung untergeordnet ist, wird das P2P-System als **unstrukturiert** bezeichnet; gibt es dagegen eine wohldefinierte Systematik, die die Nachbarschaft zwischen den Knoten vorschreibt, so wird das System **strukturiert** genannt.

3.1 Unstrukturierte Peer-to-Peer-Systeme

Bei **unstrukturierten P2P-Systemen** verfügen die Knoten über keine Informationen, an welchen benachbarten Knoten sie eine Suchanfrage weiterleiten müssen, damit die Anfrage einen Knoten erreicht, der über die angeforderten Ressourcen verfügt. Anfragen müssen „ziellos“ an benachbarte Knoten weitergegeben werden.

3.1.1 Reine Peer-to-Peer-Systeme

In **reinen P2P-Systemen** (engl.: *pure*) bearbeitet jeder Knoten eine empfangene Suchanfrage lokal und sendet mögliche lokale Ergebnisse zurück an den Auftraggeber. Zudem reicht er die Anfrage an alle benachbarten Peers weiter, das Netz wird geflutet (engl.: *flooding*).



(a) reines P2P-System
flooding im gesamten Netz
mit TTL = 2

(b) hybrides P2P-System
flooding unter Superpeers
mit TTL = 2

Abbildung 3.2: Unstrukturierte P2P-Systeme

Damit eine Suchanfrage nicht endlos im System kreist, besitzt jede Anfrage einen Zähler, der angibt, wie oft die Nachricht bereits weitergereicht wurde. Jeder Knoten inkrementiert diesen Zähler; ist ein Maximalwert (engl. *Time To Live (TTL)*: Lebenszeit) erreicht, wird die Nachricht nicht mehr an benachbarte Knoten weitergeleitet. Dadurch wird die Suchanfrage nur innerhalb eines bestimmten Radius um ihren Ausgangspunkt verbreitet (siehe Abbildung 3.2(a)).

Daraus folgt, dass nicht garantiert werden kann, dass eine Suchanfrage eine Antwort erhält, auch wenn eine entsprechende Ressource im System vorhanden ist. Dies geschieht, wenn der Radius der Suchanfrage geringer ist als der Durchmesser des Overlay-Graphen.

Um zu verhindern, dass eine Anfrage an einen Knoten weitergereicht wird, der diese bereits bearbeitet hat, kann einer Nachricht eine Liste bereits besuchter Knoten angehängt werden. Jeder Knoten, der eine Suchanfrage erhält, hängt seine ID an diese Liste an; die Nachricht wird nur an Knoten weitergereicht, die nicht in dieser Liste aufgeführt sind.

Dadurch wird die Zahl der weitergereichten Suchanfragen zwar verringert, es ist aber dennoch nicht ausgeschlossen, dass eine Anfrage einen Knoten mehrfach auf verschiedenen Wegen erreicht. Daher wird jede Anfrage mit einer eindeutigen ID assoziiert. Die Knoten speichern die IDs bereits empfangener Anfragen; erreicht sie eine Anfrage ein weiteres mal, wird sie ohne weitere Bearbeitung verworfen.

Diese Optimierungen können die Performance des Algorithmus zwar deutlich verbessern; dennoch skalieren P2P-Systeme, die *flooding* als Suchalgorithmus verwenden, bei einer steigenden Zahl von Knoten meist nicht, da ein großer Teil der übertragenen Daten aus redundanten oder erfolglosen Suchanfragen besteht.

Beispiele für reine P2P-Systeme sind Freenet[1] und Version 0.4 von Gnutella[2].

3.1.2 Hybride Peer-to-Peer-Systeme

Um die Suche im Overlay-Graphen auf eine geringere Anzahl von Knoten zu reduzieren, übernehmen in **hybriden P2P-Systemen** einzelne Peers dynamisch je nach Bedarf besondere Verantwortung und werden zu **Superpeers**. Als Superpeers werden nur Peers ausgewählt, die über hinreichende große Speicherkapazität und Datenübertragungsrate verfügen.

Jeder Peer ist einem Superpeer zugeordnet; die Peers teilen die von ihnen angebotenen Ressourcen ihrem Superpeer mit, der die Ressourcen in einem lokalen Index verzeichnet.

Suchanfragen werden von den Peers zunächst an ihren Superpeer gerichtet. Dieser durchsucht seinen lokalen Index und gibt mögliche Ergebnisse an den anfragenden Peer zurück. Danach wird die Suchanfrage vom Superpeer an die übrigen Superpeers (beispielsweise durch *flooding*) weitergereicht (siehe Abbildung 3.2(b)).

Auf diese Weise kann das gesamte System zuverlässiger und performanter als reine P2P-Systeme durchsucht werden, da nur ein kleiner Teilgraph (der Superpeer-Subgraph) durchsucht werden muss.

Ein hybrides P2P-System kann dynamisch auf eine sich ändernde Zahl von Peers reagieren, indem es weitere Superpeers auswählt oder Superpeers entlässt. Damit ist das System in der Lage, sich den jeweiligen Gegebenheiten optimal anzupassen.

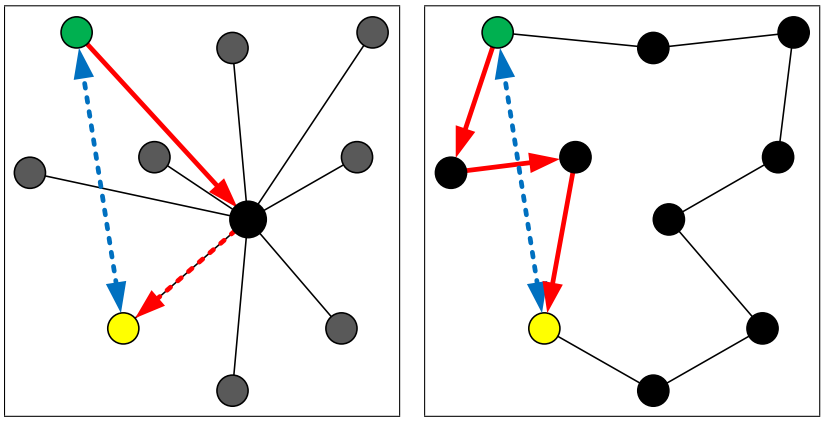
Die Version 0.6 von Gnutella[3] ist ein Beispiel für ein hybrides P2P-System.

3.2 Strukturierte Peer-to-Peer-Systeme

Ein Nachteil der unstrukturierten P2P-Systeme ist, dass Suchanfragen nicht gezielt an einzelne, sondern stets an alle benachbarten Knoten weitergegeben werden. Dadurch werden auch Knoten in den Suchprozess involviert, die die Suche einer Ressource nicht unterstützen können.

In **strukturierten P2P-Systemen** ist mit jeder Ressource ein **Suchschlüssel** assoziiert, jeder Suchschlüssel wird einem spezifischen Knoten zugeordnet. Das Overlay-Netz lotst (engl.: *to route*) Anfragen nach einem Schlüssel gezielt zu dem Knoten, der für den Suchschlüssel zuständig ist. Dadurch wird das Netz nicht überflutet, die Anfragen werden auf möglichst kurzen Pfaden weitergereicht.

[1] `freenetproject.org`
[2] `www9.limewire.com/developer/gnutella_protocol_0.4.pdf`
[3] `www.gnutella2.com`



(a) zentralisiertes P2P-System Sternstruktur

(b) Distributed Hash Table Ringstruktur

Abbildung 3.3: Strukturierte P2P-Systeme

3.2.1 Zentralisierte Peer-to-Peer-Systeme

Bei der einfachsten Form strukturierter P2P-Systeme, den **zentralisierten P2P-Systemen**[4], wird die Suchfunktionalität durch einen einzigen zentralen Index-Knoten implementiert; das P2P-Prinzip ist hier nur zum Teil umgesetzt.

Jeder Peer teilt die Suchschlüssel der von ihm bereitgestellten Ressourcen dem Index-Knoten mit. Sucht ein Peer eine bestimmte Ressource, sendet er den zugehörigen Suchschlüssel als Suchanfrage an den Index-Knoten. Dieser antwortet mit einer Liste von Peers, die die gewünschte Ressource anbieten.

Das System ist vergleichbar mit einem hybriden P2P-System, das einen einzigen Superpeer besitzt.

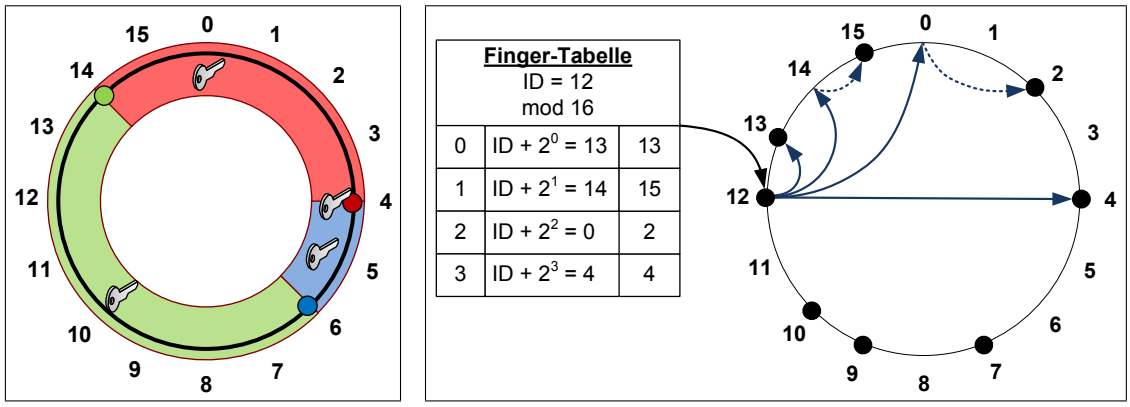
Durch das Vorhandensein einer zentralen Instanz, über die alle Anfragen laufen, skaliert dieses System bei einer großen Zahl von Peers sehr schlecht. Zudem versagt das gesamte System beim Ausfall des zentralen Index-Knotens.

Ein Beispiel für zentralisierte P2P-Systeme ist Napster [17, S. 38].

3.2.2 DHT-basierte Peer-to-Peer-Systeme

Auch bei DHT-basierten P2P-Systemen wird eine Indizierung der angebotenen Ressourcen verwendet, die es ermöglicht, Suchanfragen gezielt und performant zu bear-

[4] In der Literatur werden zentralisierte P2P-Systeme üblicherweise zu den unstrukturierten Systemen gezählt. Da sie aber eine feste Stern-Struktur aufweisen und Suchanfragen gezielt an einen zuständigen Knoten weitergeleitet werden, werden sie in dieser Arbeit den strukturierten Systemen zugeordnet.



(a) Zuteilung der Index-Intervalle ($k = 4$)

(b) Finger-Tabelle und Kanten im Overlay-Graphen ($k = 4$)

Abbildung 3.4: Beispiele für Chord-Ringe

beiten. Im Gegensatz zu zentralisierten Systemen wird die Indexstruktur allerdings nicht zentral, sondern gemeinschaftlich von allen Peers verwaltet.

Eine komplexe Menge von Suchschlüsseln kann durch eine geeignete Hashfunktion auf eine begrenzte Indexmenge abgebildet werden. Eine Index- oder **Hashtabelle**, die für jeden Index einen Verweis auf die Ressource speichert, ermöglicht es, direkt auf die zugehörige Ressource zuzugreifen.

DHT-basierte P2P-Systeme verwenden eine über das Rechnernetz verteilte Hashtabelle (engl.: *Distributed Hash Table (DHT)*), um die angebotenen Ressourcen zu indizieren.

Jeder Knoten ist für einen Teil der Indexmenge verantwortlich. Ein Knoten, der eine Ressource anbietet, berechnet den Index des Suchschlüssels und teilt dem zuständigen Knoten die angebotene Ressource mit.

Wird eine Ressource gesucht, kann über den Suchschlüssel direkt der zuständige Knoten ermittelt werden.

Ein Beispiel für ein DHT-basiertes P2P-System ist **Chord**[5]. Bei Chord werden sowohl die Suchschlüssel wie auch die Knoten in eine ringförmige Indexstruktur abgebildet.

Die Indexmenge von Chord ist $\mathcal{I} = (0...2^k - 1), k \in \mathbb{N}$; üblicherweise wird $k = 160$ gewählt. Das Intervall $\mathcal{I}$ wird zu einem modulo-2^k-Ring geformt, in dem auf den Index $2^k - 1$ der Index 0 folgt.

Jeder Knoten bekommt zufällig einen Index $i \in \mathcal{I}$ zugeteilt. Er ist für das Index-Intervall zuständig, welches zwischen ihm und dem Knoten mit dem nächst geringeren Index liegt. Dieser Knoten heißt Vorgänger (engl.: *predecessor*) von Knoten i.

[5] `pdos.csail.mit.edu/chord`

Abbildung 3.4(a) veranschaulicht die Zuordnung der Suchschlüssel-Indizes zu den Knoten. Die Suchschlüssel mit Index 0 und 4 sind dem Knoten mit Index 4 zugeordnet, der Suchschlüssel 5 dem Knoten 6 und so weiter.

Jeder Knoten i verwaltet eine Routing-Tabelle mit k Einträgen; diese Tabelle wird Finger-Tabelle (engl.: *finger table*) genannt. Die Finger-Tabelle enthält in Zeile $m \in (0...k-1)$ einen Verweis auf den Knoten, der für den Index $i + 2^m$ zuständig ist (siehe Abbildung 3.4(b)). Im Overlay-Graph ist der Knoten i mit allen Knoten in seiner Finger-Tabelle durch eine Kante verbunden.

Erhält ein Knoten eine Anfrage nach einem Suchschlüssel mit Index s, leitet er diese Anfrage an den Knoten in seiner Finger-Tabelle weiter, der den höchsten Index i mit $i \leq s$ hat.

Im Beispiel in Abbildung 3.4(b) wird eine Anfrage nach Suchindex 1 von Knoten 12 an den Knoten 15 weitergegeben. Dieser leitet im nächsten Schritt die Anfrage an den für Index 1 zuständigen Knoten 2 weiter.

Mit diesem Routing-Schema kann jede Suche in $O(\log(k))$ Routing-Schritten durchgeführt werden.

Weitere Beispiele für DHT-basierte P2P-Systeme sind Pastry[6], CAN [13] und Kademlia [9].

3.3 Peer-to-Peer Programmierschnittstellen

Derzeit implementieren die meisten P2P-Anwendungen ein eigenes, spezifisches Overlay-Netz. Das führt dazu, dass bei Verwendung mehrerer P2P-Anwendungen getrennte Systeme zum Einsatz kommen, so dass Funktionalitäten redundant ausgeführt werden.

Wenn beispielsweise mehrere Anwendungen verwendet werden, die DHT-basiert sind, muss ein Peer für jede Anwednung ein eigenes Index-Intervall verwalten. Dadurch wird mehr Speicher benötigt und es werden mehr Nachrichten ausgetauscht, als wenn alle Anwendungen den selben DHT verwenden würden.

Aus diesem Grund werden von Firmen und Forschungseinrichtungen Protokolle und Programmbibliotheken entwickelt, die es ermöglichen sollen, verschiedene P2P-Anwendungen in einem P2P-System über eine einheitliche Programmierschnittstelle auszuführen. Entwickler gewinnen dadurch den Vorteil, sich nicht um den P2P-Unterbau ihrer Anwendung kümmern zu müssen und können sich voll auf die Anwendungsfunktionalität konzentrieren.

Ein Beispiel für eine P2P-Programmierschnittstelle sind die P2P-Fähigkeiten von Windows Vista[7]. Windows Vista bietet unter anderem einen DHT-basierten Index-

[6] `www.research.microsoft.com/~antr/Pastry/`
[7] `www.microsoft.com/P2P`

Dienst an, an dem jeder unter Windows Vista betriebene Computer weltweit teilnehmen und der von jeder Windows-Anwendung verwendet werden kann. Derzeit ist allerdings nicht abzusehen, ob die verwendeten Protokolle offengelegt werden und auch außerhalb der Windows-Welt Verwendung finden können.

Die Firma Google[8] bietet die P2P-Programmbibliothek für ihre Chat- und Internet-Telefonie-Software Google-Talk unter der Bezeichnung Libjingle[9] als OpenSource-Paket an. Die P2P-Schnittstelle dieser Bibliothek stellt eine Socket-Abstraktion dar, die es ermöglicht, über Firewall- und Network Address Translation (NAT) -Grenzen hinweg Daten auszutauschen.

Collaber[10] ist eine P2P-Programmbibliothek, das den Zugriff auf ein hybrides P2P-System bietet. Auf der P2P-Funktionalität aufsetzend bietet Collaber ein Graphical User Interface (GUI), Filesharing, Datei-Synchronisation und eine Chat-Anwendung an. Die Kommunikation kann symmetrisch oder asymmetrisch verschlüsselt werden.

Von Sun[11] wird seit dem Jahr 2001 die P2P-Protokoll-Spezifikation JXTA[12] entwickelt. JXTA definiert eine Reihe von Protokollen und Mechanismen, die ein von der darunter liegenden Rechnernetz-Topologie unabhängiges Overlay-Netz ermöglichen.

Die in dieser Arbeit vorgestellte sichere P2P-Gruppenkommunikation verwendet eine JXTA-Protokoll-Implementierung als zugrundeliegendes P2P-System. Aus diesem Grund wird im nächsten Abschnitt JXTA näher erläutert.

3.3.1 JXTA - It's all about protocols.

Die JXTA-Protokollfamilie [1, 19] wurde entwickelt, um einer möglichst weit gefächerten Anwendungspalette ein gemeinsames P2P-System zur Verfügung zu stellen.

Daher wurde beim Design der Protokolle darauf geachtet, dass sie unabhängig vom verwendeten Betriebssystem sind. Die JXTA-Protokolle können für Pocket-PCs, Windows- oder Unix-Arbeitsplatzrechner, für Mac OS und für Großrechner oder Supercomputer implementiert werden. Bisher existieren Implementierungen in den Sprachen Java (Referenzimplementierung), C, C# (über einen C-Wrapper), Ruby, Python, Perl und Smalltalk.

JXTA ist ebenfalls unabhängig von der zugrundeliegenden Rechnernetz-Topologie. Die JXTA-Protokolle werden auf der Anwendungsebene des ISO-OSI-Referenzmodells implementiert. Dadurch kann JXTA auf Funk- oder Kabelnetzen, Ethernet oder TokenRing, Transmission Control Protocol (TCP) oder Hypertext

[8] www.google.com
[9] code.google.com/apis/talk/index.html
[10] www.collaber.com
[11] www.sun.com
[12] www.jxta.org

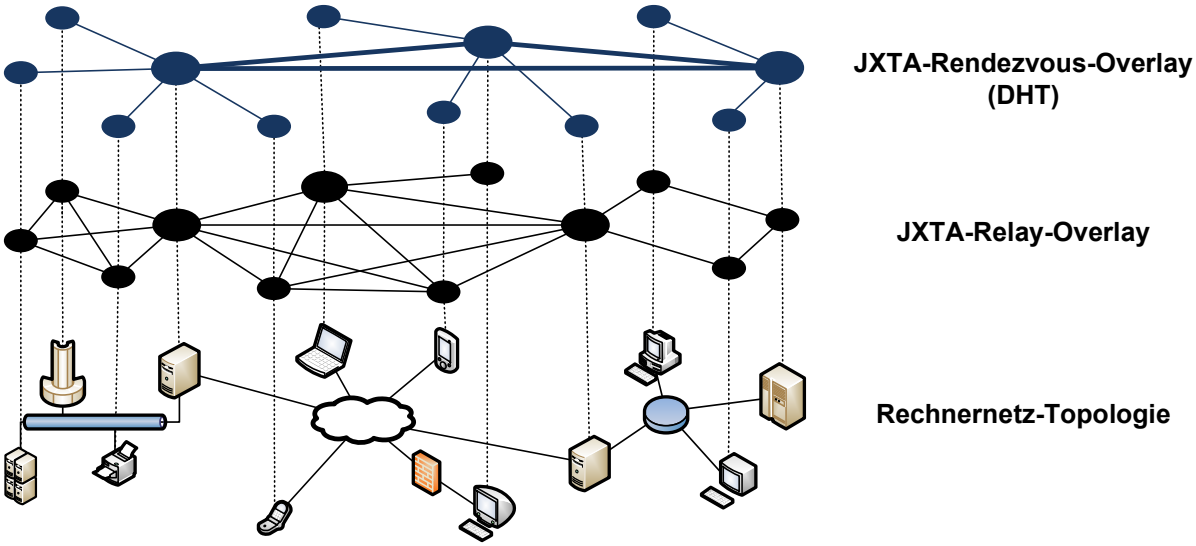


Abbildung 3.5: JXTA-Rendezvous-Overlay und JXTA-Relay-Overlay

Transfer Protocol (HTTP), dem Internet oder einem Intranet und auf allen möglichen Kombinationen dieser Netzschichten aufgesetzt werden. JXTA ermöglicht dabei das Routing zwischen verschiedenen Subnetzen und ermöglicht damit Punkt-zu-Punkt-Verbindungen zwischen allen beteiligten Instanzen.

Neben dem Routing von Nachrichten werden von JXTA weitere allgemeine P2P-Funktionalitäten wie das Suchen von Ressourcen oder das Auflösen von Namen zu Adressen angeboten.

JXTA ist eine Mischung aus einem hybriden und einem DHT-basierten P2P-System. Jeder geeignete Peer des JXTA-Netzes kann besondere Funktionen übernehmen und dadurch zu einem Superpeer werden. Ein Relay-Superpeer ist für das Weiterleiten von Nachrichten an Peers in anderen Netzsegmenten verantwortlich. Die Rendezvous-Superpeers verwalten eine verteilte Hashtabelle und beantworten Suchanfragen nach Ressourcen.

Dadurch besitzt JXTA zwei Overlay-Abstraktionen: Das Relay-Overlay abstrahiert von der zugrundeliegenden Netztopologie und ermöglicht den Datenaustausch zwischen beliebigen Peers. Auf das Relay-Overlay setzt das Rendezvous-Overlay auf, in dem die Suchindizes verwaltet werden. Abbildung 3.5 veranschaulicht diese Abstraktionsebenen.

In JXTA stellen **Peers** abstrakte Kommunikationspunkte dar. Zwischen Peer und Benutzer muss differenziert werden; ein Benutzer kann mehrere Peers gleichzeitig beispielsweise auf seinem Handy, seinem Notebook und auf seinem Rechner im Büro verwenden oder mehrere Peers auf einem einzigen Gerät betreiben.

Ein **Endpunkt** (engl.: *endpoint*) ist die Adresse eines Peers in Verbindung mit einem Kommunikationsprotokoll. Ein Peer kann mehrere Endpunkte besitzen und damit

über verschiedene Protokolle kommunizieren. JXTA definiert beispielsweise Endpunktprotokolle zur Kommunikation via HTTP oder TCP; es ist bei Bedarf möglich, weitere Endpunktprotokolle zu implementieren.

Jede Kommunikation wird in JXTA über **Pipes** (engl. für: Rohr, Leitung) abgewickelt. Pipes sind eine abstrakte Verbindung zwischen mindestens zwei Peers; sie sind an Endpunkte gebunden. Es gibt eine ganze Reihe verschiedener Arten von Pipes, beispielsweise uni- und bidirektionale Pipes, die als Punkt-zu-Punkt Pipes zwischen genau zwei Peers oder als Propagate-Pipes (engl. *propagate*: verbreiten), die mehrere Peers miteinander verbinden, ausgeführt sein können.

Über Pipes werden **Nachrichten** ausgetauscht. Die Nachrichten sind je nach Anwendung oder Protokoll entweder in Extensible Markup Language (XML) oder in einem binären Format codiert. Für einen auf HTTP aufsetzenden Endpunkt bietet sich das XML-Format an; soll die Nachricht beispielsweise verschlüsselt werden, kann das binäre Format geeigneter sein.

Peergruppen ermöglichen das Zusammenfassen mehrerer Peers zu einem Kontext; Peers können Gruppen erzeugen, ihnen beitreten und sie wieder verlassen. Peergruppen ermöglichen es, innerhalb eines großen P2P-Netzes verschiedene Anwendungen zu betreiben, ohne dass sie einander beeinträchtigen; jede Anwendung läuft im exklusiven Kontext einer eigenen Gruppe.

Peers bieten **Dienste** an, mit denen sie anderen Peers Ressourcen zur Verfügung stellen. Innerhalb von Gruppen können Dienste angeboten werden, die nur Gruppenmitgliedern zur Verfügung stehen. Anwendungsspezifische Funktionalitäten werden beispielsweise als Gruppendienste implementiert.

Peers, Peergruppen, Dienste, Endpunkte und Pipes werden in JXTA durch **Advertisements** (engl. für: Ankündigung, Annonce) beschrieben. Advertisements werden von den Superpeers verwendet, um Suchanfragen aufzulösen. Jede Information, die im P2P-Netz bekannt gemacht werden soll, wird in Form eines Advertisements veröffentlicht.

4 Sichere Gruppenkommunikation

Sichere Gruppenkommunikation (engl.: *secure group communication*) ist im Rahmen dieser Arbeit wie folgt definiert:

Definition 4.1
*Eine **sichere Gruppenkommunikation** ermöglicht einem oder mehreren autorisierten Sendern, Nachrichten in Form eines Multicast zu mehreren autorisierten Empfängern zu senden.*

*Sowohl Sender wie auch Empfänger sind **Mitglieder** der Gruppe.*

Laut Amir, Nita-Rotaru, Stanton und Tsudik sollte jedes System, das eine sichere Gruppenkommunikation implementiert, mindestens folgende Dienste anbieten [2]:

Authentifizierung: Jede Kommunikationsinstanz, die das System verwendet, muss authentifiziert werden.

Vertraulichkeit: Der Inhalt der übertragenen Nachrichten darf nur für autorisierte Kommunikationsteilnehmer zugänglich sein.

Integrität: Veränderungen an den übertragenen Nachrichten müssen durch Empfänger eindeutig erkannt werden können.

Gruppenschlüssel-Management: Um Vertraulichkeit und Integrität zu ermöglichen, muss das System die Erzeugung und Verwaltung eines Gruppenschlüssels anbieten.

Zugriffskontrolle (engl.: *access control*): Wenn eine Kommunikationsinstanz auf System-Ressourcen zugreift (beispielsweise Beitritt zur Gruppe, Senden und Empfangen von Gruppennachrichten), muss überprüft werden, ob sie zu diesem Zugriff autorisiert ist.

4.1 Sichere Gruppenkommunikation in Peer-to-Peer-Systemen

Der Nachrichtentransport einer sicheren Gruppenkommunikation basiert auf einem Multicast-Mechanismus, der die Nachrichten an alle Gruppenmitglieder zustellt. Es gibt verschiedene P2P-Systeme, die Multicast-Mechanismen implementieren, bei-

spielsweise die Internet Indirection Infrastructure[1] (*i*3) [18], JXTA oder Scribe [5].

Die Dienste der sicheren Gruppenkommunikation können in einem P2P-System wie folgt realisiert werden:

Für die Authentifizierung der Kommunikationsinstanzen kann beispielsweise Public-Key-Kryptographie verwendet werden.

Das Schlüssel-Management kann über zwei verschiedenen Ansätze realisiert werden:

- **verteilt:** Seit einiger Zeit werden Verfahren zur verteilten Erzeugung von Gruppenschlüsseln erforscht und entwickelt [14, 3, 15]. Diese Verfahren sind eine Erweiterung des Diffie-Hellman-Schlüsseltausch für mehrere Kommunikationsteilnehmer.

  Da bei diesen Verfahren die Kommunikationsteilnehmer meist gleichberechtigt sind, eignen sie sich gut für P2P-Systeme.

- **zentralisiert:** Es kann eine zentrale Instanz verwendet werden, die den Gruppenschlüssel erzeugt und an Gruppenmitglieder weiter gibt. Die Übertragung des Gruppenschlüssels muss über einen sicheren Kanal erfolgen[2].

  Diese Variante widerspricht allerdings dem P2P-Prinzip der dezentralen Selbstorganisation.

Damit existieren Mechanismen für die Erzeugung eines Gruppenschlüssels, mit dem Vertraulichkeit und Integrität der Gruppenkommunikation gewährleistet werden können.

Kim, Mazzocchi und Tsudik stellen in [8] drei verschiedene Arten der Zugriffskontrolle für P2P-Systeme vor:

1. **ACL:** Eine Access Control Liste (ACL) verzeichnet alle Peers, die autorisiert sind, an der sicheren Gruppenkommunikation teilzunehmen. Als Variante kann eine negative ACL (NACL) verwendet werden, die unautorisierte Peers auflistet.

   Es muss genau definiert sein, wer an der Liste Änderungen durchführen, das heißt Peers einladen oder ausschließen darf.

   Jeder Peer muss Zugriff auf die Liste haben, damit er bei der Erzeugung oder Weitergabe des Gruppenschlüssels feststellen kann, welche Peers in den Besitz des Schlüssels gelangen dürfen. Bei jeder Änderung an der Liste müsse alle Peers über diese Änderung informiert werden.

   Wenn den Peers unterschiedliche Listen vorliegen, ist die Zugriffskontrolle unwirksam. Das macht die Verbreitung der ACL zu einem möglichen Angriffspunkt für einen Angreifer.

2. **GROUP:** Über die Autorisierung eines Peers wird von einer statischen oder dynamischen Anzahl von Gruppenmitgliedern entschieden:

[1] `i3.cs.berkeley.edu`
[2] vgl. Abschnitt 2.5.3: Schlüsselverteilung

- **statisch:** Eine feste Anzahl von Gruppenmitgliedern stimmt über die Autorisierung ab.
- **dynamisch:** Ein definierter Prozentsatz der Gruppenmitglieder stimmt über die Autorisierung ab.

Bei dieser Variante stellt sich die Frage, auf welcher Basis die einzelnen Gruppenmitglieder eine Entscheidung über die Autorisierung eines Peers treffen. In der Literatur wird dies meist offen gelassen [8, 11, 20].

Ein Peer gilt als autorisiert, wenn ein definierter Prozentsatz der Gruppenmitglieder der Autorisierung zustimmt. Dies ist notwendig, da ein böswilliges Gruppenmitglied sonst berechtigten Peers die Autorisierung verweigern und damit die Zugriffskontrolle unwirksam machen könnte.

Daraus resultiert, dass ein Peer nicht sicher, sondern nur "wahrscheinlich" autorisiert ist, die Autorisierung ist unscharf.

3. GAUTH: Eine Gruppenautorität (engl.: *group authority*) entscheidet über die Autorisierung eines Peers. Die Gruppenautorität kann der Gruppengründer oder ein vertrauenswürdiger Dritter (engl.: *trusted third party*) sein.

Die Verwendung einer Gruppenautorität besitzt nicht die Nachteile der ACL- und GROUP-Autorisierung, die Autorisierung einer Gruppenautorität ist verlässlich und eindeutig.

Ihr Nachteil ist allerdings, dass die Einführung einer zentralen Gruppenautorität gegen das P2P-Prinzip der dezentralen Selbstorganisation verstößt.

Bei den Varianten ACL und GROUP kann die Zugriffsberechtigung eines Peers nicht mit absoluter Sicherheit festgestellt werden. Daher ist eine Gruppenautorität (GAUTH) derzeit die sicherste Variante zur Zugriffskontrolle in P2P-basierten, sicheren Gruppenkommunikationssystemen. Die Notwendigkeit einer garantierten Überprüfung der Zugriffsberechtigung eines Peers überwiegt in diesem Fall den Nachteil der Verwendung einer zentralen Komponente gegenüber der dezentralen Selbstorganisation.

5 Entwurf

Die Aufgabe dieser Diplomarbeit ist es, ein System zu entwickeln, das einer Gruppe von autorisierten Benutzern ermöglicht, auf einem gemeinsamen verschlüsselten Kommunikationskanal Daten auszutauschen. Da die Kommunikation als Multicast direkt zwischen gleichberechtigten Benutzern und nicht zwischen Clients und einem Server stattfinden soll, ist die Verwendung eines P2P-Overlays vorgesehen.

Diese Anforderung deckt sich mit Definition 4.1 einer sicheren Gruppenkommunikation, bei der Mitglieder der Gruppe sowohl zum Senden wie auch zum Empfangen von Nachrichten berechtigt sind.

Zur Sicherung der Gruppenkommunikation wird eine Zugriffskontrolle benötigt. Diese Zugriffskontrolle soll durch eine zentrale Gruppenautorität durchgeführt werden, die Identität und Teilnahmeberechtigung der Gruppenmitglieder zweifelsfrei nachweisen kann.

5.1 Systembeschreibung

Das in dieser Diplomarbeit entwickelte System „Secure Communication Group“ (SCG) ermöglicht berechtigten Benutzern, den **Gruppenverwaltern**, Gruppen zu erzeugen, Benutzer in diese Gruppen einzuladen sowie Mitglieder wieder auszuschließen. Eingeladene Benutzer können der Gruppe beitreten und an der kryptographisch gesicherten Gruppenkommunikation teilnehmen.

Die Authentifizierung eines Benutzers gegenüber der Gruppe wird durch einen zentralen Dienst mit der Bezeichnung **`MembershipService`** ermöglicht. Der `MembershipService` speichert die Gruppenmitgliedschaften der Benutzer. Gruppenverwalter teilen dem `MembershipService` neue Mitgliedschaften mit.

Ein hinzukommender Benutzer muss sich nicht gegenüber einem Gruppenverwalter oder der Gruppe, sondern gegenüber dem `MembershipService` authentisieren und erhält von diesem einen Gruppenschlüssel, der dem Benutzer die Teilnahme an der sicheren Gruppenkommunikation ermöglicht.

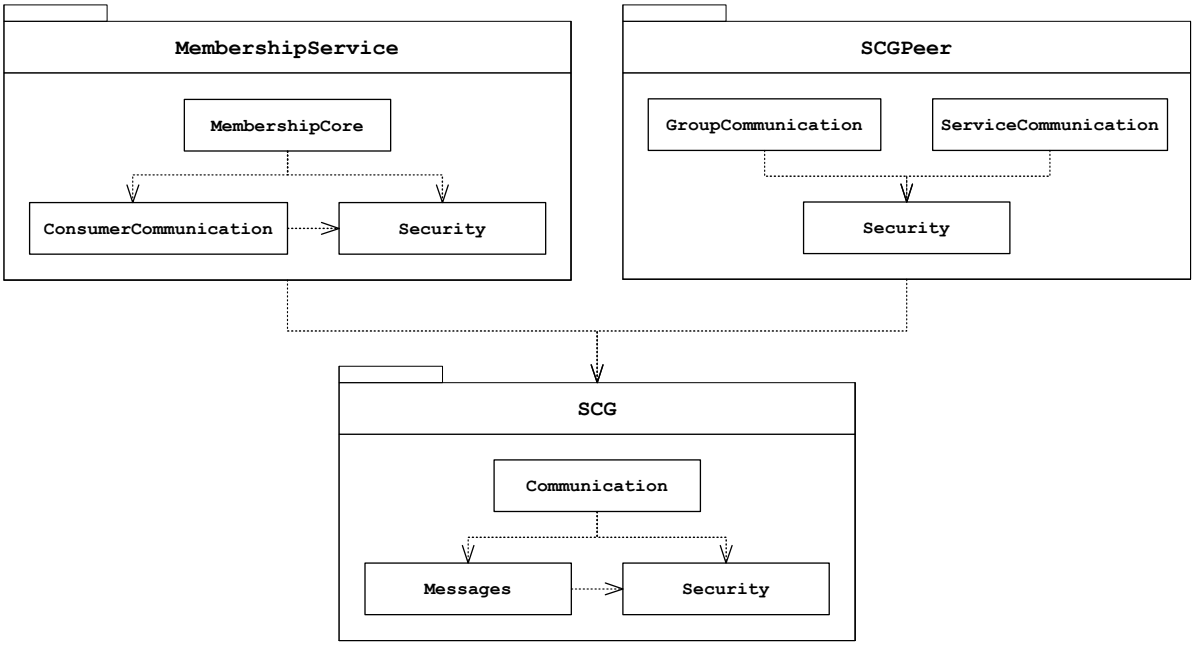


Abbildung 5.1: Systemarchitektur

5.2 Systemarchitektur

Innerhalb des Systems werden die Benutzer durch Peers repräsentiert. Jeder Peer nimmt eine Identität des Benutzers an, den er repräsentiert. Der selbe Benutzer kann mit verschiedenen Peers unter unterschiedlichen Identitäten im System vertreten sein.

Der `MembershipService` wird ebenfalls durch einen Peer repräsentiert. Dennoch werden im Folgenden stets ausschließlich die Repräsentationen eines Benutzers als *Peer* bezeichnet, der `MembershipServer` wird stets namentlich benannt.

Abbildung 5.1 stellt die Einteilung des Systems in Pakete und Unterpakete dar. Das System ist in die drei Pakete `SCG`, `MembershipService` und `SCGPeer` aufgeteilt, die wiederum drei je Unterpakete besitzen.

Unterpakete des Paketes `SCG`:

Das Paket `SCG` enthält die gemeinsamen Funktionalitäten, die sowohl von den Peers als auch vom `MembershipService` verwendet werden:

- Im Unterpaket `Communication` (engl. für: Kommunikation) befinden sich allgemeine Strukturen für die Kommunikation und Mechanismen zur Interaktion mit dem zugrundeliegenden P2P-System.
- Das Unterpaket `Security` (engl. für: Sicherheit) bietet Algorithmen und Strukturen zur Sicherung der Kommunikation an.

- Im Unterpaket `Messages` (engl. für: Nachrichten) sind die Nachrichtenformate der Nachrichten enthalten, auf denen die Kommunikation innerhalb der Gruppen und zwischen Peers und `MembershipService` basiert.

Auf das Paket `SCG` setzen zwei Pakete auf, die die Funktionalitäten der Peers und die des `MembershipService` differenzieren:

Unterpakete des Paketes `MembershipService`:

- Die Service-Funktionalitäten befinden sich im Unterpaket `Core`.
- Das Unterpaket `ConsumerCommunication` dient der Kommunikation mit den Peers.
- Im Unterpaket `Security` werden die Sicherheitsfunktionen des `SCG`-Paketes für die Verwendung durch den MembershipService spezialisiert.

Unterpakete des Paketes `SCGPeer`:

- Im Unterpaket `GroupCommunication` ist die Gruppenkommunikation untergebracht.
- Das Unterpaket `ServiceCommunication` ermöglicht die Kommunikation mit dem `MembershipService`.
- Das Unterpaket `Security` spezialisiert die Sicherheitsfunktionen des `SCG`-Paketes.

5.3 Kommunikationskanäle

Es werden zwei verschiedene Kommunikationskanäle benötigt:

1. **`GroupChannel`**: Der `GroupChannel` (engl. für: Gruppenkanal) ermöglicht die Kommunikation zwischen den Peers innerhalb einer Gruppe.
2. **`ServiceChannel`**: Der `ServiceChannel` (engl. für: Servicekanal) ermöglicht die Kommunikation zwischen den Peers und dem `MembershipService`.

Sowohl der `GroupChannel` als auch der `ServiceChannel` ist mit einem symmetrischen Schlüssel verschlüsselt, um die Vertraulichkeit der übertragenen Daten zu gewährleisten. Der Schlüssel, mit dem `GroupChannel` verschlüsselt wird, heißt **`GroupKey`** (engl. für: Gruppenschlüssel). Er besteht aus zwei Komponenten, dem **`InvitationKey`** (engl. für: Einladungsschlüssel), den der Gruppenverwalter erzeugt, und dem **`MembershipKey`** (engl. für: Mitgliedsschlüssel), den der `MembershipService` erzeugt. Die genaue Funktion dieser Schlüssel wird in Abschnitt 5.6.4 beschrieben.

5.4 Nachrichten

Die Kommunikation der Peers innerhalb einer Gruppe und mit dem `Membership-Service` wird durch den Austausch von Nachrichten abgewickelt.

Definition 5.1: Nachricht, Nachrichtenelement und Nachrichtenformat
*Eine **Nachricht** ist eine Liste von Nachrichtenelementen.*

*Ein **Nachrichtenelement** ist ein Tupel aus einem eindeutigen Namen und einem Inhalt. Der Inhalt eines Nachrichtenelementes sind die zu übertragenden Daten. Mit Hilfe des Namens können die Daten einem Bedeutungskontext zugeordnet werden.*

*Durch ein **Nachrichtenformat** wird bestimmt, welche Nachrichtenelemente eine Nachricht enthalten muss oder kann.*

Eine Nachricht ist also eine durch ihr Nachrichtenformat definierte Menge von Namens-Inhalts-Paaren.

5.5 Gruppen

Gruppen bilden eine Baumstruktur: mit Ausnahme der **Wurzelgruppe** ist jede Gruppe **Untergruppe** einer anderen Gruppe. Eine Gruppe, die Untergruppen besitzt, wird **Obergruppe** ihrer Untergruppen genannt.

Jedes Mitglied einer Gruppe hat in dieser Gruppe definierte **Rechte**, die ihm erlauben oder verbieten, bestimmte **Gruppenoperationen** - wie beispielsweise das Erfragen des Gruppenschlüssels oder das Einladen von neuen Mitgliedern - in der Gruppe auszuführen.

Durch diese hierarchische Einteilung der Gruppen in Ober- und Untergruppen und die gruppenspezifische Vergabe von Berechtigungen ist es möglich, einer Vielzahl Anwendungsfällen gerecht zu werden.

Sollen sich beispielsweise die Mitarbeiter einer Firma eigenverantwortlich in Interessensgruppen organisieren können, die keine firmenfremde Mitglieder haben sollen, können alle Mitarbeiter in eine gemeinsame Obergruppe aufgenommen werden. Sie erhalten in dieser Gruppe die Berechtigung, Untergruppen zu erzeugen, in die sie nur Mitglieder der Obergruppe einladen dürfen. Durch die Eigenverantwortung der Mitarbeiter wird eine große Flexibilität erreicht und dennoch sichergestellt, dass sensible Informationen nicht die Firma verlassen.

In den folgenden beiden Abschnitten werden die Rechte, die ein Mitglied in einer Gruppe besitzen kann, und die damit verbundenen Gruppenoperationen erläutert.

5.5.1 Gruppenrechte

Insgesamt existieren fünf verschiedene Gruppenrechte, wobei einzelne Rechte andere Rechte implizieren können. Die Hierarchie der Rechte ist in Abbildung 5.2 dargestellt.

`Communicate`: Ein Benutzer, der das `Communicate`-Recht in einer Gruppe besitzt, kann den aktuellen Gruppenschlüssel dieser Gruppe vom `MembershipService` erfragen und somit an der Gruppenkommunikation teilnehmen.

`CreateSubGroup`: Das `CreateSubGroup`-Recht erlaubt einem Benutzer, in einer Gruppe eine neue Untergruppe zu erzeugen. Der Benutzer ist automatisch Mitglied der Untergruppe und erhält dort das `SubGroupAdmin`-Recht sowie alle Rechte, die er in der Obergruppe besitzt.
Nur der Erzeuger (engl.: *creator*) einer Gruppe und Gruppenverwalter können eine Gruppe wieder auflösen.

`SeeMembers`: Ein Benutzer, der in einer Gruppe das `SeeMembers`-Recht besitzt, erhält auf Anfrage vom `MembershipService` eine Liste aller übrigen Mitglieder der Gruppe.

`SubGroupAdmin`: Ein Benutzer, der in einer Gruppe über das `SubGroupAdmin`-Recht (*admin* steht für engl. *administrator*: Verwalter) verfügt, wird **Untergruppenverwalter** genannt.
Ein Untergruppenverwalter darf ein Mitglied der Obergruppe, einen so genannten **Kandidaten** (engl.: *candidate*), in seine Gruppe einladen.
Des Weiteren kann ein Untergruppenverwalter anderen Gruppenmitgliedern die Rechte `SubGroupAdmin`, `Communicate` und `SeeMembers` gewähren oder entziehen. Das `CreateSubGroup`-Recht kann er genau dann modifizieren, wenn er selbst dieses Recht in der Gruppe besitzt. Das `GroupAdmin`-Recht kann er weder gewähren noch entziehen.
Ein Untergruppenverwalter kann Mitglieder seiner Gruppe aus der Gruppe ausschließen.
Der Besitz des `SubGroupAdmin`-Rechtes impliziert das `SeeMembers`-Recht.

`GroupAdmin`: Nur Gruppenverwalter besitzen das `GroupAdmin`-Recht in ihrer Gruppe. Ein Gruppenverwalter kann beliebige neue Benutzer in die Gruppe aufnehmen, nachdem er die Identität eines neuen Benutzers zuvor verifiziert hat[1].
Ein Gruppenverwalter kann anderen Gruppenmitgliedern alle Rechte gewähren und entziehen.
Das `GroupAdmin`-Recht impliziert die Rechte `SubGroupAdmin`, `SeeMembers` und `CreateSubGroup`.

[1] vgl. Kapitel 5.6.2: Verteilung öffentlicher Schlüssel

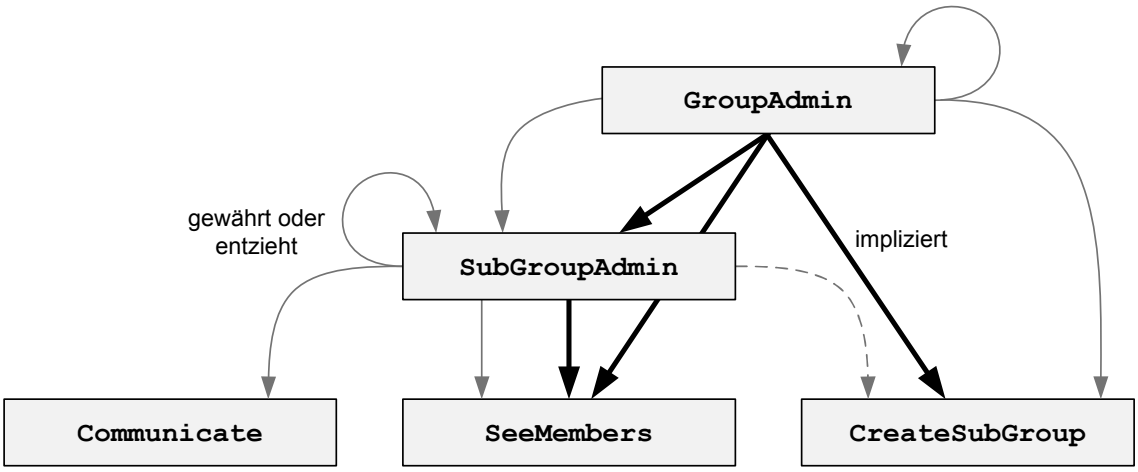


Abbildung 5.2: Hierarchie der Gruppenrechte.

5.5.2 Gruppenoperationen

Die Gruppenoperationen werden ausgeführt, indem ein Peer eine entsprechende Anfrage-Nachricht (engl.: *request message*) an den `MembershipService` sendet.

Dieser überprüft, ob der Peer die nötigen Rechte für diese Operation besitzt, und führt entsprechende Änderungen an Gruppen und Mitgliedschaften durch. Verfügt der Peer, der die Anfrage-Nachricht gesendet hat, nicht über die notwendigen Rechte, wird die Ausführung der Operation abgelehnt.

Der `MembershipService` teilt dem anfragenden Peer in einer Antwort-Nachricht (engl.: *response message*) mit, ob die Anfrage erfolgreich durchgeführt werden konnte.

Es werden folgende Gruppenoperationen angeboten:

1. `GetMemberships`-Operation

Nachdem ein Peer sich mit dem P2P-Overlay verbunden hat, führt er die `GetMemberships`-Operation aus, um seine Gruppenmitgliedschaften vom `MembershipService` zu erhalten.

Zu diesem Zweck sendet er eine **`MembershipsRequest`-Nachricht** an den `MembershipService`. Eine `MembershipsRequest`-Nachricht überträgt keine Informationen an den `MembershipService` und besitzt daher keine Nachrichtenelemente.

Der `MembershipService` antwortet auf eine `MembershipsRequest`-Nachricht mit einer **`MembershipsResponse`-Nachricht**, die die Gruppenmitgliedschaften und die offenen Gruppeneinladungen des Peers enthält. Ist der anfragende Peer ein Gruppenverwalter, erhält er zusätzlich die Daten aller bisher von ihm eingeladenen Benutzer.

Das `MembershipsResponse`-Nachrichtenformat sieht folgende Nachrichtenele-

mente vor:

`Memberships`: Das Nachrichtenelement `Memberships` enthält eine Liste der Gruppenmitgliedschaften des Peers einschließlich der `InvitationKeys` der Gruppen.

`(Invitations)`: Das Nachrichtenelement `Invitations` ist optional. Falls offene Gruppeneinladungen vorliegen, enthält dieses Element eine Liste der angebotenen Mitgliedschaften.

`(InvitedUsers)`: Wenn der anfragende Peer ein Gruppenverwalter ist, der zuvor neue Benutzer in Gruppen eingeladen hat, enthält dieses optionale Nachrichtenelement eine Liste der von ihm eingeladenen Benutzer.

2. `AcceptInvitations`-Operation

Die `AcceptInvitations`-Operation wird nur durchgeführt, wenn offene Einladungen vorhanden sind.

Nachdem ein Peer offene Einladungen erhalten hat, teilt er dem `MembershipService` durch eine `AcceptInvitations`-Operation mit, welche der offenen Einladungen er annimmt. Zu diesem Zweck sendet er eine **`AcceptInvitations`-Nachricht** an den `MembershipService`.

Das Nachrichtenformat einer `AcceptInvitations`-Nachricht enthält folgende Nachrichtenelemente:

`AcceptedInvitations`: Das Element `AcceptedInvitations` enthält eine Liste der Einladungen, die der Benutzer annehmen möchte.

`RejectedInvitations`: Dieses Element enthält eine Liste der abgelehnten Einladungen.

Der `MembershipService` quittiert die `AcceptInvitations`-Nachricht mit einer **`AckResponse`-Nachricht** (*ack* steht für engl. *acknowledgement*: Bestätigung). Diese Nachricht informiert den Peer darüber, ob die Operation erfolgreich durchgeführt werden konnte.

Eine `AckResponse`-Nachricht besitzt zwei Elemente:

`OpperationSucceded`: Dieses Element gibt an, ob die Operation erfolgreich durchgeführt werden konnte.

`(Comment)`: Konnte der `MembershipService` die Operation nicht durchführen, kann mit diesem optionalen Nachrichtenelement eine Fehlermeldung an den Peer gesendet werden.

3. `GetGroupKeys`-Operation

Nachdem der Peer seine Gruppenmitgliedschaften kennt und mögliche Einladungen angenommen oder abgelehnt hat, kann er im Zuge einer `GetGroupKeys`-Operation mit einer **`GroupKeysRequest`-Nachricht** die aktuellen Gruppenschlüssel der Gruppen erfragen, in denen er `Communicate`-Rechte besitzt.

Das Nachrichtenformat einer `GroupKeysRequest`-Nachricht hat nur ein Element mit dem Namen `Groups`. Es enthält eine Liste der Gruppen, deren aktuelle Gruppenschlüssel der Peer benötigt.

Die `GetGroupKeys`-Operation wird auch ausgeführt, wenn Gruppenschlüssel abgelaufen sind und der Peer diese erneuern möchte.

`GroupKeysRequest`-Nachrichten beantwortet der `MembershipService` mit einer **`GroupKeysResponse`-Nachricht**, in deren `GroupKeys`-Nachrichtenelement eine Liste der aktuellen `MembershipKey`s übermittelt wird.

Aus den `MembershipKey`s und den durch die `GetMemberships`-Operation erhaltenen `InvitationKey`s erzeugt der Peer die aktuellen Gruppenschlüssel.

Nachdem die Operationen `GetMemberships` und `GetGroupKeys` durchgeführt wurden, besitzt der Benutzer alle notwendigen Informationen, um an der sicheren Gruppenkommunikation teilzunehmen. Die im Folgenden beschriebenen Operationen dienen allein der Gruppenverwaltung:

4. `GetMembers`-Operation

Wenn ein Benutzer in einer Gruppe das `SeeMembers`-Recht besitzt, kann er mit einer `GetMembers`-Operation eine Liste der übrigen Mitglieder dieser Gruppe in Erfahrung bringen.

Der Peer sendet eine **`MembersRequest`-Nachricht** an den `MembershipService`, die in ihrem `Group`-Element die gewünschte Gruppe enthält.

Als Antwort sendet der `MembershipService` eine **`MembersResponse`-Nachricht**, die eine Liste der Gruppenmitglieder enthält. Wenn der Peer in der angeforderten Gruppe über das `SubGroupAdmin`-Recht verfügt, enthält die Nachricht zusätzlich eine Liste der Mitglieder der Obergruppe.

Das `MembersResponse`-Nachrichtenformat hat damit bis zu zwei Elemente:

`Members`: Die Liste der Gruppenmitglieder.

`(Candidates)`: Das `Candidates`-Element ist optional; es enthält die Mitglieder der Obergruppe, die mögliche Kandidaten für eine Mitgliedschaft in der angeforderten Gruppe sind.

5. `CreateSubGroup`-Operation

Gruppenverwalter und Gruppenmitglieder mit dem `CreateSubGroup`-Recht sind berechtigt, durch das Ausführen der `CreateSubGroup`-Operation eine neue Untergruppe zu erzeugen.

Der Gruppenerzeuger benötigt den zu diesem Zeitpunkt von ihm selbst erstellten `InvitationKey`, um Benutzer in die Gruppe einladen zu können. Damit er ihn nicht selbst für die spätere Benutzung speichern muss, sendet er ihn mit seinem eigenen öffentlichen Schlüssel verschlüsselt (und damit nur für ihn selbst lesbar) zusammen mit einer Signatur an den `MembershipService`.

Dieser speichert `Invitationkey` und Signatur für die spätere Verwendung durch den Gruppenerzeuger; beide werden dem Gruppenerzeuger im Zuge einer `GetMemberships`-Operation zugestellt. Anhand der Signatur kann dieser dann verifizieren, dass der `InvitationKey` nicht manipuliert wurde.

Die **`CreateSubGroupRequest`**–Nachricht besteht aus vier Elementen:

`Group`: Die Daten der Gruppe, die erzeugt werden soll.

`InvitationKey`: Der mit dem öffentlichen Schlüssel des Gruppenerzeugers verschlüsselte `InvitationKey` der Gruppe.

`InvitationSignature`: Die Signatur des verschlüsselten `InvitationKey`.

Die Antwort des `MembershipService` wird dem Peer in einer **`CreateSubGroup`–Response-Nachricht** übermittelt. Diese besitzt das Element `Rights`, das die Rechte des Gruppenerzeugers in der neuen Gruppe enthält.

6. `InviteUser`-Operation

Die `InviteUser`-Operation erlaubt einem Gruppenverwalter einen Benutzer in eine Gruppe einzuladen. Ein Untergruppenverwalter kann mit dieser Operation ein Mitglied der Obergruppe in die Gruppe einladen.

Die Anfrage-Nachricht der `InviteUser`-Operation ist eine **`InviteUser`-Nachricht** mit folgenden Elementen:

`User`: Der Benutzer, der eingeladen werden soll.

`Group`: Die Gruppe, in die der Benutzer eingeladen werden soll.

`Rights`: Die neuen Rechte, die der Benutzer in der Gruppe erhält.

`InvitationKey`: Der mit dem öffentlichen Schlüssel des einzuladenden Benutzers verschlüsselte `InvitationKey`.

`InvitationSignature`: Anhand der Signatur des `InvitationKey` kann der

eingeladene Benutzer die Identität des Gruppenverwalters, von dem er eingeladen wird, verifizieren.

Der `MembershipService` quittiert den Erfolg oder Misserfolg der Operation mit einer `AckResponse`-Nachricht.

7. `ModifyMembership`-Operation

Durch eine `ModifyMembership`-Operation können Gruppenverwalter und Untergruppenverwalter die Rechte von Gruppenmitgliedern modifizieren. Dazu wird eine **`ModifyMembership`-Nachricht** an den `MembershipService` gesendet.

Die `ModifyMembership`-Nachricht besitzt drei Elemente:

`User`: Der Benutzer, dessen Rechte geändert werden sollen.

`Group`: Die Gruppe, in der die Rechte des Benutzers geändert werden sollen.

`Rights`: Die neuen Rechte, die der Benutzer in der Gruppe erhält.

Der `MembershipService` teilt mit einer `AckResponse`-Message mit, ob die Modifikation erfolgreich durchgeführt werden konnte.

8. `RemoveMembers`-Operation

Mit der `RemoveMembers`-Operation können (Unter-)Gruppenverwalter Mitglieder aus einer Gruppe ausschließen.

Die **`RemoveMembers`-Nachricht** besteht aus folgenden Elementen:

`Group`: Die Gruppe, aus der die Mitglieder ausgeschlossen werden sollen.

`Members`: Eine Liste der auszuschließenden Mitglieder.

Auch diese Operation wird vom `MembershipService` mit einer `AckResponse`-Nachricht quittiert.

9. `RemoveGroups`-Operation

Die `RemoveGroups`-Operation kann von Gruppenverwaltern und Gruppenerzeugern angewendet werden.

Das **`RemoveGroups`**–Nachrichtenformat besitzt ein Element mit dem Namen `Groups`. Dieses Element enthält eine Liste der Gruppen, die gelöscht werden sollen.

Wenn eine der Gruppen nicht gelöscht werden kann, da sie mindestens eine Untergruppe besitzt oder der anfragende Benutzer keine Rechte zum Löschen der Gruppe hat, schlägt die Operation fehl und keine der Gruppen wird gelöscht.

Der `MembershipService` quittiert den Erfolg der Operation mit einer `AckResponse`-Nachricht.

Rechte \ Operationen	GetMemberships	AcceptInvitations	GetGroupKeys	GetMembers	CreateSubGroup	InviteUser	ModifyRights	RemoveGroup	RemoveUser
Communicate	Ein Anwender benötigt keinerlei Rechte, um diese Operationen auszuführen.		●					und Erzeuger der Gruppe	
CreateSubGroup					●				
SeeMembers				●					
SubGroupAdmin				●		○	○		●
GroupAdmin				●	●	●	●	●	●

● Operation kann uneingeschränkt ausgeführt werden

○ Ausführung der Operation unterliegt Einschränkungen

Abbildung 5.3: Gruppenoperationen und erforderliche Rechte

Die Tabelle in Abbildung 5.3 fasst zusammen, welche Gruppenoperationen durchgeführt werden können und welche Rechte zur ihrer Durchführung notwendig sind.

5.6 Sicherheitsarchitektur

Die Sicherheitsarchitektur von SCG basiert darauf, dass Benutzern abhängig von ihrer Identität verschiedene Rechte eingeräumt werden.

Benutzer können je nach Kontext über eine oder mehrere Identitäten verfügen. Im Anwendungsfall der Gruppenkommunikation kann ein Benutzer beispielsweise unter verschiedenen Identitäten innerhalb derselben Gruppe kommunizieren oder für verschiedene Gruppen unterschiedliche Identitäten verwenden. Eine Identität kann die Rechte für Gruppenerzeugung und -verwaltung oder lediglich für die Mitgliedschaft in einer Gruppe besitzen.

Zum Nachweis der Identität eines Peers wird ein Public-Key-Kryptosystem verwendet. Die folgenden beiden Abschnitte beschreiben, wie die Schlüsselpaare verlässlich

erzeugt und die öffentlichen Schlüssel verteilt werden, so dass sie die Authentifizierung der Peers ermöglichen.

Anschließend wird beschrieben, wie die Vertraulichkeit, Integrität, Authentizität und Verbindlichkeit der Kommunikationskanäle gewährleistet wird.

Zuletzt wird die Sicherheitsschicht betrachtet, die die Kenntnis der notwendigen Kommunikationsschlüssel sicherstellt und die für die Verschlüsselung der Nachrichten sorgt.

5.6.1 Erzeugung und Schutz der privaten Schlüssel

Damit jeder Kommunikationsteilnehmer zweifelsfrei authentifiziert werden kann, besitzen sowohl der `MembershipService` als auch jede Identität eines Benutzers ein asymmetrisches Schlüsselpaar.

Private Schlüssel der Benutzer

Das Schlüsselpaar jeder Identität eines Benutzers wird von ihm selbst erzeugt, damit der private Schlüssel ausschließlich ihm bekannt ist.

Damit der private Schlüssel dauerhaft (beispielsweise auf einer Festplatte oder einem Memory-Stick) gespeichert werden kann, wird er mit einem symmetrischen Schlüssel verschlüsselt. Dieser symmetrische Schlüssel wird aus einem Mantra[2] (engl. *passphrase*) erzeugt, das nur dem Benutzer bekannt ist. Dadurch ist der private Schlüssel vor dem Zugriff anderer geschützt, nur der Benutzer selbst kann den privaten Schlüssel entschlüsseln.

Bei jedem Start der P2P-Anwendung muss der Benutzer das Mantra eingeben; entschlüsselt wird der private Schlüssel nur in flüchtigem Speicher vorgehalten.

Somit kann lückenlos aus der Kenntnis des geheimen Schlüssels auf die Identität des Benutzers geschlossen werden.

Privater Schlüssel des `MembershipService`

Der private Schlüssel des `MembershipService` muss mit besonderer Vorsicht behandelt werden, da es unbedingt erforderlich ist, die Identität dieser zentralen Verwaltungsinstanz verifizieren zu können. Wenn ein Angreifer in den Besitz des privaten Schlüssels des `MembershipService` gelangt, ist das gesamte System kompromittiert, da der Angreifer sich gegenüber den Peers als `MembershipService` ausgeben kann.

[2] Mantra bezeichnet ein besonders langes Passwort, beispielsweise einen ganzer Satz mit Satz- und Leerzeichen.

Eine Verschlüsselung des privaten Schlüssels mit einem Mantra führt zu dem Problem, dass bei jedem Neustart des `MembershipService`-Systems ein Administrator das Mantra zur Entschlüsselung eingeben muss. Das Mantra darf nur vertrauenswürdigen Personen bekannt sein, da eine Offenlegung des Mantras ähnliche Risiken beinhaltet wie eine Offenlegung des privaten Schlüssels selbst.

Ein unverschlüsseltes Speichern des privaten `MembershipServer`-Schlüssels ist zwar im produktiven Betrieb praktikabler, da bei einem System-Neustart kein Mantra eingegeben werden muss, das Risiko einer Offenlegung des Schlüssels ist aber höher.

Eine verbindliche Empfehlung zur Sicherung des privaten `MembershipServer`-Schlüssels kann leider nicht gegeben werden. In jedem Fall muss sichergestellt sein, dass aus der Kenntnis des privaten Schlüssels des `MembershipService` eindeutig auf dessen Identität geschlossen werden kann.

5.6.2 Verteilung öffentlicher Schlüssel

Die Verbreitung der öffentlichen Schlüssel muss mit einiger Sorgfalt geschehen, damit die Authentifizierung von `MembershipService` und Peers zweifelsfrei möglich ist. Insbesondere dürfen auch öffentliche Schlüssel nicht über ungesicherte Kanäle verteilt werden, damit die Schlüssel bei der Übertragung nicht manipuliert oder ausgetauscht werden können.

Öffentlicher Schlüssel des `MembershipService`

Der öffentliche Schlüssel des `MembershipService` wird gemeinsam mit der P2P-Anwendung ausgeliefert. Der Fingerprint des Schlüssels sollte vom Benutzer beispielsweise anhand eines schriftlichen Dokumentes überprüft werden.

Öffentlicher Schlüssel eines Gruppenverwalters

Der öffentliche Schlüssel eines Gruppenverwalters muss dem `MembershipService` über einen sicheren Kommunikationskanal zugestellt werden. Beispielsweise kann eine postalisch zugestellte Transaktionsnummer (TAN) verwendet werden, um den Gruppenverwalter einmalig zur Bekanntgabe des öffentlichen Schlüssels gegenüber dem `MembershipService` zu authentifizieren. Von diesem Zeitpunkt an kann der öffentliche Schlüssel zur Authentifizierung verwendet werden.

Öffentliche Schlüssel der Gruppenmitglieder

Die Gruppenmitglieder wiederum authentisieren sich gegenüber ihrem Gruppenverwalter.

Wenn Gruppenverwalter und Benutzer über ein gemeinsames Authentifizierungssystem (beispielsweise einen Kerberos-Server) verfügen, kann dieses System für den authentifizierten Austausch der öffentlichen Schlüssel zwischen Benutzern und Gruppenverwalter verwendet werden.

Wenn ein solches System nicht zur Verfügung steht, müssen die Benutzer sich persönlich unter Vorlage ihres Personalausweises bei ihrem Gruppenverwalter authentisieren. Dabei übergeben sie ihm ihren öffentlichen Schlüssel auf einem Datenträger oder verifizieren gemeinsam mit dem Gruppenverwalter einen per eMail zugestellten Schlüssel. Damit der Benutzer Einladungen des Gruppenverwalters verifizieren kann, erhält er bei dieser Gelegenheit den öffentlichen Schlüssel des Gruppenverwalters.

Durch dieses Verfahren können die Identitäten Benutzer und des `MembershipService` zweifelsfrei mit den jeweiligen öffentlichen Schlüsseln nachgewiesen werden, jeder Peer kann über eine Identität einem Benutzer zugeordnet werden.

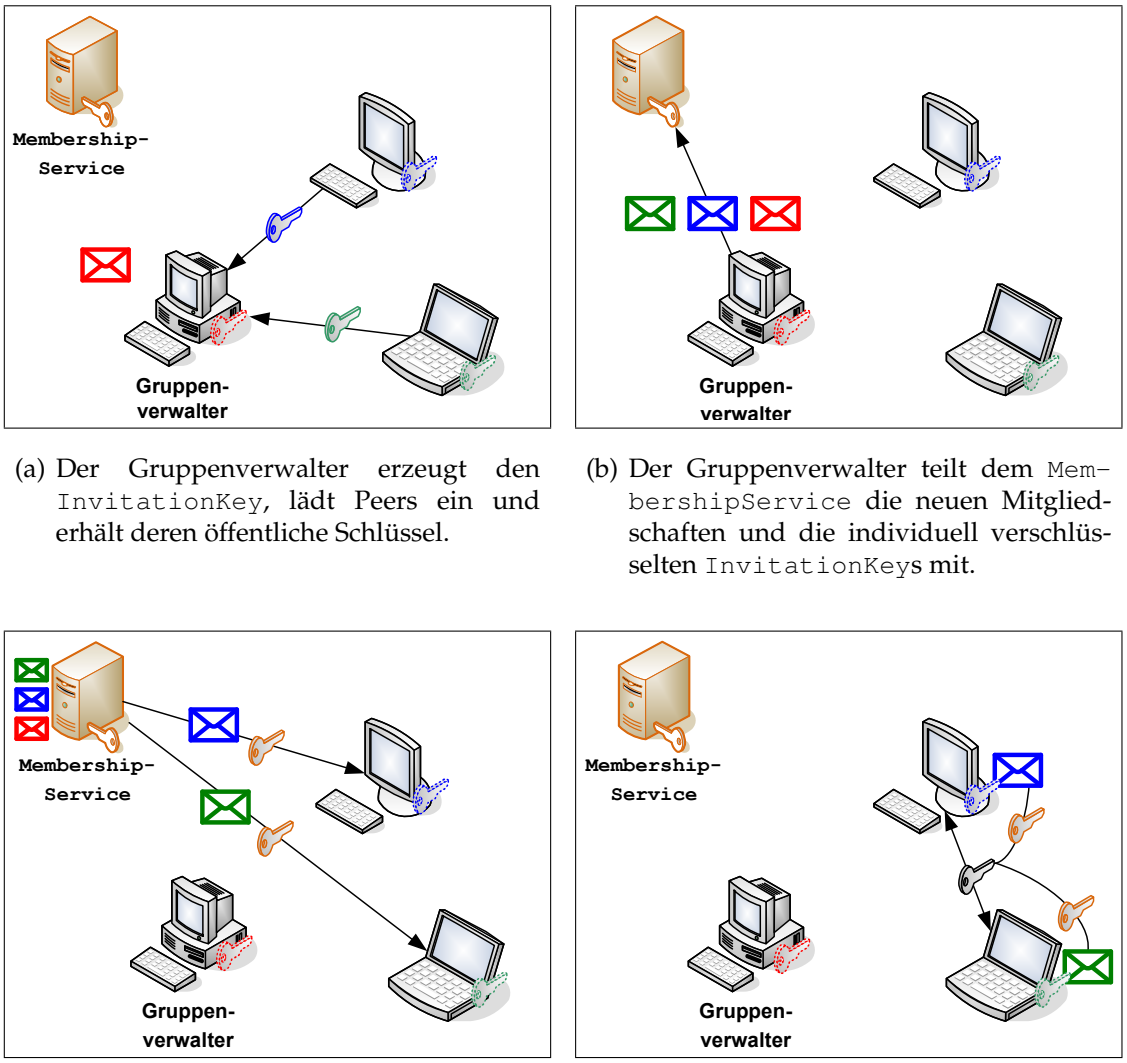
5.6.3 Symmetrischer Schlüssel des `ServiceChannels`

Der symmetrische Schlüssel, der den `ServiceChannel` verschlüsselt, wird zu Beginn der Kommunikation mit einem Diffie-Hellman-Schlüsseltausch ausgehandelt. Damit ist der Schlüssel unabhängig von Langzeitschlüsseln und die Kommunikation bleibt auch vertraulich, wenn die Langzeitschlüssel der teilnehmenden Peers kompromittiert werden sollten.

Der Diffie-Hellman-Schlüssel hat eine kurze Gültigkeitsdauer beispielsweise von wenigen Minuten oder einigen Nachrichten. Nach Ablauf dieser Frist muss der Schlüssel durch einen erneuten Diffie-Hellman-Schlüsseltausch neu ausgehandelt werden. Dies stellt sicher, dass selbst wenn einer der Schlüssel kompromittiert werden sollte, nur ein kleiner Teil der ausgetauschten Nachrichten offen gelegt wird.

Der Peer und der `MembershipService` authentisieren sich jeweils, indem sie den Nachrichten, die beim DHS ausgetauscht werden, eine Signatur anhängen. Der Empfänger kann den Absender authentifizieren, indem er die Signatur verifiziert. Dies schließt eine Veränderung der Nachricht durch Dritte aus und gewährleistet so die Integrität der Nachrichten.

Nach dem Diffie-Hellman-Schlüsseltausch verfügen Peer und `MembershipService` über einen gemeinsamen geheimen Schlüssel.



(a) Der Gruppenverwalter erzeugt den `InvitationKey`, lädt Peers ein und erhält deren öffentliche Schlüssel.

(b) Der Gruppenverwalter teilt dem `MembershipService` die neuen Mitgliedschaften und die individuell verschlüsselten `InvitationKeys` mit.

(c) Die Peers erfragen ihren `InvitationKey` und den aktuellen `MembershipKey` beim `MembershipService`.

(d) Die Peers erzeugen aus dem `InvitationKey` und dem `MembershipKey` den `GroupKey` und kommunizieren verschlüsselt innerhalb der Gruppe.

Abbildung 5.4: Erzeugung des Gruppen-Schlüssels

5.6.4 Symmetrischer Schlüssel des `GroupChannels`

Die Kommunikation der Peers innerhalb einer Gruppe ist ebenfalls durch Verschlüsselung mit einen kurzlebigen symmetrischen Schlüssel geschützt. Dieser Schlüssel heißt **`GroupKey`** und besteht aus den zwei Komponenten `MembershipKey` und `InvitationKey`:

MembershipKey

Der kurzlebige Teil des GroupKeys ist der **MembershipKey**. Er wird periodisch vom MembershipService erzeugt und muss von den Peers einer Gruppe regelmäßig neu erfragt werden. Da der MembershipService nur denjenigen Peers den MembershipKey bekannt macht, die berechtigt sind, an der Gruppenkommunikation teilzunehmen, ist eine weitere Authentifizierung der Peers innerhalb der Gruppe nicht erforderlich; jeder Peer, der den gültigen MembershipKey besitzt, ist garantiert autorisiert, Nachrichten zu senden und zu empfangen.

Es ist zu beachten, dass ein Peer, auch wenn er vom Gruppenverwalter aus der Gruppe ausgeschlossen wurde, weiterhin an der Gruppenkommunikation teilnehmen kann, bis der momentane MembershipKey abgelaufen ist und der MembershipService eine Erneuerung wegen fehlender Mitgliedschaft ablehnt.

InvitationKey

Wenn lediglich der MembershipKey für die Verschlüsselung des GroupChannels verwendet würde, wäre der MembershipService in der Lage, die Gruppenkommunikation abzuhören, da er notwendigerweise über die MembershipKeys verfügt. Aus diesem Grund besitzt der GroupKey als weitere Komponente den **InvitationKey**.

Der InvitationKey ist der Langlebige Teil des GroupKey. Er wird vom Gruppenverwalter beim Anlegen der Gruppe erzeugt und behält seine Gültigkeit über die gesamte Lebensdauer der Gruppe hinweg. Der InvitationKey wird dem MembershipService nicht bekannt gemacht.

Damit der Gruppenverwalter während der gesamten Lebenszeit der Gruppe auf den InvitationKey zugreifen kann, verschlüsselt er ihn zum Schutz vor unberechtigtem Zugriff mit seinem öffentlichen Schlüssel und sendet ihn an den MembershipService, der ihn zusammen mit den Gruppeninformationen speichert. Bei Bedarf kann der Gruppenverwalter die InvitationKeys seiner Gruppen wieder vom MembershipService beziehen.

Jedes Mal, wenn der Gruppenverwalter einen Peer in die Gruppe aufnimmt, verschlüsselt er den InvitationKey mit dem öffentlichen Schlüssel des neuen Mitgliedes. Der so geschützte InvitationKey wird vom Gruppenverwalter an den MembershipService übergeben.

GroupKey

Der MembershipService übermittelt autorisierten Gruppenmitgliedern auf Anfrage den aktuellen MembershipKey und den InvitationKey der Gruppe. Diese

können aus `InvitationKey` und `MembershipKey` den aktuellen **`GroupKey`** generieren und diesen Schlüssel für die sichere Gruppenkommunikation verwenden.

Die einzelnen Schritte der Erzeugung des `GroupKey` sind in Abbildung 5.4 schematisch dargestellt.

5.6.5 Verbindlichkeit des `ServiceChannels`

Über den `ServiceChannel` werden die Nachrichten der Gruppenoperationen übertragen. Die Verbindlichkeit der übertragenen Nachrichten muss dabei gewährleistet sein, damit weder der Peer noch der `MembershipService` nachträglich die Durchführung einer Operation abstreiten können.

Daher werden alle über den `ServiceChannel` übertragenen Nachrichten vom Sender signiert. Sowohl der Sender als auch der Empfänger erstellen einen Log-Eintrag für die Anfrage- und Antwortnachricht jeder durchgeführten Operation. Der Log-Eintrag enthält den Namen des Absenders, die Nachricht und die Signatur der Nachricht.

5.6.6 Wiedereinspielung

Um eine Wiedereinspielung von vorangegangenen Nachrichten zu verhindern, besitzt jede Nachricht einen Zeitstempel. Liegt der Zeitstempel einer empfangen Nachricht außerhalb eines definierten Zeitintervalls oder wurde bereits eine Nachricht mit dem gleichen Zeitstempel empfangen, wird die Nachricht verworfen.

5.6.7 Sicherheitsschicht

Die Kommunikationskanäle `ServiceChannel` sowie `GroupChannel` sind in Schichten gegliedert, wie in Abbildung 5.5 zu sehen ist.

`ServiceChannel`

Wenn sich ein Peer mit dem P2P-System verbindet, wird zunächst über das P2P-Overlay eine Verbindung mit dem `MembershipService` hergestellt. Für diesen Kommunikationskanal, den `ServiceChannel`, handeln Peer und `MembershipService` auf der Ebene der Sicherheitsschicht (engl. `security layer`) mit dem Diffie-Hellman-Schlüsseltausch-Verfahren einen kurzlebigen Schlüssel aus.

Zu diesem Zweck führen die `DiffieHellmanRequester`-Komponente des Peers und die `DiffieHellmanResponder`-Komponente des `MembershipService` einen Diffie-Hellman-Schlüsseltausch durch. Die dafür benötigten Daten

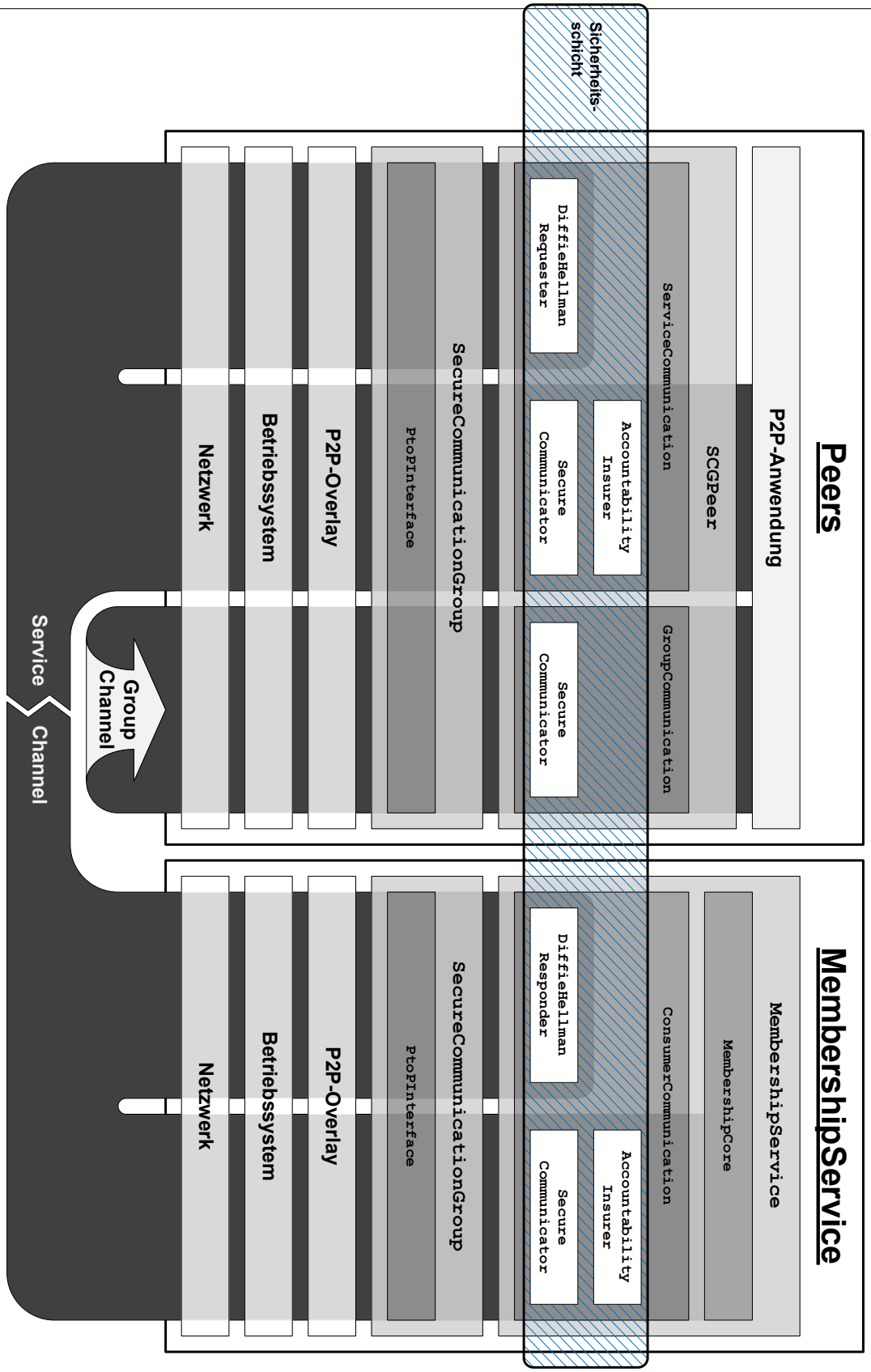


Abbildung 5.5: Kommunikationskanäle und -schichten

werden in DiffieHellman-Nachrichten übermittelt. Nach dem Diffie-Hellman-Schlüsseltausch besitzen Peer und MembershipService einen gemeinsamen symmetrischen Schlüssel, den **DiffieHellmanKey**.

Nun kann der Peer über den ServiceChannel Gruppenoperationen durchführen, das heißt Anfragen mit der gewünschten Gruppenoperation an den MembershipService senden und Antworten empfangen.

Die Anfragen werden von der Anwendungsschicht an die Sicherheitsschicht übergeben. Dort durchläuft die Anfrage die folgenden Sicherheitskomponenten:

- Zunächst wird die Anfrage von der AccountabilityInsurer-Komponente bearbeitet. Diese Komponente erstellt einen Log-Eintrag für die Anfrage, signiert sie und bettet sie in eine Accountability-Nachricht ein.
- Die Accountability-Nachricht wird an die SecureCommunicator-Komponente übergeben. Diese Komponente verschlüsselt die Nachricht und verpackt die Chiffre in eine Secrecy-Nachricht.

Die Secrecy-Nachricht wird über das P2P-Overlay an den MembershipService zugestellt.

Der MembershipService entschlüsselt die Nachricht in seiner SecureCommunicator-Komponente und übergibt sie an die AccountabilityInsurer-Komponente. Diese überprüft die Signatur und erstellt einen Log-Eintrag, der den Namen des Absenders, den Inhalt der Nachricht und die Signatur enthält, um die Anfrage nachweisen zu können.

Antworten werden vom MembershipService auf die gleiche Weise signiert, verschlüsselt und über das Overlay an den Peer zugestellt. Auch der Peer überprüft die Signatur und erstellt einen Log-Eintrag für jede erhaltene Antwort.

GroupChannel

Nachdem der Peer die aktuellen Gruppenschlüssel in Erfahrung gebracht hat, kann er sich über die jeweiligen GroupChannels an der sicheren Grupppenkommunikation seiner Gruppen beteiligen. Alle Nachrichten, die über diese Kommunikationskanäle versendet werden, werden in der Sicherheitsschicht mit einem GroupKey verschlüsselt und von einer SecureCommunicator-Komponente in Secrecy-Nachrichten verpackt.

5.6.8 Erneuerung der symmetrischen Schlüssel

Secrecy-Nachrichten sind je nach Kommunikationskanal mit einem DiffieHellmanKey oder einem GroupKey verschlüsselt:

- **GroupKey:** Wenn bei der Verschlüsselung des `GroupChannels` in der Sicherheitsschicht festgestellt wird, dass die `MembershipKey`-Komponente des `GroupKeys` abgelaufen ist, wird der aktuelle Schlüssel vom `MembershipService` angefordert. Zu diesem Zweck wird über den `ServiceChannel` eine Anfrage nach dem benötigten Schlüssel an den `MembershipService` gesendet.
- **DiffieHellmanKey:** Stellt der Peer beim Verschlüsseln einer Anfrage an den `MembershipService` in der Sicherheitsschicht des `ServiceChannels` fest, dass der `DiffieHellmanKey` abgelaufen ist, wird erneut ein Diffie-Hellman-Schlüsseltausch angestoßen.

Auf diese Weise beschaffen sich die Peers dynamisch die von ihnen benötigten Schlüssel und können damit kontinuierlich an der sicheren Kommunikation ihrer Gruppen teilnehmen.

5.6.9 Nachrichtenformate der Sicherheitsschicht

In der Sicherheitsschicht werden das `DiffieHellman`-, das `Accountability`- und das `Secrecy`-Nachrichtenformat verwendet:

`DiffieHellman`-Nachricht

Der Diffie-Hellman-Schlüsseltausch wird über **`DiffieHellman`-Nachrichten** abgewickelt. Das Nachrichtenformat einer `DiffieHellman`-Nachricht besitzt folgende Elemente:

`TimeStamp`: Zeitstempel der Nachricht zur Vereitelung von Wiedereinspielungsangriffen.

`KeyExchange`: Das `KeyExchange`-Element enthält das Diffie-Hellman-Geheimnis[3] des jeweiligen Senders.

`(Expiry)`: Die Gültigkeitsdauer des Schlüssels wird vom `MembershipService` bestimmt und mit dem öffentlichen Schlüssel des Peers verschlüsselt an den Peer gesendet. Wenn ein Peer eine `DiffieHellman`-Nachricht an den `MembershipService` sendet, entfällt dieses Element.

`Signature`: Mit der Signatur wird der Kommunikationspartner authentifiziert. Der in der Signatur enthaltene verschlüsselte Hash-Wert umfasst alle anderen Nachrichtenelemente.

Wie in Abschnitt 2.5.3 beschrieben, werden für den Schlüsseltausch eine große Primzahl p und eine Primitivwurzel g von $\mathbb{Z}_p^*$ benötigt. Da diese Parameter öffentlich

[3] vgl. Abschnitt 2.5.3: Diffie-Hellman-Schlüsseltausch

bekannt sein dürfen und nicht variieren müssen, wird stets dieselbe Primzahl mit passender Primitivwurzel verwendet. Daher müssen sie nicht in den Nachrichten übertragen werden.

`Accountability`-Nachricht

Durch das `Accountability`-Nachrichtenformat wird die Verbindlichkeit des `ServiceChannels` garantiert.

Eine **`Accountability`-Nachricht** hat folgende Elemente:

`AccountedMessage`: Dieses Element enthält die Nachricht, die verbindlich versendet werden soll.

`Signature`: Durch der Signatur der eingebetteten Nachricht kann der Urheber der Nachricht nachgewiesen werden.

`Secrecy`-Nachricht

Die Nachrichten, die die Peers über den `ServiceChannel` und den `GroupChannel` austauschen, werden in der Sicherheitsschicht verschlüsselt und in **`Secrecy`-Nachrichten** eingebettet.

Das Nachrichtenformat einer `Secrecy`-Nachricht definiert nur ein Element, das **`Cipher`**-Element. Dieses Element enthält die mit dem `DiffieHellmanKey` respektive dem `GroupKey` verschlüsselte Nachricht der höheren Schicht. Durch die Verschlüsselung ist die Vertraulichkeit der Nachricht gewährleistet.

Das `Secrecy`-Nachrichtenformat sieht kein Element für eine Signatur vor. Eine Signatur wird nicht benötigt, da jede Manipulation der Chiffre dazu führt, dass das `Cipher`-Element nicht mehr mit dem symmetrischen Schlüssel entschlüsselt werden kann. Somit sind allein durch die Verwendung des symmetrischen Schlüssels die Vertraulichkeit und Integrität der Nachricht gesichert.

Wenn eine `Secrecy`-Nachricht im `GroupChannel` verwendet wird, besteht keine Notwendigkeit, den Absender der Nachricht zu authentifizieren, da er offensichtlich über den geheimen Gruppenschlüssel verfügt und sich daher garantiert zuvor beim `MembershipService` authentisiert hat.

Um zusätzlich Wiedereinspielungsangriffe zu verhindern, benötigt auch eine `Secrecy`-Nachricht einen Zeitstempel. Dieser kann nicht als zusätzliches Nachrichtenelement übertragen werden, da auch er vor Manipulation geschützt werden muss. Daher enthält das `Cipher`-Element neben der verschlüsselten Nachricht einen zusammen mit der Nachricht verschlüsselten Zeitstempel, der auf diese Weise sicher übertragen werden kann. Bei der Entschlüsselung wird der Zeitstempel nach seiner Überprüfung verworfen.

6 Implementierung

In diesem Kapitel werden die wichtigsten Mechanismen vorgestellt, die bei der Implementierung der „Secure Communication Group“ (SCG) verwendet wurden.

Damit die Bibliothek den Anforderungen der Firma SYNCING.NET entsprechend plattformunabhängig eingesetzt werden kann, wurde sie für die .NET-Laufzeitumgebung implementiert. Es gibt Versionen der .NET-Laufzeitumgebung für die Betriebssysteme Windows durch das *.NET Framework*[1] von Microsoft sowie Linux, Solaris und MacOS durch das OpenSource-Projekt *Mono*[2].

SCG setzt auf die JXTA-C-Bibliothek[3] auf. JXTA-C implementiert das JXTA-Protokoll in der Programmiersprache C. Da JXTA-C die Apache Portable Runtime (APR)[4] verwendet, ist auch JXTA-C auf den oben genannten Systemen einsetzbar.

Als Programmiersprache für die Implementierung von SCG wurde C# gewählt, da diese Sprache eine der modernsten objektorientierten Hochsprachen ist und speziell für die .NET-Laufzeitumgebung entwickelt wurde.

6.1 Kryptographische Algorithmen

Die Implementierungen der kryptographischen Algorithmen greifen auf Klassen der .NET-Laufzeitumgebung zurück und stellen eine vereinfachte Schnittstelle zur Verwendung der .NET-Sicherheitsmechanismen zur Verfügung.

6.1.1 Symmetrische Schlüssel

Für die symmetrische Verschlüsselung wird das Kryptosystem Advanced Encryption Standard (AES) verwendet. Die .NET-Laufzeitumgebung implementiert dieses Kryptosystem in der Klasse `System.Security.Cryptography.ManagedRijndael`.

Da die symmetrischen Schlüssel von SCG eine begrenzte Gültigkeitsdauer haben, wird der Zugriff auf die Klasse `ManegedRijndael` von der abstrakten Klasse `SymmetricKey` gekapselt, die zusätzliche Eigenschaften besitzt.

[1] `www.microsoft.com/germany/msdn/netframework/default.mspx`
[2] `www.mono-project.com`
[3] `jxta-c.jxta.org/`
[4] `apr.apache.org`

Die Klasse `SymmetricKey` hat die folgende Schnittstelle:

```
public abstract class SymmetricKey
{
  public abstract void RefreshKey();

  public DateTime Expiry { get; }
  public bool IsExpired { get; }

  internal ICryptoTransform Encryptor { get; }
  internal ICryptoTransform Decryptor { get; }
}
```

Die Methode `RefreshKey` wird aufgerufen, wenn beim Zugriff auf den symmetrischen Schlüssel festgestellt wird, dass dieser abgelaufen ist; diese Funktionalität muss von einer Unterklasse definiert werden. Die Eigenschaften `Expiry` und `IsExpired` geben Auskunft über die Gültigkeit des Schlüssels. Die Eigenschaften `Encryptor` und `Decryptor` kapseln den Zugriff auf die Klasse `ManagedRijndael` und ermöglichen die Verschlüsselung bzw. Entschlüsselung von Daten.

Die abstrakte Klasse `SymmetricKey` wird von folgenden Klassen spezialisiert:

`GroupKey`: Die Peers verwenden den `GroupKey` für die Verschlüsselung des Gruppenkanals. Wenn der Schlüssel abgelaufen ist, wird von der `RefreshKey`-Methode die Ausführung der `GetGroupKeys`-Operation angestoßen.

`DiffieHellmanKey`: Der `DiffieHellmanKey` wird für die Verschlüsselung des Servicekanals verwendet. Wenn er abgelaufen ist, wird von der `RefreshKey`-Methode ein Diffie-Hellman-Schlüsseltausch durchgeführt.

6.1.2 Asymmetrische Schlüssel

Als asymmetrisches Kryptosystem wird RSA verwendet. RSA wird von der .NET-Laufzeitumgebung in der Klasse `System.Security.Cryptography.RSACryptoServiceProvider` implementiert. Der Zugriff auf die RSA-Schlüssel wird von der Klasse `AsymmetricKey` gekapselt.

Diese Klasse bietet folgende Schnittstelle an:

```
public class SymmetricKey
{
  public byte[] Sign(params byte[][] Objects);
  public void Verify(byte[] Signature, params byte[][] Objects);

  public byte[] Encrypt(byte[] PlainText);
  public byte[] Decrypt(byte[] Cipher);
```

```
  public RSAParameters PublicKey { get; };
}
```

Die Methode `Sign` signiert alle übergebenen Objekte mit dem privaten Schlüssel. Mit der Methode `Verify` kann eine Signatur mit dem öffentlichen Schlüssel verifiziert werden.

`Encrypt` ermöglicht das Verschlüsseln mit dem öffentlichen, `Decrypt` das Entschlüsseln mit dem privaten Schlüssel.

Der öffentliche Schlüssel kann mit der Eigenschaft `PublicKey` ausgelesen werden.

Die Klasse `SymmetricKey` muss nicht notwendigerweise über einen privaten Schlüssel verfügen. Die Methoden `Verify` und `Encrypt` können beispielsweise mit dem öffentlichen Schlüssel eines Kommunikationspartners verwendet werden.

6.1.3 Schlüsselbund

Die asymmetrischen Schlüssel eines Peers werden von der Klasse `KeyStore` (engl. für: Schlüssellager, Schlüsselbund) verwaltet. Diese Klasse implementiert die folgende Schnittstelle:

```
public class KeyStore
{
  public void Load(Stream InStream, SecureString Mantra);
  public void Save(Stream OutStream, SecureString Mantra);

  public AsymmetricKey NativeKey { get; }

  public AsymmetricKey this[string Name] { get; }

  public void AddForeignKey(SecureString Mantra, RSA Key,
                            string Name);
}
```

Der `KeyStore` kann mit der Methode `Load` geladen und mit der Methode `Save` gespeichert werden. Zum Schutz vor Manipulation werden alle öffentlichen Schlüssel mit dem privaten Schlüssel des Peers signiert. Der private Schlüssel wird mit dem angegebenen Mantra verschlüsselt.

Die Eigenschaft `NativeKey` gewährt Zugriff auf das eigene Schlüsselpaar des Peers.

Die öffentlichen Schlüssel anderer Peers werden über die Namen der Peers indiziert. Der Zugriff erfolgt über die `this[]`-Eigenschaft der Klasse.

Schlüssel anderer Peers können über die Methode `AddForeignKey` zum Schlüsselbund hinzugefügt werden. Beim Hinzufügen eines Schlüssels muss das Mantra angegeben werden.

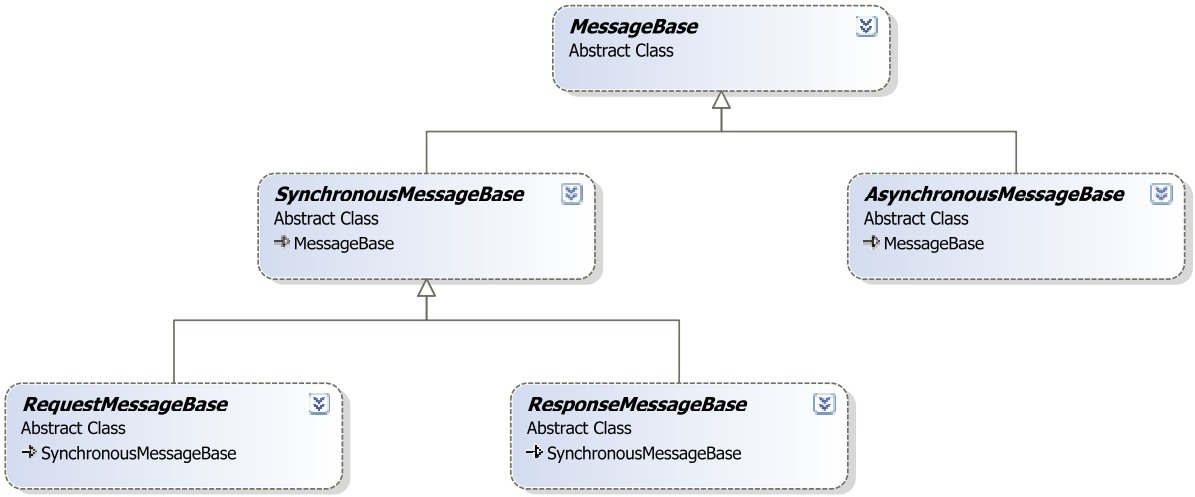


Abbildung 6.1: Hierarchie der abstrakten Nachrichtenklassen

6.1.4 Diffie-Hellman-Schlüsseltausch

Eine Implementierung des Diffie-Hellman-Schlüsseltausches wird vom Mono-Projekt[5] mit der Klasse `Mono.Security.Cryptography.DiffieHellman` angeboten. Der Zugriff auf die Implementierung wird von der Klasse `DiffieHellmanKey` gekapselt.

6.1.5 Hash-Algorithmus

Als Hash-Algorithmus wird der Secure Hash Algorithm (SHA) verwendet. Die .NET-Laufzeitumgebung bietet verschiedene Versionen dieses Algorithmus an, die sich hinsichtlich der Länge des berechneten Hash-Wertes unterscheiden. In den meisten Fällen werden 256-Bit-Hash-Werte verwendet, die mit der Klasse `System.Security.Cryptography.SHA256Managed` berechnet werden.

6.2 Nachrichten

Alle Nachrichten, die für die Gruppenoperationen oder für die Sicherheitsschicht benötigt werden, sind als Unterklassen der abstrakten Klasse `MessageBase` implementiert. Die Nachrichtenelemente werden dabei als öffentliche Attribute der Klasse abstrahiert. Die Namen der Nachrichtenelemente entsprechen den Attributnamen, der Inhalt der Elemente wird in den Attributen gespeichert.

Abbildung 6.1 zeigt die allgemeine Hierarchie der Nachrichtenklassen. Von der abstrakten Klasse `MessageBase` sind zunächst zwei weitere abstrakte Klassen abgelei-

[5] `www.mono-project.com`

tet, die zwischen synchron und asynchron versendeten Nachrichten differenzieren. Diese Klassen heißen `SynchronousMessageBase` und `AsynchronousMessageBase`.

Die abstrakte Klasse `SynchronousMessageBase` besitzt zwei weitere abstrakte Unterklassen `RequestMessageBase` und `ResponseMessageBase` zur Differenzierung von Anfrage- und Antwort-Nachrichten.

Die Nachrichten aller Nachrichtenformate sind wie in den Abbildungen 6.2 und 6.3 dargestellt in diese Klassenhierarchie eingeordnet.

Die `CreateSubGroupRequest`-Nachricht mit den Nachrichtenelementen `InvitationKey`, `InvitationSignature` und `Group`[6] ist beispielsweise folgendermaßen implementiert:

```
public class CreateSubGroupRequest : RequestMessageBase
{
  public byte[] InvitationKey;
  public byte[] InvitationSignature;
  public Group Group;
}
```

Damit die Nachrichten verschlüsselt oder über das Netz übertragen werden können, müssen sie in einen Datenstrom umgewandelt werden. Dazu werden sie mit Hilfe der Klasse `System.Xml.XmlSerializer` aus der .NET-Laufzeitumgebung in ein XML-Dokument serialisiert[7]. Das XML-Dokument, das aus der Nachricht generiert wird, enthält alle öffentlichen Attribute der Klasse (und damit alle Nachrichtenelemente) als XML-Elemente.

Die XML-Serialisierung einer `CreateSubGroupRequest`-Nachricht kann beispielsweise wie folgt aussehen:

```
<CreateSubGroupRequest>
  <InvitationKey encoding="Base64">
    gDX+fyXoaIkXDaGzWAnoaNpjsyPqZdJ4LZ79Lqzlwpp
   </InvitationKey>
  <InvitationSignature encoding="Base64">
    k4tSe0hjuFXlK9vThzJNdIKG4gf1vGtrQKgfJIiiuG5
  </InvitationSignature>
  <Group>
    <Name>BabelFishUserGroup</Name>
    <ParentGroup>UniverseTravalingGroup</ParentGroup>
  </Group>
</CreateSubGroupRequest>
```

[6] vgl. Abschnitt 5.5.2: Gruppenoperationen
[7] Serialisierung bezeichnet das Abbilden eines Objektes auf eine serielle Darstellungsform.

Abbildung 6.2: Hierarchie der Nachrichtenformate Teil 1

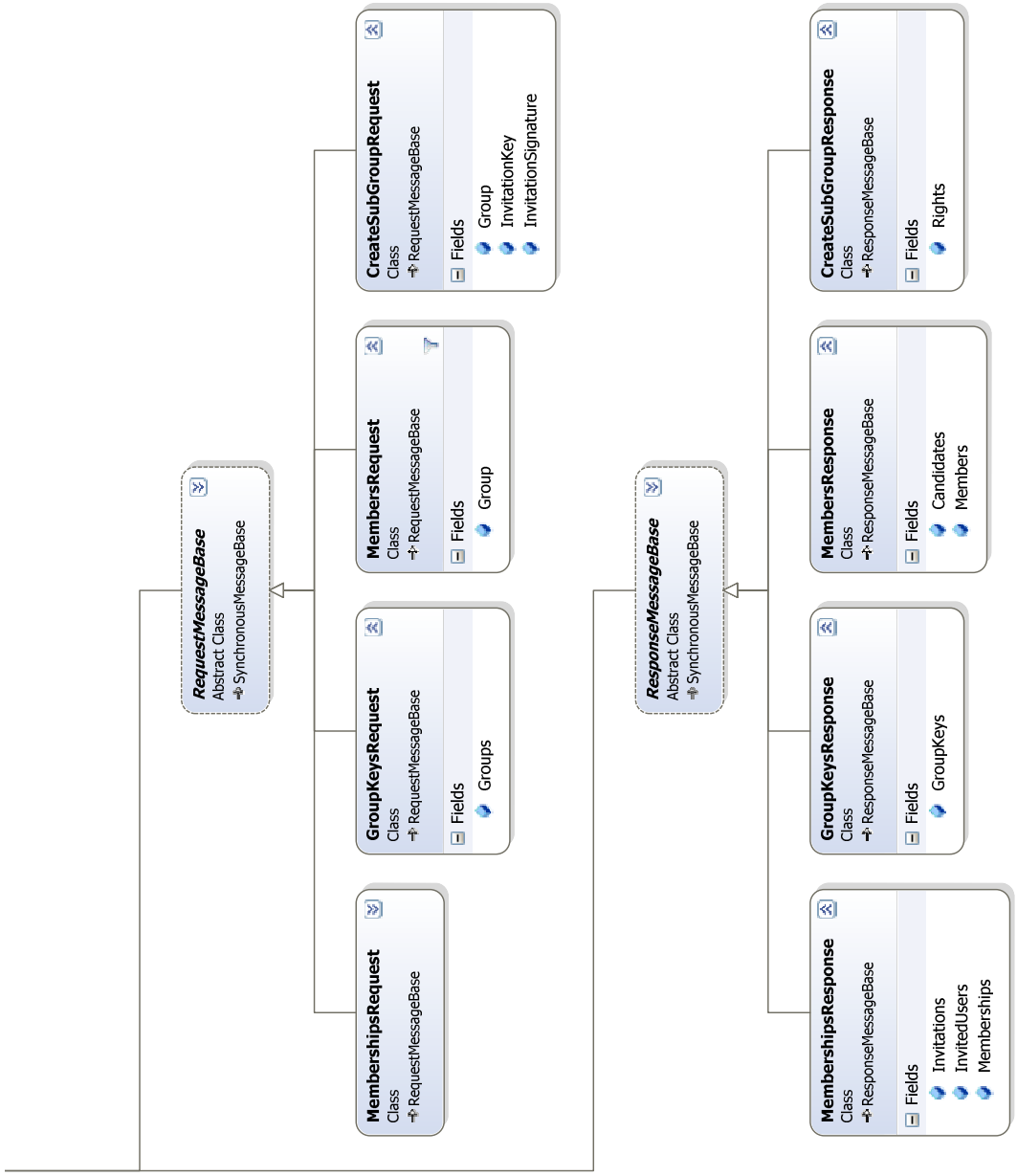


Abbildung 6.3: Hierarchie der Nachrichtenformate Teil 2

6.3 Sicherheitsschicht

Das Rückrat von SCG bilden die Nachrichtenkanäle `GroupChannel` und `ServiceChannel`, über die die Nachrichten der Gruppenkommunikation respektive der Gruppenoperationen übertragen werden. Diese Kanäle sind, wie in Abschnitt 5.6.7 beschrieben, in mehrere Schichten gegliedert.

Die Sicherheitsschicht ist für die Verbindlichkeit der Kommunikation, die Verschlüsselung der Nachrichten und die Erzeugung der symmetrischen Kommunikationsschlüssel zuständig. Sie ist über eine Programmierschnittstelle mit dem darunter liegenden P2P-Overlay verbunden. Entschlüsselte Nachrichten werden je nach Format an unterschiedliche Module in der darüber liegenden Anwendungsschicht weitergereicht; von dort eingehende Nachrichten werden verschlüsselt versendet.

Die Kommunikation über das P2P-Overlay verläuft asynchron, die Gruppenoperationen basieren jedoch auf dem Frage-Antwort-Prinzip, sind also synchron. Die Sicherheitsschicht muss daher einen Mechanismus anbieten, der eine synchrone Kommunikation über asynchrone Nachrichtenkanäle ermöglicht.

Um diese unterschiedlichen Aufgaben der Sicherheitsschicht strukturiert zu implementieren, wurde eine Sammlung von Schnittstellen und Klassen entwickelt, die die einzelnen Aspekte der Schicht kapseln. Aus diesen Modulen wird die Funktionalität der Sicherheitsschicht zusammengesetzt. Die Module der Sicherheitsschicht heißen `MessageHandler`.

In den folgenden beiden Abschnitten werden die `MessageHandler` der Peers und des `MembershipService` beschrieben. Darauf folgen Abschnitte, die den Aufbau der verschiedenen `MessageHandler` eingehender erläutern.

6.3.1 Sicherheitsschicht der Peers im Detail

Abbildung 6.4 stellt die Komponenten der Sicherheitsschicht auf der Seite des Peers dar. Die Sicherheitsschicht wird vertikal von `GroupChannel` und `ServiceChannel` geschnitten.

GroupChannel

Der Austausch von Nachrichten über den `GroupChannel` ist gänzlich asynchron. Der Kanal besitzt nur einen `MessageHandler`: den `SecureCommunicator`. Der **`SecureCommunicator`** verschlüsselt aus- und entschlüsselt eingehende Nachrichten mit dem `GroupKey`.

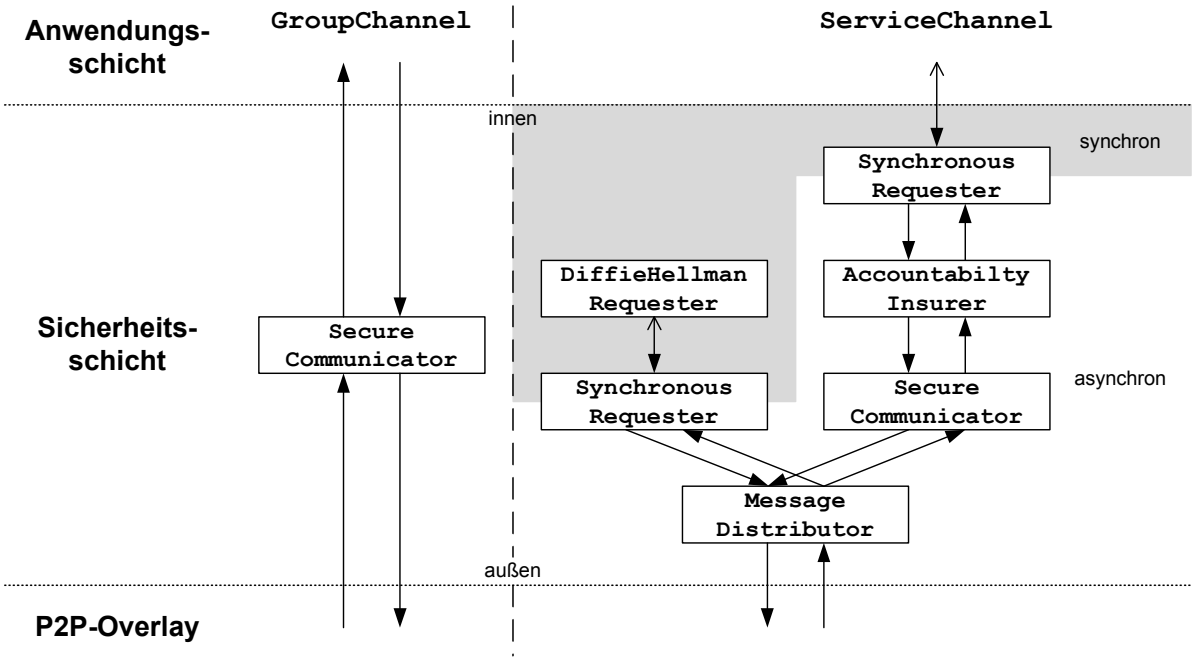


Abbildung 6.4: Peer-seitige Implementierung der Sicherheitsschicht

ServiceChannel

Der `ServiceChannel` ist aus mehreren `MessageHandler`n aufgebaut und in einen synchronen sowie einen asynchronen Abschnitt unterteilt. Er sendet die Anfragen seines Peers an den `MembershipService` und erhält dessen Antworten. Die Nachrichten fließen daher zunächst von innen nach außen und später wieder von außen nach innen zurück.

Innerhalb der Sicherheitsschicht besitzt der `ServiceChannel` zwei Nachrichtenströme: den Diffie-Hellman-Strom, der für den Diffie-Hellman-Schlüsseltausch zuständig ist, und den Gruppenoperations-Strom, über den die Nachrichten der Gruppenoperationen übertragen werden.

Ausgehende Nachrichten werden folgendermaßen behandelt:

1. Diffie-Hellman-Strom:

Über den Diffie-Hellman-Nachrichtenstrom wird mit Hilfe von `DiffieHellman`-Nachrichten ein symmetrischer Schlüssel für die Kommunikation ausgehandelt.

Wenn ein Diffie-Hellman-Schlüsseltausch durchgeführt werden soll, übergibt der **`DiffieHellmanRequester`** eine `DiffieHellman`-Nachricht an einen unter ihm liegenden `SynchronousRequester`.

Der **`SynchronousRequester`** ist dafür zuständig, synchrone auf asynchrone Kommunikation umzusetzen. Zu diesem Zweck bettet er die erhaltene Nach-

richt in eine `Synchronization`-Nachricht ein.

Das **`Synchronization-Nachrichtenformat`** besitzt ein Element `WrappedMessage`, das die synchronisiert zu sendende Nachricht enthält, sowie eine Element `MessageID`, in dem eine eindeutige ID für den synchronen Nachrichtenaustausch übertragen wird. Eine Antwort-Nachricht hat immer die gleiche ID wie die Anfrage-Nachricht; auf diese Weise kann eine Antwort ihrer Anfrage zugeordnet werden.

Der `SynchronousRequester` gibt die eingebettete Anfrage nach außen weiter und pausiert so lange, bis er eine Antwort des `MembershipService` zugestellt bekommt.

2. Gruppenoperations-Strom:

Über den Gruppenoperations-Strom werden Nachrichten der Gruppenoperationen verschlüsselt an den `MembershipService` übertragen.

Die Anfrage-Nachricht einer Gruppenoperation wird von der Anwendungsschicht an die Sicherheitsschicht übergeben. Sie wird von einem weiteren **`SynchronousRequester`** in eine `Synchronization`-Nachricht eingebettet und an einen `AccountabilityInsurer` weitergereicht.

Der **`AccountabilityInsurer`** signiert die Nachricht und erstellt einen Log-Eintrag, der die Signatur und den Inhalt der Nachricht enthält. Dann bettet er die Nachricht zusammen mit der Signatur in eine `Accountability`-Nachricht ein und übergibt die Nachricht an den `SecureCommunicator`.

Der **`SecureCommunicator`** verschlüsselt die Nachricht und gibt eine `Secrecy`-Nachricht, die die Chiffre enthält, nach außen weiter.

Die beiden Nachrichtenströme laufen in einem **`MessageDistributor`** zusammen. Ausgehende Nachrichten, die er vom `SynchronousRequester` des Diffie-Hellman-Stroms oder vom `SecureCommunicator` des Gruppenoperations-Stroms erhält, leitet er an das P2P-Overlay weiter.

Eingehende Nachrichten werden abhängig von ihrem Format an die weiter innen liegenden `MessageHandler` übergeben: Nachrichten im `Synchronization`-Format erhält der `SynchronousRequester` des Diffie-Hellman-Stroms, Nachrichten im `Secrecy`-Format gelangen zum `SecureCommunicator` des Gruppenoperations-Stroms.

1. Diffie-Hellman-Strom:

Wenn die Nachrichten-ID einer eingehenden Antwort-Nachricht zu der ID der Anfrage passt, übergibt der **`SynchronousRequester`** die Antwort an den `DiffieHellmanRequester`.

Der **`DiffieHellmanRequester`** erzeugt aus dem erhaltenen Diffie-Hellman-Geheimnis den symmetrischen Schlüssel für die Verschlüsselung des `ServiceChannel`.

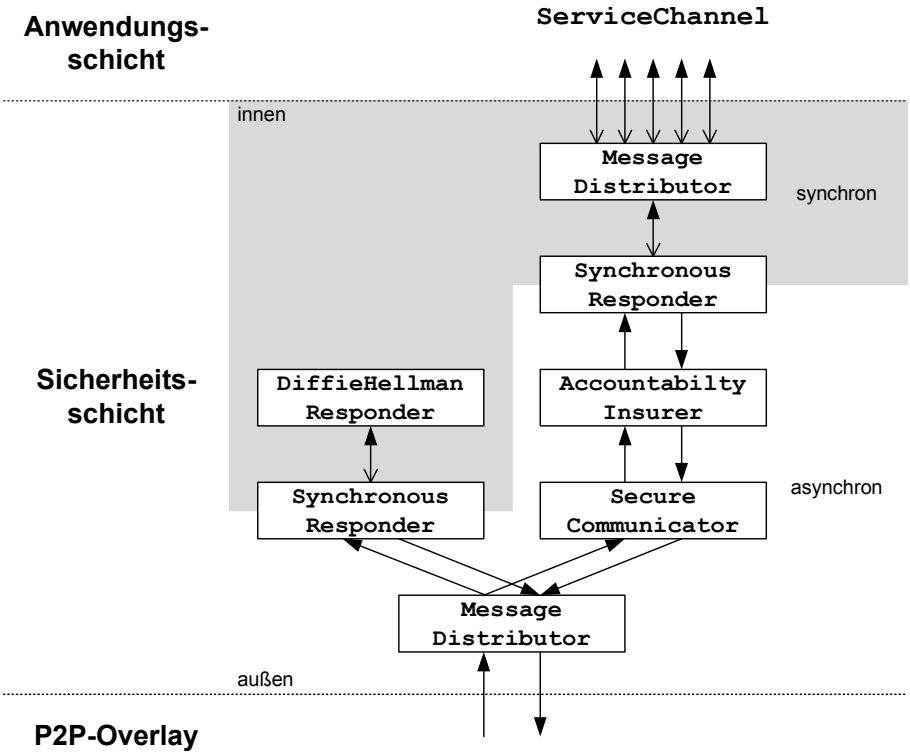


Abbildung 6.5: `MembershipService`-Implementierung der Sicherheitsschicht

2. Gruppenoperations-Strom:

Eingehende Nachrichten werden vom **`SecureCommunicator`** entschlüsselt und an den `AccountabilityInsurer` übergeben.

Der **`AccountabilityInsurer`** überprüft die Signatur der Nachricht. Wenn er die Signatur verifizieren kann, erstellt er einen Log-Eintrag, der den Namen des Absenders, die Signatur und den Inhalt der Nachricht enthält. Dann gibt er die eingebettete Nachricht an den `SynchronousRequester` weiter.

Der **`SynchronousRequester`** stellt die Antwort-Nachricht unter Berücksichtigung ihrer ID an den Ausgangspunkt der Anfrage zu.

Durch dieses System werden Nachrichten definiert durch die Nachrichtenströme geleitet und von den zuständigen `MessageHandlern` behandelt.

6.3.2 Sicherheitsschicht des `MembershipService` im Detail

Der `ServiceChannel` des `MembershipService` nimmt Anfrage-Nachrichten der Peers entgegen, führt die angeforderte Gruppenoperation durch und sendet eine Antwort-Nachricht zurück. Die Nachrichten fließen daher zunächst von außen nach innen und später wieder von innen nach außen zurück. Abbildung 6.5 zeigt die `MessageHandler` der Sicherheitsschicht.

Auch der `ServiceChannel` des `MembershipService` ist in die Nachrichtenströme Diffie-Hellman-Strom und Gruppenoperations-Strom unterteilt.

Die unterste Komponente der Sicherheitsschicht ist ein **`MessageDistributor`**. Dieser speist eingehende Nachrichten in den zuständigen Strom ein: Eingehende Nachrichten des `Synchronization`-Formates werden an den `SynchronousResponder` des Diffie-Hellman-Stromes übergeben, Nachrichten des `Secrecy`-Formates erhält der `SecureCommunicator` des Gruppenoperations-Stromes.

1. Diffie-Hellman-Strom:

Der **`SynchronousResponder`** entpackt die in die `Synchronization`-Nachricht eingebettete `DiffieHellman`-Nachricht und übergibt sie synchronisiert an den `DiffieHellmanResponder`.

Vom **`DiffieHellmanResponder`** wird der Diffie-Hellman-Schlüsseltausch durchgeführt, eine Antwort-Nachricht für den Peer erzeugt und diese an den `SynchronousResponder` zurück gegeben.

Die Antwort wird vom `SynchronousResponder` in eine `Synchronization`-Nachricht mit der ID der Anfrage verpackt und über den `MessageDistributor` an den Peer zurückgesendet.

2. Gruppenoperations-Strom:

Der **`SecureCommunicator`** entschlüsselt die Chiffre der `Secrecy`-Nachricht und übergibt die entschlüsselte Nachricht an den `AccountabilityInsurer`.

Der **`AccountabilityInsurer`** verifiziert die Signatur der Nachricht, erstellt einen Log-Eintrag und reicht die Nachricht an den `SynchronousResponder` des Gruppenoperations-Stromes weiter.

Der **`SynchronousResponder`** entnimmt der `Synchronization`-Nachricht die eingebettete Gruppenoperations-Nachricht und übergibt sie synchron an einen weiteren, inneren `MessageDistributor`.

Der innere **`MessageDistributor`** stellt die Nachricht an den für die Operation zuständigen `MessageHandler` der Anwendungsschicht zu und erhält von diesem eine Antwort-Nachricht. Die Antwort-Nachricht wird vom `MessageDistributor` an den `SynchronousResponder` zurück gegeben.

Der `SynchronousResponder` packt die ausgehende Nachricht in eine `Synchronization`-Nachricht mit der ID der Anfrage ein und leitet sie an den `AccountabilityInsurer` weiter.

Der `AccountabilityInsurer` signiert die ausgehende Nachricht und erstellt einen Log-Eintrag. Er erzeugt eine `Accountability`-Nachricht, die die Signatur und die Antwort enthält, und übergibt diese an den `SecureCommunicator`.

Der `SecureCommunicator` verschlüsselt die Nachricht und übergibt sie dem äußeren `MessageDistributor`, der sie schließlich über das P2P-Overlay ver-

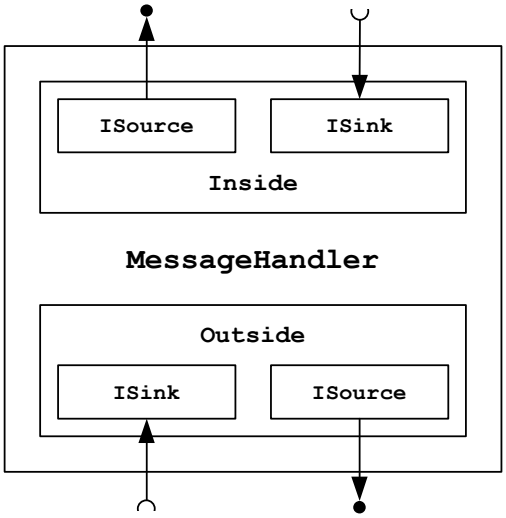


Abbildung 6.6: Komponenten eines asynchronen `MessageHandler`s

sendet.

6.3.3 `MessageHandler` der asynchronen Kommunikation

Die `MessageHandler` der asynchronen Kommunikation bestehen aus folgenden Bestandteilen:

Der Teil eines `MessageHandler`s, der Nachrichten von einem weiter außen gelegenen `MessageHandler` entgegen nimmt bzw. Nachrichten nach außen sendet, wird durch eine abstrakte Klasse mit Namen `Outside` repräsentiert. Der nach Innen gerichtete Teil eines `MessageHandler`s wird durch die abstrakte Klasse `Inside` repräsentiert.

Die Komponenten `Inside` und `Outside` bestehen ihrerseits aus den kleinsten Bestandteilen der asynchronen Kommunikation, nämlich den Schnittstellen `ISink` (*sink* engl. für: Senke) und `ISource` (*source* engl. für: Quelle). Sie stellen einen Nachrichteneingang respektive -ausgang dar.

Abbildung 6.6 zeigt die Komponenten eines `MessageHandler`s.

`ISink`

Die Schnittstelle `ISink` bietet folgende Funktionen an:

```
public interface ISink
{
    void Handle(AsynchronousMessage Message);
```

```
    ICollection<Type> MessageFormats { get; }
}
```

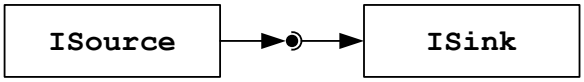
Mit der Methode `Handle` nimmt eine Senke Nachrichten für eine Bearbeitung oder Weiterleitung durch den `MessageHandler` entgegen.

Die Eigenschaft `MessageFormats` gibt eine Liste der Nachrichtenformate zurück, die von dieser Senke angenommen und durch ihren `MessageHandler` bearbeitet werden können.

ISource

Die Schnittstelle `ISource` ermöglicht dem `MessageHandler` das Zustellen einer Nachricht an die Senke eines anderen `MessageHandlers`. Sie ist wie folgt definiert:

```
public interface ISource
{
    void Connect(ISink Sink);
    void Disconnect(ISink Sink);
    void HandOver(AsynchronousMessage Message);
}
```



Die Methode `Connect` dient dazu, die Quelle mit einer Senke zu verbinden. Es kann sowohl eine Senke mit mehreren Quellen als auch eine Quelle mit mehreren Senken verbunden sein.

Im zweiten Fall muss sichergestellt sein, dass die `MessageFormats`-Eigenschaften aller Senken, die mit der gleichen Quelle verbunden sind, verschiedene Nachrichtenformate enthalten. Andernfalls kann ein deterministisches Verhalten nicht garantiert werden.

Mit der Methode `Disconnect` kann die angegebene Senke von der Quelle getrennt werden.

Die Methode `HandOver` übergibt die angegebene Nachricht an die mit der Quelle verbundene Senke; sind mehrere Senken mit ihr verbunden, wird die für das Format der Nachricht zuständige Senke ausgewählt.

Inside und Outside

Die Komponente `Inside` besitzt eine Methode `Connect`, durch die sie mit einer `Outside`-Komponente eines anderen `MessageHandlers` verbunden werden kann. Das bedeutet, dass die Quelle der `Inside`-Komponente an die Senke der `Outside`-Komponente gehängt wird und umgekehrt. Abbildung 6.7 illustriert dies.

Die `Outside`-Komponente besitzt eine entsprechende `Connect`-Methode, die sie

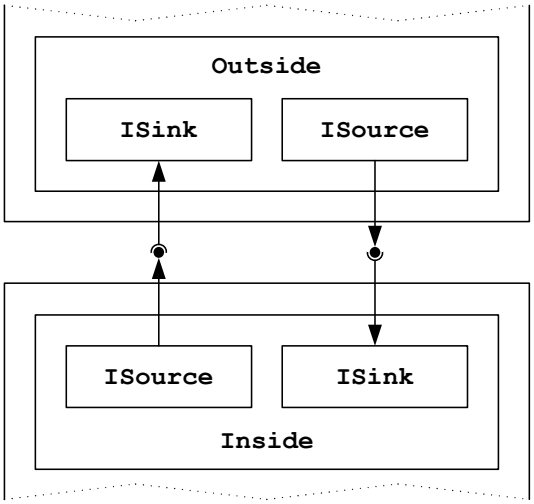


Abbildung 6.7: Verbindung der Komponenten `Inside` und `Outside` zweier `MessageHandler`

mit einer `Inside`-Komponente verbinden kann. Um zwei Komponenten zu verbinden, genügt es, die `Connect`-Methoden einer der beiden Komponenten aufzurufen.

6.3.4 `MessageHandler` der synchronen Kommunikation

Die `MessageHandler` der synchronen Kommunikation bestehen aus den Schnittstellen `IRequester` (*requester* engl. für: der Anfordernde) und `IResponder` (*responder* engl. für: der Antwortende).

`IResponder`

Die Schnittstelle `IResponder` bietet auf den ersten Blick die gleichen Methoden an wie die Schnittstelle `ISink`:

```
public interface IResponder
{
    SynchronousMessageBase Handle(
        SynchronousMessageBase Message);

    ICollection<Type> MessageFormats { get; }
}
```

Zu beachten ist, dass die Methode `Handle` in diesem Fall einen Rückgabewert hat: die Antwort-Nachricht.

Die Eigenschaft `MessageFormats` gibt wie die Schnittstelle `ISink` eine Liste der

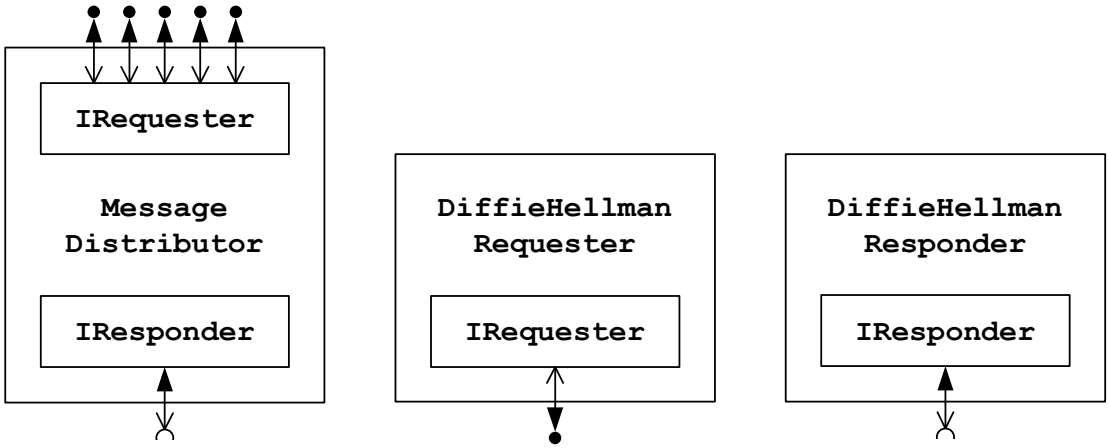


Abbildung 6.8: Synchrone Komponenten der Sicherheitsschicht im Detail

Nachrichtenformate zurück, für die der `MessageHandler` des `IResponder`s zuständig ist.

IRequester

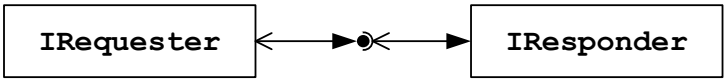
Die Schnittstelle `IRequester` ähnelt der asynchronen Schnittstelle `ISource`:

```
public interface Synchronous.IRequester
{
    void Connect(IResponder Responder);
    void Disconnect(Type MessageFormat);

    SynchronousMessageBase HandOver(
        SynchronousMessageBase Message);
}
```

Der Unterschied zur asynchronen Schnittstelle `ISource` ist auch hier, dass die `HandOver`-Methode einen Rückgabewert besitzt und so eine Antwort-Nachricht zurück liefert.

Die Methoden `Connect` und `Disconnect` verbinden einen `IResponder` mit einem `IRequester` bzw. trennen sie wieder.



Über `IRequester`-Schnittstellen werden Anfrage-Nachrichten an mit ihnen verbundene `IResponder`-Schnittstellen übergeben (geschlossener Pfeil). Von dort erhalten sie eine Antwort-Nachricht zurück (offener Pfeil).

Abbildung 6.8 zeigt die synchronen Komponenten der Sicherheitsschicht im Detail.

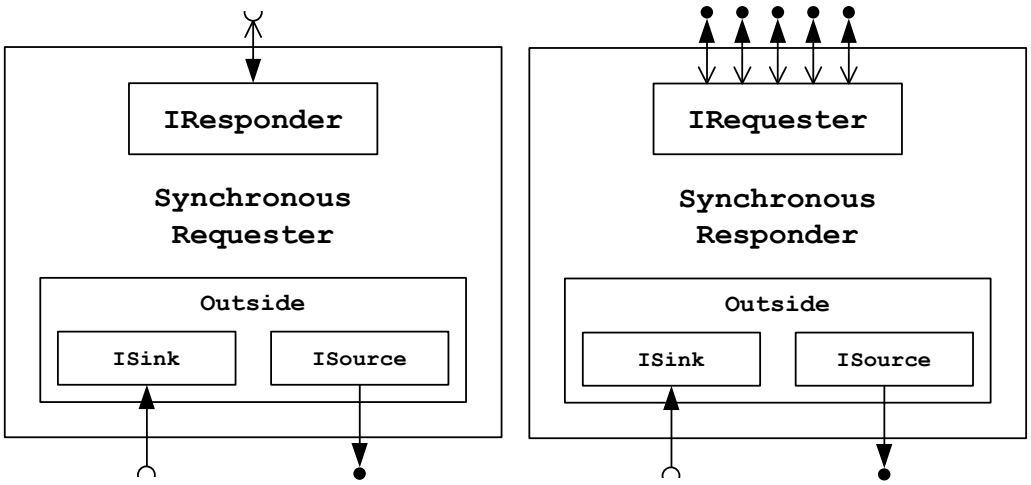


Abbildung 6.9: `SynchronousRequester` und `SynchronousResponder` im Detail

Der synchrone **`MessageDistributor`** des `MembershipService` implementiert sowohl eine Schnittstelle `IResponder` für eingehende Anfrage-Nachrichten als auch eine Schnittstelle `IRequester`, die die Anfragen an die zuständige Komponente der Anwendungsschicht weiter gibt.

Der **`DiffieHellmanRequester`** der Peers besitzt lediglich eine Schnittstelle `IRequester`, die die `DiffieHellman`-Nachricht an den `MembershipService` sendet und eine `DiffieHellman`-Nachricht als Antwort erhält.

Der **`DiffieHellmanResponder`** des `MembershipService` bietet die Schnittstelle `IResponder` an, die die Anfrage der Peers empfängt und eine Antwort zurück gibt.

6.3.5 Synchronisierer

Die `MessageHandler` zum Synchronisieren des asynchronen P2P-Overlays, `SynchronousRequester` und `SynchronousResponder`, bieten sowohl synchrone als auch asynchrone Schnittstellen an. Abbildung 6.9 stellt die Bestandteile von `SynchronousRequester` und `SynchronousResponder` dar.

Der `SynchronousRequester` nimmt synchrone Anfragen über seine Schnittstelle `IResponder` entgegen und versendet sie asynchron über eine `Outside`-Komponente. Danach blockiert er die Ausführung des Anfrage-Threads, bis ihn die Antwort über die `Outside`-Komponente erreicht. Nun wird der Anfrage-Thread fortgeführt und die Antwort über die `IResponder`-Schnittstelle als Rückgabewert zurückgeben.

Der `SynchronousResponder` ist symmetrisch dazu aufgebaut. Nachrichten erreichen ihn über eine `Outside`-Komponente. Die Anfrage wird über eine `IRequester`-

Schnittstelle an den verbundenen synchronen `MessageHandler` weiter gereicht. Die Antwort, die die `IRequester`-Schnittstelle als Rückgabewert von dort erhält, wird über die `Outside`-Komponente zurückgesendet.

6.3.6 Einbettung der Sicherheitsschicht

Die Sicherheitsschicht muss zwischen dem P2P-Overlay und der Anwendungsschicht eingebettet, das heißt mit diesen Schichten verbunden werden.

P2P-Overlay

Die Sicherheitsschicht wird mit dem P2P-Overlay über eine Abstraktion des P2P-Overlays verbunden. Diese Abstraktion muss die Schnittstellen `ISink` und `ISource` für die Gruppenkanäle jeder Gruppe und den Service-Kanal anbieten. Mit diesen Schnittstellen wird die äußere `MessageDistributor`-Komponente des `ServiceChannels` respektive der `SecureCommunicator` des `GroupChannels` verbunden.

Die Implementierung einer Abstraktion für das P2P-Overlay JXTA wird in Abschnitt 6.5 beschrieben.

Anwendungsschicht der Peers

Die Anwendungsschicht der Peers muss für den `GroupChannel` jeder Gruppe zum Empfangen von Gruppennachrichten eine `ISink`-Schnittstelle anbieten. Diese Schnittstelle wird mit der `ISource`-Schnittstelle des `SecureCommunicator` verbunden.

Zum Senden von Gruppennachrichten bietet die Sicherheitsschicht für jede Gruppe die Methode `Send` an, die die Nachrichten direkt in den `SecureCommunicator` des jeweiligen `ServiceChannel`s einspeist.

Auch für den `ServiceChannel` bietet die Sicherheitsschicht eine `Send`-Methode an, die Anfrage-Nachrichten der Gruppenoperationen entgegennimmt und die Antworten des `MembershipService` als Rückgabewert zurückgibt.

Anwendungsschicht des MembershipService

Der `MembershipService` implementiert für jedes Nachrichtenformat der Gruppenoperationen einen `MessageHandler`, der die Schnittstelle `IResponder` anbietet. Jeder dieser `MessageHandler` wird mit dem inneren `MessageDistributor` der Sicherheitsschicht verbunden. In Kapitel 6.6.2 werden die `MessageHandler` der Anwendungsschicht beschrieben.

6.4 Credential

Die verschiedenen Komponenten der Kommunikationskanäle benötigen Informationen über die Identität des Kommunikationspartners. Von eingehenden Nachrichten ist zunächst nur der nicht-authentifizierte Name des Absenders bekannt. Der Absender wird von den Komponenten `SecureCommunicator` bzw. `DiffieHellmanRequester` und `DiffieHellmanResponder` authentifiziert. An die Anwendungsschicht wird eine Nachricht erst übergeben, nachdem der Absender in der Sicherheitsschicht authentifiziert wurde.

Im Folgenden wird beschrieben, wie die Sicherheitsschicht ihren Komponenten und der Anwendungsschicht eine Möglichkeit anbietet, Informationen über die Identität des Kommunikationspartners zu beziehen.

Eine Struktur, die Informationen über einen Kommunikationspartner enthält, wird als **Credential** (engl. für: Berechtigungsnachweis) bezeichnet. Alle Credential-Implementierungen sind von der Schnittstelle **`ICredential`** abgeleitet.

Von jedem `MessageHandler` kann während der Bearbeitung der Nachricht das momentan gültige Credential über eine statische Eigenschaft der Klasse **`Credential`** abgefragt werden:

```
public abstract class Credential : ICredential
{
    public static ICredential ActiveCredential { get; }
}
```

Das Credential, welches über diese Eigenschaft bezogen werden kann, enthält alle aktuell über den Kommunikationspartner verfügbaren Informationen, wie den Namen, den öffentlichen Schlüssel oder den symmetrischen Kommunikationsschlüssel.

Das aktive Credential wird in der Sicherheitsschicht neu gesetzt, sobald detailliertere Informationen über den Kommunikationspartner bekannt werden.

Beim Eingang einer Nachricht erzeugt die P2P-Schnittstelle zunächst lediglich ein **`BasicCredential`**, welches ausschließlich Auskunft über den Namen des Kommunikationspartners gibt:

```
public class BasicCredential : Credential
{
    public string OpponentName { get; }
}
```

Solange der Kommunikationspartner nicht authentifiziert wurde, enthält `Credential.ActiveCredential` ein Credential dieses Typs.

Die `SecureCommunicator`-Komponente setzt nach erfolgreicher Entschlüsselung der Nachricht das aktive Credential auf ein Credential, das die **`ISecrecyCredential`**-Schnittstelle implementiert:

```
public interface ISecrecyCredential : ICredential
{
    SymmetricKey SecrecyKey { get; }
}
```

Ein `ISecrecyCredential` belegt, dass der Kommunikationspartner über den symmetrischen Kommunikationsschlüssel verfügt. Im Fall des `GroupChannel`s ist dadurch die Berechtigung des Absenders belegt, an der Gruppenkommunikation teilzunehmen, im Fall des `ServiceChannel`s die Identität des Kommunikationspartners. Der symmetrische Schlüssel kann über die Eigenschaft `SecrecyKey` bezogen werden.

`ICredentialFactory`

Der `SecureCommunicator` besitzt keine Informationen darüber, ob er eine Komponente des `GroupChannel`s oder des `ServiceChannel`s ist, und weiß auch nicht, ob er für einen Peer oder für den `ServiceChannel` arbeitet.

Daher kann er selbst nicht entscheiden, welche Implementierung der `ISecrecyCredential`-Schnittstelle er dem aktiven Credential zuweisen muss. Zu diesem Zweck benötigt er ein Objekt, welches die korrekte Implementierung für ihn erzeugt. Dieses Objekt muss die Schnittstelle **`ICredentialFactory`** anbieten:

```
public interface ICredentialFactory
{
    ISecrecyCredential CreateSecrecyCredential();
}
```

Beim Erzeugen des `SecureCommunicator`s wird ihm als Parameter die `ICredentialFactory` übergeben, die das gewünschte Credential erzeugt.

Die Schnittstelle `ICredentialFactory` wird von den Kommunikationskanälen implementiert.

Implementierung `ISecrecyCredential`-Schnittstelle

Die `ISecrecyCredential`-Schnittstelle wird von drei Klassen implementiert:

- **`GroupCredential`**: Dieses Credential wird vom `GroupChannel` der Peers verwendet und ist von der Klasse `BasicCredential` abgeleitet. Es enthält den symmetrischen Gruppenschlüssel des Kanals.
- **`ServiceCredential`**: Das `ServiceCredential` wird vom `ServiceChannel` der Peers verwendet. Es ist von der Klasse **`AuthenticationCredential`** abgeleitet und ermöglicht den Zugriff auf das eigene asymmetrische Schlüsselpaar des Peers und den öffentlichen Schlüssel des `MembershipService`.

- **`MemberCredential`**: Der `ServiceChannel` des `MembershipService` verwendet dieses Credential. Es ist ebenfalls von der Klasse `AuthenticationCredential` abgeleitet und ermöglicht den Zugriff auf das eigene asymmetrische Schlüsselpaar des `MembershipService` und den öffentlichen Schlüssel des Kommunikationspartners.

Abbildung 6.10 zeigt die Klassenhierarchie der Credential-Varianten.

`ActiveCredential`

Die Implementierung der Eigenschaft `ActiveCredential` ist nicht trivial, da das aktive Credential nicht an den Kommunikationskanal, sondern an den ausführenden Thread gebunden ist. Mehrere Threads können gleichzeitig mit verschiedenen aktiven Credentials Nachrichten im selben Kanal bearbeiten.

Daher wird zum Speichern des aktiven Credentials auf ein Objekt der Klasse `System.Threading.Thread.LocalDataStoreSlot` zurückgriffen. Diese Klasse ermöglicht das thread-abhängige Speichern von Daten. Beim Setzten des aktiven Credentials wird das Credential in den `LocalDataStoreSlot` geschrieben, beim Abfragen daraus gelesen.

6.5 JXTA Abstraktion

Im Rahmen dieser Diplomarbeit wurde die Klasse `JxtaOverlay` implementiert, um die Sicherheitsschicht mit einem JXTA-Overlay verbinden zu können. `JxtaOverlay` verwendet die JXTA-C-Implementierung[8] des JXTA-Protokolls. Diese Implementierung ist in der Programmiersprache C geschrieben und kann für diverse Plattformen compiliert werden.

Im Herbst 2005 wurde am LfBS eine Bibliothek geschrieben, die den Zugriff auf JXTA-C von der .NET-Laufzeitumgebung aus ermöglicht. Damit stehen die wichtigsten Funktionen der JXTA-C-Programmierschnittstelle für die C#-Programmiersprache zur Verfügung.

In JXTA werden Kommunikationskanäle durch Pipes abstrahiert[9]. Pipes können sowohl als Punk-zu-Punkt-Verbindung (Unicast-Pipes) als auch als Multicast-Verbindung (Propagate-Pipes) ausgelegt sein.

Unicast-Pipes eignen sich für die Nachrichtenübertragung zwischen den Peers und dem `MembershipService`, während Propagate-Pipes für die Gruppenkommunikation eingesetzt werden. Alle Mitglieder einer Gruppe lauschen an einer Propagate-Pipe und können die Gruppennachrichten empfangen.

[8] `jxta-c.jxta.org/`
[9] vgl. Kapitel 3.3.1: JXTA - It's all about protocols

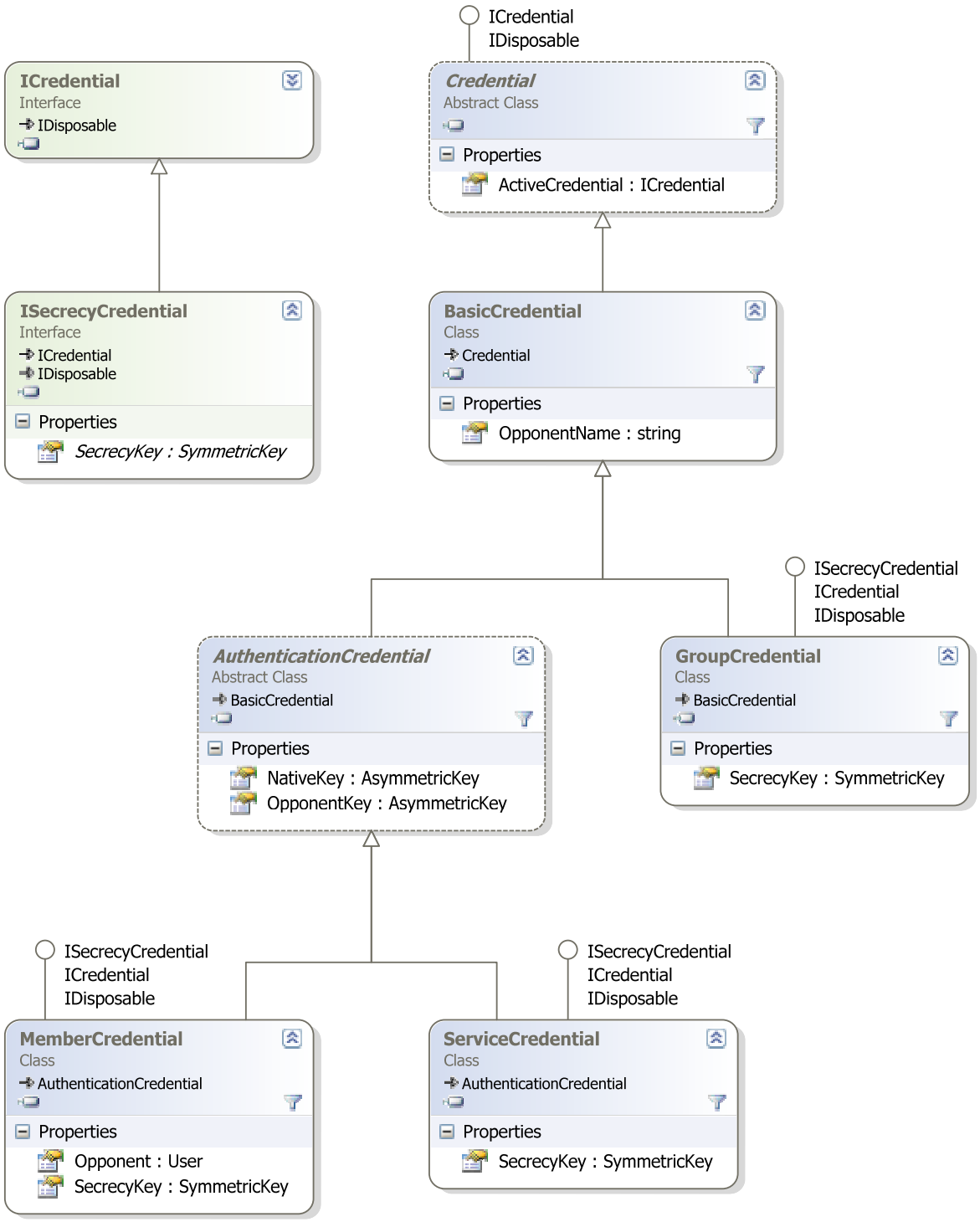


Abbildung 6.10: `Credential`-Klassenhierarchie

Zum Versand einer SCG-Nachricht muss diese in eine JXTA-Nachricht eingebettet werden. Zu diesem Zweck wird die Nachricht wie in Kapitel 6.2 beschrieben in ein XML-Dokument serialisiert.

Für die JXTA-Nachrichten werden folgende Elemente verwendet:

`Message:` Die SCG-Nachricht, die über das JXTA-Overlay versandt werden soll, als XML-Dokument.

`Type:` Der .NET-Datentyp der Nachricht. Der Datentyp wird benötigt, um auf Empfänger-Seite aus dem XML-Dokument wieder eine SCG-Nachricht deserialisieren zu können.

`Channel:` Der Kanal, in dem die Nachricht transportiert wird.

`ReplyName:` Der Name des Absenders. Antwort-Nachrichten werden an diesen Peer gesendet.

`JxtaOverlay` erzeugt für jeden Nachrichtenkanal ein Objekt der Klasse `JxtaSink`, die die Schnittstelle `ISink` implementiert. Diese Senken werden mit den Kanälen der Sicherheitsschicht verbunden. Von `JxtaSink` werden ausgehende SCG-Nachrichten serialisiert, in eine JXTA-Nachricht verpackt und über das JXTA-Overlay versendet.

Die vom Overlay empfangenen Nachrichten werden von `JxtaOverlay` deserialisiert und an den in der JXTA-Nachricht angegeben Kanal, genauer gesagt an den äußersten `MessageHandler` der Sicherheitsschicht, übergeben.

6.6 Ausführung der Gruppenoperationen

Der `MembershipService` speichert alle Informationen über Benutzer, Gruppen und Gruppenmitgliedschaften in einer SQL-Datenbank. Auf den Daten dieser Datenbank führt er die von den Peers angeforderten Gruppenoperationen durch.

6.6.1 Datenbankstruktur

Gruppenmitgliedschaften stellen eine *Many-to-Many* Beziehung zwischen Benutzern und Gruppen her: ein Benutzer kann in mehreren Gruppen Mitglied sein und eine Gruppe besitzt mehrere Mitglieder.

Aus diesem Grund besitzt die Datenbank je eine Tabelle für Benutzer (engl.: *users*) und für Gruppen (engl: *groups*), die über eine Tabelle für Mitgliedschaften (engl.: *memberships*) miteinander verbunden sind.

Die mit einer Mitgliedschaft assoziierten Rechte (engl: *rights*) werden in einer eigenen Tabelle abgelegt.

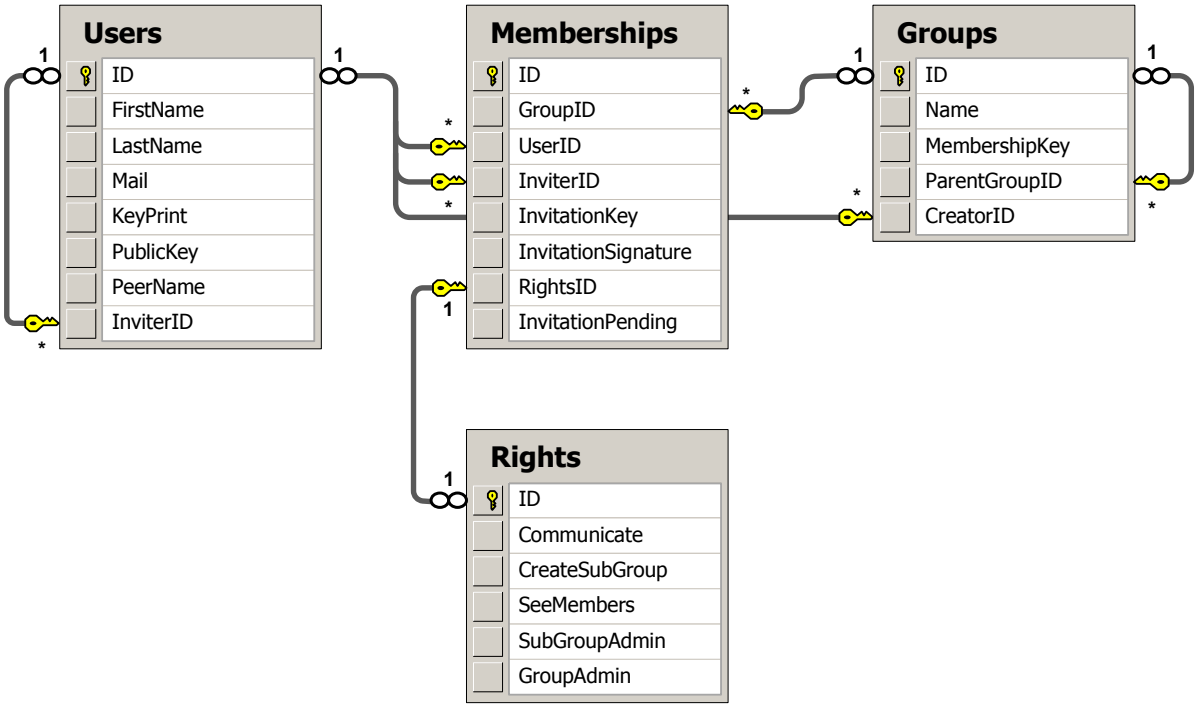


Abbildung 6.11: Datenbankstruktur

Abbildung 6.11 zeigt die Tabellenabhängigkeiten und die Spalten der Tabellen.

Die in der Tabelle genannten Benutzer-Datenfelder "FirstName", "LastName" und "Mail" sind exemplarisch; sie können durch Adressdaten oder sonstige Informationen erweitert bzw. ersetzt werden.

6.6.2 `MessageHandler` des `MembershipService`

Die `MessageHandler` des `MembershipService` führen die von den Peers angeforderten Gruppenoperationen aus.

Alle `MessageHandler` arbeiten nach demselben Schema:

Zunächst werden die Rechte, die der anfragende Peer in der betreffenden Gruppe hat, aus der Datenbank geladen. Dann wird anhand dieser Rechte überprüft, ob der Benutzer autorisiert ist, die angeforderte Operation durchzuführen. Wenn dies zutrifft, werden entsprechend der Operation Änderungen in der Datenbank vorgenommen. Zuletzt wird eine Antwort an den Peer gesendet.

Der `MembershipService` implementiert `MessageHandler` für alle neun Gruppenoperationen. Diese `MessageHandler` bieten die Schnittstelle `IResponder` an und sind mit dem inneren `MessageDistributor` der Sicherheitsschicht verbunden (siehe Abbildung 6.5 auf Seite 65).

Die SQL-Befehle für den Zugriff auf die Datenobjekte der Datenbank werden nicht direkt in den `MessageHandlern` erzeugt, da dies sehr fehleranfällig ist (s.u.) und häufig ein Sicherheitsrisiko darstellt. Statt dessen werden Wrapper-Klassen für den Datenbankzugriff verwendet. Das Projekt SOODA[10] stellt zu diesem Zweck eine sehr mächtige Werkzeugsammlung zur Verfügung. Der Umgang mit der Datenbank wird dadurch deutlich erleichtert, wie das folgende Beispiel zeigt:

Um mit einer SQL-Anfrage die Rechte des Benutzers *Ford* in einer Gruppe *UniverseTravelingGroup* aus der Datenbank zu laden, ist der folgende SQL-Befehl notwendig:

```
select * from dbo.Rights where ID =
(
  select RightsID from dbo.Memberships where UserID =
  (
    select ID from dbo.Users where PeerName = 'Ford'
  )
  and GroupID =
  (
    select ID from dbo.Groups where Name =
                              'UniverseTravelingGroup'
  )
)
```

Dieser Befehl muss als Zeichenkette erzeugt werden und über ein Command-Objekt an eine Datenbankverbindung übergeben werden. Das Ergebnis der Datenbank-Anfrage ist eine Liste der Rechte, auf die entweder sequentiell oder über den Spaltennamen zugegriffen werden kann. Ob der Benutzer das `InviteUser`-Recht besitzt, kann beispielsweise so überprüft werden:

```
if ((bool)RightsList["InviteUser"] == true)
   DoSomething();
```

Die Fehleranfälligkeit bei der direkten Verwendung von SQL-Befehlen rührt daher, dass sowohl beim Erzeugen der Anfrage als auch beim Auswerten des Ergebnisses Zeichenketten verwendet werden müssen. Wenn sich während der Programm-Entwicklung Bezeichnungen in der Datenbank (beispielsweise Spaltennamen) ändern, müssen im gesamten Programmcode die alten Bezeichnung durch die neuen ersetzt werden; werden einzelne Vorkommen übersehen, treten schwer aufspürbare Fehler auf.

Allerdings lässt sich gerade das schematische Erzeugen von SQL-Befehlen gut automatisieren. SOODA erzeugt aus einer Konfigurationsdatei Wrapper-Klassen, die als Datenbank-Stubs (*stub* engl. für: Stutzen) dienen. Von diesen Stubs werden automatisiert die notwendigen Zeichenketten für die SQL-Befehle generiert. Änderungen an der Datenbank müssen nur in der Konfigurationsdatei eingetragen werden.

[10] www.sooda.org

Die oben genannte SQL-Anfrage lässt sich mit Datenbank-Stubs wie folgt formulieren:

```
MembershipStub Membership =
  MembershipStub.LoadSingleObject
  (
    (MembershipField.User.PeerName == "Ford")
  &&
    (MembershipField.Group.Name == "UniverseTravelingGroup")
  );

if (Membership.Rights.InviteUser == true)
  DoSomething();
```

Durch die Datenbank-Stubs können die SQL-Anfragen typsicher formuliert werden. Das Ergebnis der Anfrage kann voll objekt-orientiert bearbeitet werden. Die einzigen Zeichenketten, die benötigt werden, sind die Namen des Peers und der Gruppe, die in der Nachricht der Gruppenoperation enthalten sind (*Ford* und *UniverseTravelingGroup* werden hier nur exemplarisch als konstante Werte verwendet).

6.7 Sicherheitshinweise

Beim Einsatz von SCG muss beachtet werden, dass die Sicherheitsarchitektur zwar eine sichere Gruppenkommunikation garantiert, dass aber dennoch Sicherheitsrisiken bestehen.

Die folgende Auflistung beinhaltet bekannte Sicherheitsrisiken; es wird kein Anspruch auf Vollständigkeit erhoben:

- Der Relay-Mechanismus von JXTA ist nicht nur autorisierten Gruppenmitgliedern, sondern allgemein offen zugänglich. Über den Relay-Mechanismus können Nachrichten über das JXTA-Overlay zugestellt werden, indem sie entlang einer Route von einzelnen Peers weitergereicht werden. Es besteht also die Risiko, dass das JXTA-Overlay von SCG für die Übertragung von Daten missbraucht wird. Da diese Daten im JXTA-Overlay von SCGweitergereicht werden, können zusätzliche Kosten für die Benutzer entstehen.
- Die JXTA-C-Implementierung kann nicht als gänzlich sicher eingestuft werden. Es ist anzunehmen, dass ein Angreifer Sicherheitslücken in JXTA-C ausnützen kann, um Zugriff auf die Systeme der Benutzer oder des `MembershipService` zu erlangen und private Daten auszuspähen oder Systemressourcen zu missbrauchen. Des Weiteren könnten sowohl die privaten Schlüssel der okkupierten Instanzen wie auch die aktuellen Gruppenschlüssel offengelegt werden.

Um von den Sicherheitseigenschaften, die SCG besitzt, in vollem Umfang profitie-

ren zu können, ist es daher notwendig, den Relay-Mechanismus in JXTA-C abzusichern und die JXTA-C-Implementierung eingehend auf Sicherheitslücken zu untersuchen. Alternativ könnte auf ein anderes P2P-Overlay zurückgegriffen werden, da SCG nicht auf die Verwendung mit JXTA-C eingeschränkt ist.

7 Zusammenfassung

Die Firma SYNCING.NET vertreibt eine Anwendung, die es Benutzern ermöglicht, Kalenderdaten und andere Informationen untereinander auszutauschen. Derzeit werden noch Fremd-Anwendungen für den Austausch der Daten verwendet. In der nächsten Version der Software soll die Kommunikation über ein integriertes P2P-Overlay abgewickelt werden.

Dafür wird ein System benötigt, in dem sich die Benutzer eigenverantwortlich in Gruppen organisieren können. Innerhalb dieser Gruppen soll es möglich sein, die Daten gleichberechtigt allen Gruppenmitgliedern zugänglich zu machen. Der Austausch der Daten soll ohne eine zentrale Instanz auf P2P-Basis möglich sein.

In dieser Diplomarbeit wurde die Programmbibliothek *Secure Communication Group* (SCG) entwickelt und implementiert, die ein solches System realisiert. Diese Bibliothek kann in einer .NET-Laufzeitumgebung ausgeführt werden und ist damit plattformunabhängig beispielsweise unter Windows, Linux oder MacOS einsetzbar.

SCG bietet folgende Funktionalitäten an:

- **Gruppenkommunikation:** SCG implementiert einen Mechanismus für eine Gruppenkommunikation. Benutzer können Mitglied in einer oder mehreren Gruppen sein. Innerhalb jeder ihrer Gruppen können sie Nachrichten an die übrigen Mitglieder der Gruppe senden.
- **Vertraulichkeit:** Zur Sicherung von sensiblen Daten wie beispielsweise Geschäftsgeheimnissen werden von SCG alle Nachrichten vor der Übertragung mit einem symmetrischen Schlüssel verschlüsselt und sind damit vor dem Zugriff Unberechtigter geschützt.
- **Authentifizierung:** Da die Gruppenkommunikation nur berechtigten Benutzer zugänglich sein darf, wurde eine zentrale Instanz, der `MembershipService`, implementiert, gegenüber dem sich die Benutzer authentifizieren müssen, um an der Gruppenkommunikation teilnehmen zu können. Für die Authentifizierung werden Signaturen und ein Public-Key-Kryptosystem verwendet.
- **Zugriffskontrolle:** Durch den `MembershipService` wird eine Zugriffskontrolle durchgeführt. Er stellt ausschließlich den Benutzern den symmetrischen Schlüssel einer Gruppenkommunikation zur Verfügung, die Mitglied dieser Gruppe sind.
- **Gruppenverwaltung:** Auch für die Verwaltung der Gruppen ist der `MembershipService` zuständig. Er nimmt von den Benutzern Aufträge zur Grup-

penverwaltung entgegen. Wenn ein Benutzer über entsprechende Rechte verfügt, kann er den `MembershipService` dazu veranlassen, Benutzer in Gruppen einzuladen oder Mitglieder einer Gruppe aus dieser auszuschließen. Des Weiteren können Gruppen erzeugt und aufgelöst sowie die Berechtigungen von Benutzern verändert werden.

- **Rechteverwaltung:** Der `MembershipService` verfügt über eine Datenbank, in der die Rechte der Benutzer verzeichnet sind. Wenn der `MembershipService` einen Auftrag zur Gruppenverwaltung erhält, überprüft er die Berechtigung des Benutzers anhand dieser Datenbank.

Für SCG wurde kein neues P2P-System entwickelt; SCG kann auf beliebige P2P-Systeme aufgesetzt werden. Im Rahmen dieser Arbeit wurde eine Schnittstelle realisiert, über die auf eine Implementierung der JXTA-P2P-Protokollspezifikation zugegriffen werden kann. Wegen der gewünschten Interoperabilität mit .NET und wegen ihrer Plattformunabhängigkeit wurde die JXTA-C-Implementierung ausgewählt, die über einen C# -Wrapper verfügt.

Die Schnittstelle kann mit geringem Aufwand für andere P2P-Systeme angepasst werden.

Zwei Mechanismen der Protokolldefinition JXTA konnten ihm Rahmen dieser Diplomarbeit nicht in das System eingebracht werden:

- **JXTA-Dienste:** Die Zugriffskontrolle der sicheren Gruppenkommunikation wird von einer zentralen Komponente zur Verfügung gestellt. Wenn diese Komponente ausfällt oder von einem Angreifer (beispielsweise durch eine Denial-Of-Service-Attacke) unerreichbar gemacht wird, fällt das ganze System aus. Daher sollte diese Komponente redundant ausgelegt sein.

  JXTA bietet mit Gruppendiensten eine Lösung für diese Anforderung an. Mehrere JXTA-Peers können den gleichen Gruppendienst anbieten, auf den die Mitglieder einer Gruppe transparent zugreifen. Die Anfrage eines Peers wird automatisch einer Instanz dieses Gruppendienstes zugestellt. Ist einer der Peers, die diesen Dienst anbieten, nicht erreichbar, wird die Anfrage zu einer der übrigen Instanzen geleitet. Dadurch wird eine Lastverteilung unter den Anbietern der Gruppendienste erreicht und ein Angriff auf die Funktionalität des Systems erschwert.

  Leider werden die Service-Funktionalitäten der JXTA-C-Implementierung derzeit noch nicht über den C# -Wrapper für .NET zugänglich gemacht, daher musste auf eine Implementierung des `MembershipService` als JXTA-Service verzichtet werden.

- **JXTA-Gruppen:** Die Protokollspezifikation von JXTA sieht das Zusammenfassen von mehreren Peers, die die gleichen Interessen vertreten, zu eine Gruppe vor.

  Peer-Gruppen sind auch in JXTA-C implementiert; allerdings konnte in Tests

nicht immer von jedem Peer eine Verbindung zu seinen Gruppen hergestellt werden. Aus diesem Grund wurde in der Implementierung von SCG auf eine Verwendung von JXTA-Gruppen verzichtet.

Wenn von JXTA-C die notwendigen Bedingungen erfüllt werden, ist die Einbindung von JXTA-Diensten und -Gruppen eine sinnvolle Erweiterung für die sichere Gruppenkommunikation von SCG. Diese Erweiterung wäre ein mögliches Thema beispielsweise für eine Studienarbeit.

Die Implementierung eines C# -Wrappers für die Service-Funktionalitäten der JXTA-C-Implementierung ist dagegen mit größerem Aufwand verbunden und könnte im Rahmen einer weiteren Diplomarbeit realisiert werden.

Mit zunehmender Verbreitung von P2P-Anwendungen im privaten, akademischen und gewerblichen Bereich steigt auch der Bedarf an sicheren P2P-Systemen, die Sicherheitsaspekte wie Vertraulichkeit, Integrität, Authentizität und Verbindlichkeit berücksichtigen. Diese Sicherheitsaspekte können in Form einer sicheren Gruppenkommunikation realisiert werden. Eine wichtige Voraussetzung für eine sichere Gruppenkommunikation ist eine verlässliche Zugriffskontrolle.

Die Forschung hat sich bereits mit Zugriffskontrollmechanismen für P2P-Systeme auseinander gesetzt [8]. Dennoch bleiben einige Fragen bezüglich einer gänzlich dezentralen Zugriffskontrolle für P2P-Systeme unbeantwortet.

Des Weiteren gibt es keine Forschungsarbeit, die Sicherheitsanforderungen von P2P-Systemen umfassend untersucht. Über die sichere Gruppenkommunikation hinaus müssen auch Aspekte wie Verlässlichkeit und Privatsphäre in eine gesamtheitliche Betrachtung der Sicherheit von P2P-Systemen einbezogen werden. Damit besteht Bedarf für weitere Forschung im Bereich „Sicherheit von P2P-Systemen“.

Abkürzungsverzeichnis

ACL	Access Control Liste; *access control* engl. für Zugriffskontrolle
AES	Advanced Encryption Standard; symmetrisches Kryptosystem, nach seinen Entwicklern Joan Daemen und Vincent Rijmen gelegentlich auch Rijndael-Algorithmus genannt
ARPANET	Advanced Research Projects Agency Network; Vorläufer des heutigen Internets
DHS	Diffie-Hellman-Schlüsseltausch; von Whitfield Diffie und Martin Hellman entwickelter Algorithmus zum Austausch eines geheimen Schlüssels über einen unsicheren Kommunikationskanal
DHT	Distributed Hash Table; verteilte Hashtabelle
GUI	Graphical User Interface; grafische Benutzerschnittstelle
HTTP	Hypertext Transfer Protocol; Datenübertragungsprotokoll
ISO	International Standardization Organization; Internationale Organisation für Normung
JXTA	*to juxtapose* engl. für nebeneinander stellen; von SUN entwickelte P2P-Protokolldefinition.
LfBS	Lehrstuhl für Betriebssysteme
NAT	Network Address Translation; Adressumsetzung in Rechnernetzen
OSI	Open Systems Interconnection; Referenzmodell zum Austausch von Daten über ein Rechnernetz
P2P	Peer-to-Peer; *peer* engl. für der Gleichgestellte; in P2P-Netzen sind alle Instanzen gleichwertig oder gleichberechtigt

RSA asymmetrisches Kryptosystem, benannt nach seinen Entwicklern Ronald L. Rivest, Adi Shamir und Leonard Adleman

SCG Secure Communication Group; im Rahmen dieser Diplomarbeit entwickelten Programmbibliothek für eine sichere Gruppenkommunikation

SHA Secure Hash Algorithm; Gruppe standardisierter kryptographischer Hash-Funktionen, bsp.: SHA-1, SHA-256, SHA-512

TAN Transaktionsnummer

TCP Transmission Control Protocol; Protokoll für die Kommunikation in einem Rechnernetz

Telnet Telecommunication Network; mittlerweile veraltetes Netzkommunikationsprotokoll

TTL Time To Live; Lebenszeit einer Nachricht

WWW World Wide Web; über das Internet abrufbares Hypertext-System

XML Extensible Markup Language; Standard zur Modellierung von strukturierten Daten

Index

A

Abhören 6
Absender 5
Abstraktionsschichten 5
`AcceptInvitations` 39
`AcceptInvitations`-Nachricht . . 39
`Accountability`-Nachricht 53
`AccountabilityInsurer` 64–66
`AckResponse`-Nachricht 39
Advertisement 28
asynchron 6
`AuthenticationCredential` 74
Authentifizierung 6, 12, 29
Authentisierung 6
Authentizität 7, 10, 11
Autorisierung 6

B

`BasicCredential` 73
Benutzer 6

C

`Communicate`-Recht 37
`CreateSubGroupResponse`-Nachricht 41
`CreateSubGroup` 41
`CreateSubGroup`-Recht 37
`CreateSubGroupRequest`-Nachricht 41
Credential 73
  `AuthenticationCredential` 74
  `BasicCredential` 73
  `GroupCredential` 74
  `ICredential` 73
  `ICredentialFactory` 74
  `ISecrecyCredential` 73
  `MemberCredential` 75
  `ServiceCredential` 74

D

Denial-Of-Service-Attacken 7
Dienst 28
Diffie-Hellman-Geheimnis 16
Diffie-Hellman-Schlüsseltausch 15
`DiffieHellman`-Nachricht 52
DiffieHellmanKey 52
`DiffieHellmanKey` 51
`DiffieHellmanRequester` 63, 64, 71
`DiffieHellmanResponder` . . . 66, 71

E

Empfänger 5
Endpunkt 27

F

Fingerprint 11

G

geheimer Schlüssel 9
`GetGroupKeys` 40
`GetMembers` 40
`GetMemberships` 38
`GroupAdmin`-Recht 37
`GroupChannel` 35
`GroupCredential` 74
GroupKey 52
`GroupKey` 35, 47, 49
`GroupKeysRequest`-Nachricht 40
`GroupKeysResponse`-Nachricht . . 40
Gruppe 6
Gruppenoperationen 36
  `AcceptInvitations` 39
  `CreateSubGroup` 41

GetGroupKeys 40
GetMembers 40
GetMemberships 38
InviteUser 41
ModifyMembership 42
RemoveGroups 42
RemoveMembers 42
Gruppenrechte 36
Gruppenschlüssel-Management 29

H

Hash-Wert 10
Hashtabelle 24

I

ICredential 73
ICredentialFactory 74
Identität 6
Instanzen 5
Integrität 7, 10, 11, 29
InvitationKey 35, 48
InviteUser 41
InviteUser-Nachricht 41
ISecrecyCredential 73

K

Kandidat 37
Kollision 10
Kollisionsresistenz 10
Kommunikation 5
Verbindung 5
Kryptosystem
asymmetrisch 9
symmetrisch 9

M

Man-in-the-Middle-Attacke 6
Maskerade 7
Mehrpunktverbindung 6
MemberCredential 75
MembershipKey 35, 48
MembershipsRequest-Nachricht . 38
MembershipsResponse-Nachricht 38
MembersRequest-Nachricht 40
MembersResponse-Nachricht 40
MessageDistributor 64, 66, 71
MessageHandler
AccountabilityInsurer . 64–66
DiffieHellmanRequester .. 63, 64, 71
DiffieHellmanResponder .. 66, 71
MessageDistributor . 64, 66, 71
SecureCommunicator . 62, 64–66
SynchronousRequester .. 63–65
SynchronousResponder 66
Mitglied 29
ModifyMembership 42
ModifyMembership-Nachricht 42
Multicast-Verbindung 6
multiplikative Gruppe 15

N

Nachricht 5, 28, 36
Nachrichtenelement 36
Nachrichtenformat 36
Nachrichtenformate
AcceptInvitations 39
Accountability 53
AckResponse 39
CreateSubGroupResponse ... 41
CreateSubGroupRequest 41
DiffieHellman 52
GroupKeysRequest 40
GroupKeysResponse 40
InviteUser 41
MembershipsRequest 38
MembershipsResponse 38
MembersRequest 40
MembersResponse 40
ModifyMembership 42
RemoveGroups 42
RemoveMembers 42
Secrecy 53
Synchronization 64
Nachrichtenkanal 5

O

Obergruppe 36

öffentlicher Schlüssel 9
Overlay-Netz 19

P

P2P-System
  strukturiert 22
    DHT-basiert 23
    zentralisiert 23
  unstrukturiert 20
    hybrid 22
    rein (*pure*) 20
Peer-to-Peer System 19
Peergruppe 28
Peers 27
Perfect Forward Secrecy 15
Pipe 28
Primitivwurzel 15
privater Schlüssel 9
Protokoll 5
Public-Key-Kryptosystem 9
Punkt-zu-Punkt-Verbindung 5

R

Rechte
  `Communicate` 37
  `CreateSubGroup` 37
  `GroupAdmin` 37
  `SeeMembers` 37
  `SubGroupAdmin` 37
`RemoveGroups` 42
`RemoveGroups`-Nachricht 42
`RemoveMembers` 42
`RemoveMembers`-Nachricht 42
Replay-Attacke 7

S

Salz 14
Schlüssel
  geheim 9
  öffentlich 9
  privat 9
`Secrecy`-Nachricht 53
`SecureCommunicator` 62, 64–66
`SeeMembers`-Recht 37
`ServiceChannel` 35
`ServiceCredential` 74
Sichere Gruppenkommunikation . . . 29
Sicherheitsschicht 12
Signatur 11
Social Engineering 7
Station-To-Station-Protokoll 17
`SubGroupAdmin`-Recht 37
Suchschlüssel 22
Superpeer 22
synchron 6
`Synchronization`-Nachricht 64
`SynchronousRequester` 63–65
`SynchronousResponder` 66

U

Unicast-Verbindung 5
  bidirektional 6
  unidirektional 6
Untergruppe 36
Untergruppenverwalter 37
Urbildresistenz 10

V

Verbindlichkeit 7, 10, 11
Verbindung 5
Verschlüsselung 12
Vertraulichkeit 7, 12, 29

W

Wiedereinspielung 49
Wurzelgruppe 36

Z

Zertifikat 14
Zugriffskontrolle 29

Literaturverzeichnis

[1] *JXTA v2.0 Protocols Specification*. `http://spec.jxta.org/v1.0/docbook/JXTAProtocols.pdf`. 2007

[2] AMIR, Y. ; NITA-ROTARU, C. ; STANTON, J. ; TSUDIK, G. : Scaling Secure Group Communication Systems: Beyond Peer-to-Peer. In: *DARPA Information Survivability Conference and Exposition* Bd. 1, 2003, S. 226–237

[3] ATENIESE, G. ; STEINER, M. ; TSUDIK, G. : Authenticated group key agreement and friends. In: *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*. New York, NY, USA : ACM Press, 1998. – ISBN 1–58113–007–4, S. 17–26

[4] BLESS, R. ; MINK, S. ; BLASS, E.-O. ; CONRAD, M. ; HOF, H.-J. ; KUTZNER, K. ; SCHÖLLER, M. : *Sichere Netzwerkkommunikation*. Heidelberg : Springer, Juni 2005 (X.systems.press). – ISBN 3–540–21845–9

[5] CASTRO, M. ; DRUSCHEL, P. ; KERMARREC, A. ; ROWSTRON, A. : SCRIBE: A large-scale and decentralized application-level multicast infrastructure. In: *IEEE Journal on Selected Areas in communications (JSAC)* 20 (2002), Nr. 8, S. 1489–1499

[6] DIFFIE, W. ; HELLMAN, M. E.: New Directions in Cryptography. In: *IEEE Transactions on Information Theory* IT-22 (1976), Nr. 6, S. 644–654

[7] DIFFIE, W. ; OORSCHOT, P. C. V. ; WIENER, M. J.: Authentication and authenticated key exchanges. In: *Des. Codes Cryptography* 2 (1992), Nr. 2, S. 107–125. – ISSN 0925–1022

[8] KIM, Y. ; MAZZOCCHI, D. ; TSUDIK, G. : Admission Control in Peer Groups. In: *NCA '03: Proceedings of the Second IEEE International Symposium on Network Computing and Applications*. Washington, DC, USA : IEEE Computer Society, 2003. – ISBN 0–7695–1938–5, S. 131

[9] MAYMOUNKOV, P. ; MAZIERES, D. . *Kademlia: A peer-to-peer information system based on the XOR metric*. 2002

[10] MENEZES, A. J. ; VAN OORSCHOT, P. C. ; VANSTONE, S. A.: *Handbook of Applied Cryptography*. CRC Press, 1996

[11] NARASIMHA, M. ; TSUDIK, G. ; YI, J. H.: On the Utility of Distributed Cryptography in P2P and MANETs: The Case of Membership Control. In: *ICNP '03: Proceedings of the 11th IEEE International Conference on Network Protocols*. Washington, DC, USA : IEEE Computer Society, 2003. – ISBN 0–7695–2024–3, S. 336

[12] ORAM, A. : *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly, March 2001. – ISBN 059600110X

[13] RATNASAMY, S. ; FRANCIS, P. ; HANDLEY, M. ; KARP, R. ; SHENKER, S. : A Scalable Content Addressable Network. Berkeley, CA, 2000 (TR-00-010). – Forschungsbericht

[14] STEINER, M. ; TSUDIK, G. ; WAIDNER, M. : Diffie-Hellman key distribution extended to group communication. In: *CCS '96: Proceedings of the 3rd ACM conference on Computer and communications security*. New York, NY, USA : ACM Press, 1996. – ISBN 0–89791–829–0, S. 31–37

[15] STEINER, M. ; WAIDNER, M. ; TSUDIK, G. : CLIQUES: A New Approach to Group Key Agreement. In: *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*. Washington, DC, USA : IEEE Computer Society, 1998. – ISBN 0–8186–8292–2, S. 380

[16] STEINMETZ, R. ; WEHRLE, K. : Peer-to-Peer-Networking & -Computing - Aktuelles Schlagwort. In: *Informatik Spektrum* 27 (2004), Nr. 1, S. 51–54

[17] STEINMETZ, R. (Hrsg.) ; WEHRLE, K. (Hrsg.): *Peer-to-Peer Systems and Applications*. Bd. *3485*. Springer, 2005 (Lecture Notes in Computer Science). – ISBN 3–540–29192–X

[18] STOICA, I. ; ADKINS, D. ; ZHUANG, S. ; SHENKER, S. ; SURANA, S. : Internet indirection infrastructure. 2002. – Forschungsbericht

[19] TRAVERSAT, B. ; ARORA, A. ; ABDELAZIZ, M. ; DUIGOU, M. ; HAYWOOD, C. ; HUGLY, J.-C. ; POUYOUL, E. ; YEAGER, B. . *Project JXTA 2.0 Super-Peer Virtual Network*. 2003

[20] ZHANG, Y. ; LI, X. ; HUAI, J. ; LIU, Y. : Access control in peer-to-peer collaborative systems. In: *ICDCSW'05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops*, IEEE Computer Society, 2005. – ISBN 0–7695–2328–5, S. 835–840