UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Absicherung und Verifikation einer Instant Messaging Kommunikation über DH-Key Agreement

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science
im Studiengang Informationsmanagement

vorgelegt von

Florian Dietz

Erstgutachter:	Prof. Dr. Rüdiger Grimm Institut für Wirtschafts- und Verwaltungsinformatik
Zweitgutachter:	Dipl.-Inform. Anastasia Meletiadou Institut für Wirtschafts- und Verwaltungsinformatik

Koblenz, im April 2009

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☒	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☒

.....

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Zusammenfassung

Im Rahmen dieser Bachelorarbeit soll die Absicherung und Verifikation einer Instant Messaging-Kommunikation beschrieben und in Form eines Plugins für ein bestehendes Instant Messaging-System realisiert werden. Dazu werden zunächst die Anforderungen an das System erhoben und mit Vorhandenen Konzepten und Systemen am Markt verglichen. Darauf aufbauend wird ein eigenes Konzept entwickelt und die Umsetzung in der Programmiersprache Java beschrieben.
Der Fokus dieser Arbeit liegt dabei auf der Absicherung des Chat-Kanals einer Instant Messaging-Kommunikation, aber es werden geeignete Schnittstellen beschrieben und implementiert, die eine Erweiterung der angewendeten Verfahren für beliebige Datensätze ermöglicht.

1.2 Motivation

Einer Schätzung aus dem Jahre 2005 zur Folge werden täglich über 12 Billionen Nachrichten über IM-Systeme versendet, wobei über 90% der Nachrichten privaten Zwecken dienen [Ana05]. Alleine die bloße Anzahl an versendeten Nachrichten lässt darauf schließen, dass Instant Messaging im privaten Umfeld kein temporäres Randphänomen, sondern ein alltägliches und beliebtes Werkzeug der Kommunikationsunterstützung geworden ist. Es existieren eine Vielzahl unterschiedlicher IM-Systeme, deren Funktionalität nicht mehr auf den reinen Aus-

tausch von Textnachrichten beschränkt ist, sondern Vielmehr zu einer einheitlichen Plattform für unterschiedlichste Anwendungen geworden ist.
Auch im geschäftlichen Umfeld werden zunehmend IM-Systeme eingesetzt, aber erfordern aufgrund höherer Anforderungen, insbesondere in Bezug auf Sicherheit und Zuverlässigkeit der eingesetzten Systeme eine erweiterte Betrachtungsweise. Soll IM auch in sicherheitskritischen Bereichen eingesetzt werden, erfordert dies eine genaue Analyse möglicher Gefahren und die Auswahl geeigneter Systeme und Konzepte um ein ausreichendes Sicherheitsniveau garantieren zu können. Im Vergleich zu anderen Kommunikationsmitteln wie beispielsweise E-Mail, fehlen im Bezug auf IM jedoch einheitliche Standards, die eine Auswahl geeigneter Systeme erleichtern könnten [Joe08]. Ziel der Arbeit soll es daher auch sein, eine Beispielanwendung zu liefern, die trotz potentiell unsicherer Leitungen ein ausreichendes Sicherheitsniveau für den Unternehmenseinsatz bietet.

1.3 Notation

Bei der Beschreibung von Systemen und der Darstellung von eigenen Konzepten wird auf eine Reihe von Notationen zurückgegriffen, die im folgenden eingeführt werden.

- **Modellierung:** Modelle zur Beschreibung von Systemen und Konzepten werden in der *Unified Modelling Language* (UML) in Version 2.0 dargestellt. Unabhängig von einem konkreten Modell wird dabei von der Möglichkeit Gebrauch gemacht, nur den jeweils betrachteten Zusammenhang darzustellen und nicht betrachtete Zusammenhänge von der Modellierung auszuschließen.
- **Protokollbeschreibungen:** Protokolle werden in vereinfachten UML- Sequenzdiagrammen dargestellt.
- **Namensgebung:** Werden Namen zur Darstellung von Protokollen oder Ereignissen verwendet, so werden die in der Kryptographie üblichen Namen verwendet [Sch05]:
  - Alice und Bob: Reguläre Teilnehmer einer Sitzung

- Eve: Ein Angreifer auf eine Sitzung
- Carol und Charlie: Weitere reguläre Teilnehmer

Sollte an einer Stelle von der genannten Notation abgewichen werden, wird im Text gesondert darauf hingewiesen.

Kapitel 2

Grundlagen des Instant Messaging

Bereits seit den Anfangszeiten des Internets wird das Internet (auch) dazu genutzt, Nachrichten zwischen Personen auszutauschen. Anwendungen wie *E-Mail*, *IRC* oder UNIX *talk* boten schon früh viele Funktionen, die heutige Instant Messaging-Systeme kennzeichnen.
Gegenstand dieses Kapitels ist daher die grundlegende Einführung und Definition der Begrifflichkeiten und Konzepte, die im weiteren Verlauf der Arbeit verwendet werden. Darauf aufbauend werden die wichtigsten Vertreter der jeweiligen Systeme und Protokolle am Markt charakterisiert und klassifiziert.

2.1 Instant Messaging

Ein Instant Messaging (IM)-System stellt Präsenz- und Verfügbarkeitsinformationen über die Nutzer des Systems zur Verfügung, um den spontanen Austausch von Nachrichten zwischen den Nutzern zu ermöglichen [TG07]. Damit stellt ein IM-System ein soziotechnisches System dar, das von Personengruppen zur Kommunikationsunterstützung verwendet werden kann. Die Kommunikationsunterstützung ist daher der Kern eines IM-Systems, der jedoch durch weitere Funktionen angereichert wird, um die Kommunikation möglichst spontan und komfortabel zu gestalten. Der Nutzen eines IM-Systems gegenüber Vorgängern wie Unix talk ergibt sich erst aus der Kombination verschiedener Anwendungen und Kon-

zepte in einer „Managed Messaging Platform“ [Shi02]. Typischerweise besteht ein solches IM-System aus einem IM-Client, der bei jedem Nutzer installiert sein muss, einem IM-Server, der eine Vielzahl von Clients verwaltet und einem Protokoll, über das Server und Clients miteinander kommunizieren. Auch wenn die Funktionen, die ein solches System realisiert von System zu System unterschiedlich gestaltet sind, lassen sich dennoch einige Kernfunktionalitäten feststellen, die nahezu jedes heutige IM-System realisiert:

- **Registrierung:** Nutzer eines Systems müssen sich bei einem Server registrieren, um ein Nutzerkonto (*Account*) zu erstellen. Die Registrierung erfolgt üblicherweise einmalig bei der ersten Nutzung des Systems und beinhaltet mindestens die Wahl von Nutzernamen und Passwort des Nutzerkontos sowie der Zuteilung einer eindeutigen IM-Adresse.

- **Anmelden und Abmelden:** Um von anderen Nutzern gesehen und die Funktionen des Systems nutzen zu können, müssen Nutzer sich mit ihrem Client beim Server unter Angabe von Nutzernamen und Passwort anmelden.

- **Setzen und Anfragen von Verfügbarkeitsstati:** Mittels Präsenz- und Verfügbarkeitsstati kann ein Nutzer angeben, in welchem Zustand er sich derzeit befindet und welche Bereitschaft zur Kommunikation seinerseits besteht. Das zugrundeliegende Konzept der „Awareness“ wird näher im Abschnitt „Awareness-Unterstützung“ dieses Kapitels behandelt.

- **Kontaktlisten:** Ein Nutzer kann sich eine persönliche Kontaktliste (*Contact List*) anlegen und darin andere Nutzer (*Contacts*) des Systems anzeigen lassen. Üblicherweise ist die Kontaktliste zentrales Element der Benutzeroberfläche eines Clients und bietet neben dem Anzeigen der Contacts über Kontextmenüs die Nutzung weiterer Funktionen an.

- **Gruppenbildung:** Die Contact List eines Clients kann in unterschiedliche Gruppen unterteilt werden, in die der Nutzer die Contacts sortieren und gruppenspezifische Funktionen ausführen kann.

- **Versenden von Textnachrichten:** Der Austausch von Textnachrichten zwischen einzelnen Nutzern

- **Starten von Echtzeitanwendungen:** Nutzer können Anwendungen wie Internettelefonie oder Dateiübertragung über den IM-Client miteinander starten

Neben den Kernfunktionalitäten bieten heutige IM-Systeme noch eine Vielzahl von Erweiterungen, die jedoch vorerst außer Acht gelassen werden. Die Bandbreite solcher Erweiterungen erstreckt sich jedoch von der Integration in andere Groupware-Systeme bis hin zu Online-Spielen.

2.2 Voice over IP (VoIP)

Heutige IM-Systeme beschränken sich größtenteils nicht auf den Austausch von Textnachrichten, sondern bieten auch eine Unterstützung von Sprach- und Videoübertragung. Unter dem Begriff „Voice over IP (VoIP)“ versteht man verschiedene Techniken zur Echtzeitübertragung von Sprache über IP-Netze. Das Einsatzfeld von VoIP beschränkt sich jedoch nicht auf den Einsatz innerhalb von IM-Systemen. Die zugrundeliegende Technik stellt vielmehr eine Alternative zur traditionellen, auf Leitungsvermittlung basierten Sprachtelefonie dar, die von Endgeräten bis hin zu IM-Systemen genutzt werden kann [Bun09]. Zu Beginn der Entwicklung von VoIP entstanden eine Vielzahl von Anwendungen, die die VoIP-Funktion in einzelne Anwendungen kapselten und zumeist als „VoIP-Clients“ bezeichnet wurden. Im Laufe der Zeit wurden diese VoIP-Clients jedoch um typische Funktionalitäten eines IM-Systems erweitert und gleichzeitig implementierten viele IM-Systeme die VoIP-Funktionalität in bestehende IM-Systeme. Daher findet sich in älteren Ansätzen oftmals eine Differenzierung von IM-Systemen, als Systeme zum Austausch von Textnachrichten und VoIP-Systemen zur Sprachübertragung. Im Rahmen dieser Arbeit wird IM im erweiterten Sinne betrachtet, d.h. Funktionen wie VoIP oder Dateiübertragung werden als Teil des IM-Systems gesehen (vgl. [Mel09]).

2.3 Awareness-Unterstützung

Unter dem Begriff „Awareness“ wird im Forschungsgebiet *Computer Supported Cooperative Work* (CSCW) die gegenseitige Information von Nutzern übereinander verstanden, die der Reduktion von Unsicherheiten und der spontanen Koor-

dination dient [TG07]. In IM-Systemen findet sich dieser Aspekt in dem Bereitstellen von Präsenz- und Verfügbarkeitsinformationen wieder. Ein weit verbreitetes Konzept zur Bereitstellung von Awareness-Informationen stellt das Setzen von Stati dar. Dabei kann ein Nutzer einen Status aus einer Reihe vordefinierter Stati auswählen und wird allen anderen Nutzern fortan in diesem Status dargestellt. Typische Stati, die von einer Reihe unterschiedlicher IM-Systeme verwendet werden sind beispielsweise

- **Online:** Der Nutzer ist mit dem Server verbunden und ansprechbar
- **Away:** Der Nutzer ist kurzzeitig nicht verfügbar
- **Not Available:** Der Nutzer ist längerfristig nicht verfügbar
- **Do not Disturb:** Der Nutzer möchte nicht gestört werden
- **Offline:** Der Nutzer ist nicht verfügbar

Neben einer Reihe vordefinierter Stati ermöglichen es viele IM-Systeme auch eigene Stati zu definieren und einem Status mit einer individuellen Nachricht zu ergänzen. Dadurch sind Nutzer in der Lage nicht nur den eigenen Status, sondern zusätzliche Hintergrundinformationen bereitzustellen.

2.4 Kommunikationsformen

Grundsätzlich ist eine Instant Messaging-Kommunikation darauf ausgerichtet, möglichst zeitnah Nachrichten zwischen Nutzern an unterschiedlichen Standorten auszutauschen. Ordnet man Instant Messaging beispielsweise in die zweidimensionale Raum-Zeit-Taxonomie von Johansen ein (siehe Abbildung 2.1), wäre es im Feld „gleiche Zeit“ und „verschiedener Ort“ anzusiedeln. Da heutige IM-Systeme jedoch auch die Möglichkeit von Offline-Messages bieten, ist auch eine weitere Einordnung in „Verschiedene Zeit“ und „verschiedener Ort“ denkbar. Ein weiterer Aspekt ist die Unterscheidung der Kommunikationsbeziehungen. Für alle möglichen Kommunikationsbeziehungen haben sich unterschiedliche Begrifflichkeiten gebildet, die jedoch nicht einheitlich verwendet werden. Im einfachsten Fall tauschen zwei Nutzer gleichberechtigt miteinander Nachrichten aus, die auch nur von den beiden Teilnehmern gelesen werden können. Diese

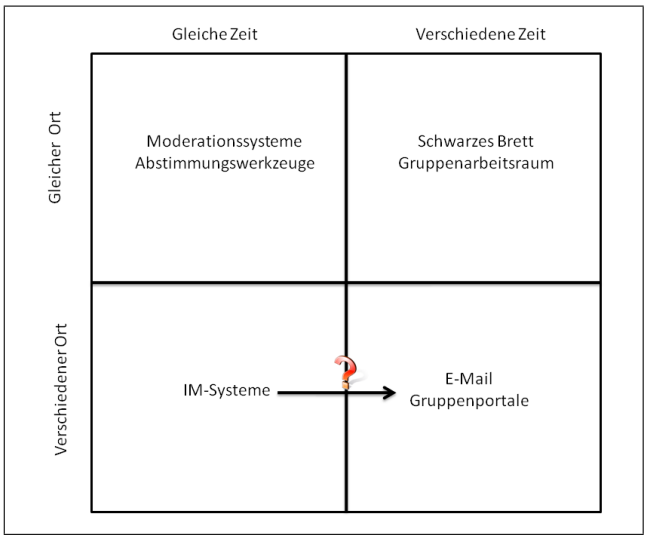

Abbildung 2.1: Einordnung von IM in die Raum-Zeit-Taxonomie nach Johansen *Quelle: [TG07]*

Form der Kommunikation wird zumeist als (privat-) Chat bezeichnet. Eine andere Konstellation ist die Kommunikation in einer Gruppe, bei der eine Person Nachrichten mit mehreren anderen Personen austauscht. Diese Form der Kommunikation wird zumeist als Gruppenchat oder Gruppenkonferenz bezeichnet. Es existieren unterschiedliche Vorgehensweisen und Techniken, wie die Kommunikation im Gruppenchat zustande kommt und welchen Regeln sie folgt. Die zwei am häufigsten verwendeten Mechanismen sind

- **Auf Einladungen basierend:** Ein Nutzer erzeugt einen Raum für den Gruppenchat und versendet Einladungen an einzelne Nutzer. Nur Nutzer die eine Einladung erhalten haben, dürfen am Chat teilnehmen. Für diese Form wird häufig der Begriff Conference-Chat verwendet.

- **Statischer Raum:** Es existieren vordefinierte Räume, zu denen ein Client keine Einladung, sondern lediglich den Namen oder Adresse des Raumes benötigt. Im Rahmen des Jabber Protokolls werden solche Räume üblicherweise als Multi-User-Chat (MUC) bezeichnet (siehe [XMP09a]).

Weiteres Merkmal eines Gruppenchats ist die Möglichkeit der Vergabe von Rollen. Jeder Nutzer im Gruppenchat kann anhand einer vordefinierten Rolle bestimmte Rechte zugeteilt bekommen. Üblicherweise besitzt beispielsweise der Ersteller einer Gruppenkonferenz die Administrationsrechte und darf weitere

Nutzer einladen, Rechte vergeben, Nutzer temporär („*kick*“) oder dauerhaft ausschließen („*ban*“). Ein „normaler“ Teilnehmer des Gruppenchats hingegen besitzt lediglich die Rechte Nachrichten zu empfangen und zu versenden.

2.5 Marktübersicht

Der Markt für IM-Systeme lässt sich nach unterschiedlichen Kriterien gliedern:

- **Protokoll:** Proprietäre- vs. Quelloffene-Protokolle
- **Client:** Multiprotocol- vs. Singleprotocol-Clients
- **Architektur:** Client-Server vs. Peer-to-Peer (P2P)
- **Nutzung:** Kostenfrei vs. Kostenpflichtig
- **Entwicklung:** Proprietäre- vs. Quelloffene Systeme
- ...

Gewichtigster Vertreter sind die IM-Dienste ICQ und AIM (AOL Instant Messaging) des amerikanischen Medienkonzerns AOL, die zusammen auf einen geschätzten Marktanteil von circa 50% kommen [Hei09]. Beide Systeme basieren auf proprietären Protokollen und sind kostenfrei nutzbar, werden allerdings durch Werbeeinblendungen finanziert. Ein weiterer wichtiger Vertreter ist das IM-System Skype, welches über die enthaltene VoIP-Funktionalität internationale Bekanntheit erlangte. Auch Skype basiert auf einem proprietären Protokoll, ist im Gegensatz zu ICQ und AIM aber nicht vollkommen kostenfrei. Das Grundsystem ist zwar kostenfrei nutzbar, aber es werden auch ksotenpflichtige Dienste wie internationale Festnetz-Telefonie angeboten. Die meisten proprietären Systeme folgend dem Client-Server-Paradigma und zeichnen sich durch eine Abschottung gegenüber anderen IM-Systemen aus [Jab09b]. Es existieren zwar auch IM-Systeme für eine Peer-to-Peer-Kommunikation, haben sich aber bisher am Markt noch nicht durchsetzen können.
Ein Client welcher gleichzeitig mit verschiedenen Netzwerken kommunizieren kann, wird als *Multiprotocol-Client* bezeichnet. Die meisten Multiprotocol-Clients versuchen jedoch nicht eigene IM-Systeme zu errichten, sondern versuchen die Funktionalität mehrerer proprietärer Clients in einer Anwendung zu vereinen

und Zusatzfunktionen wie die Unterdrückung von Werbeeinblendungen zu realisieren.
Auf der Seite der quelloffenen-Protokolle ist das XML-basierte Jabber-Protokoll als wichtiger Vertreter zu nennen. Designziel dieses Protokolls war es die Isolierung von verschiedenen IM-Systemen zu minimieren und Interoperabilität zu ermöglichen.
Festzuhalten ist die heutige Dominanz proprietärer IM-Systeme, deren Isolierung voneinander offenbar gewollt ist und einen Teil der Geschäftsstrategie darstellt.

Kapitel 3

Requirements Engineering

Unter dem Begriff „Requirements Engineering“ werden verschiedene Prozesse und Methoden zusammengefasst, die die Anforderungen an ein zu entwickelndes System systematisch erfassen und bewerten. Ziel dieses Kapitels ist daher die Sammlung und Konkretisierung von Anforderungen an das Plugin dieser Arbeit. Dazu werden zunächst aus der gegebenen Ausgangssituation Anforderungen entnommen und diese kategorisiert und analysiert. Darauf aufbauend wird ein erster Systementwurf erstellt, der als konzeptuelle Grundlage für die Realisierung dienen soll.

3.1 Ausgangssituation

Ziel des Plugins ist die Integration einer spontanen aber sicheren Gruppenkommunikation für den bestehenden IM-Client *Spark*, da die bereits in Spark integrierte Implementation für *Jabber Multi User Chats (MUC)* die gegebenen Sicherheitsanforderungen nicht erfüllt. Das Plugin soll als Teil einer übergreifenden Evoting-Applikation eingesetzt werden und neben dem Chat auch geeignete Schnittstellen für die Absicherung einer Audio-/Video-Verbindung bieten. Auch wenn die vorhandene MUC-Implementation den Anforderungen nicht genügt, sollte das Plugin dennoch die typischen Funktionalitäten eines MUC bieten. Es müssen daher Funktionen enthalten sein um Nutzer gezielt zu einer Sitzung einladen zu können und eine einfache Awareness-Unterstützung realisiert werden.
Die Daten des Plugins sollten möglichst über das Jabber-Protokoll versendet und mit Hilfe eines *Diffie-Hellman Key Agreements* verschlüsselt werden. Bevor ein Teil-

nehmer aktiv an einer Sitzung teilnehmen kann muss eine Authentifikation des Nutzers erfolgen, wobei aus Gründen der Spontanität keine *Public Key Infrastructure (PKI)* eingesetzt werden darf, sondern eine Authentifikation über den Audio-/Video-Kanal erfolgen muss. Bereits die Installation des Plugins muss daher ausreichend sein, um an einer Sitzung teilnehmen zu können. Darüber hinaus sollten die Nachrichten einer Sitzung signiert werden und die Gültigkeit der Signaturen von jedem Nutzer überprüfbar sein.

3.2 Anforderungssammlung

Unter einer Anforderung wird in der Softwaretechnik eine gewünschte Eigenschaft eines Systems oder Prozesses verstanden [Sie02]. Die Sammlung von Anforderungen umfasst üblicherweise auch eine Einteilung in eine Anforderungskategorie. Daher wird im Folgenden eine Unterscheidung von funktionalen und nichtfunktionalen Anforderungen vorgenommen [Rup07]:

- **Funktionale Anforderungen:** Funktionale Anforderungen definieren *was* ein System beziehungsweise Prozess leisten soll.
- **Nichtfunktionale Anforderungen:** Nichtfunktionale Anforderungen definieren die Eigenschaften eines Systems oder Prozesses.

Auf eine Bewertung und Priorisierung der Anforderungen wird an dieser Stelle jedoch verzichtet.

3.2.1 Funktionale Anforderungen

1. Die Anwendung muss als Plugin für den IM-Client Spark erstellt werden.
2. Das Plugin soll einen Multi User Chat (MUC) realisieren.
3. Es muss ein Mechanismus zur Einladung von Teilnehmern vorhanden sein.
4. Nachrichten müssen verschlüsselt versendet werden.
5. Jeder Teilnehmer der Sitzung muss die Nachrichten des Chats entschlüsseln können.

6. Teilnehmer einer Sitzung müssen über den Audio-/Video-Kanal authentifiziert werden.
7. Nachrichten müssen signiert versendet werden.
8. Signaturen von Nachrichten müssen für jeden Teilnehmer verifizierbar sein.
9. Das Plugin muss eine Awareness-Unterstützung integrieren.
10. Es müssen Schnittstellen für eine Audio-/Video-Verbindung implementiert werden.

3.2.2 Nichtfunktionale Anforderungen

1. Das Plugin sollte modular aufgebaut sein und von anderen Anwendungen importierbar sein.
2. Die Authentifikation darf keine PKI benötigen.
3. Daten sollen über das Jabber-Protokoll versendet werden.
4. Die Installation des Plugins in Spark ist die einzige Voraussetzung zur Teilnahme an einer Sitzung.
5. Die Kommunikation soll *sicher* sein.

3.3 Anforderungsanalyse

Die aufgeführten Anforderungen beschreiben größtenteils die Funktionalität eines gewöhnlichen MUC im Jabber-Protokoll. Besondere Herausforderung des Plugins stellen daher die gewünschten Sicherheitsanforderungen dar:

- **Integrität:** Unauthorisierte Änderungen am Inhalt einer Nachricht müssen über geeignete Verfahren verhindert beziehungsweise erkannt werden [Egg04].
- **Vertraulichkeit:** Nachrichten dürfen nur von berechtigten Teilnehmern einer Sitzung lesbar sein.

- **Authentizität:** Die Authentizität einer Nachricht ist sichergestellt, wenn sie nachweisbar vom angegebenen Empfänger stammt und Integrität besitzt [Egg04].
- **Verfügbarkeit:** Das System sollte möglichst zu 100% verfügbar sein und muss daher eine Resistenz gegen Denial-of-Service Angriffen bieten [Egg04].

Die in der Ausgangssituation beschriebenen Sicherheitsmaßnahmen decken bereits einen großen Teil der Sicherheitsanforderungen ab. Die Authentifikation der Nutzer ist notwendiger Bestandteil zur Erreichung der Authentizität. Durch die Kombination aus Authentifikation und der Verwendung digitaler Signaturen kann die Authentizität von Nachrichten erreicht werden, die die Integrität der Nachrichten als notwendigen Bestandteil enthält. Da ebenfalls eine Verschlüsselung des Nachrichtentextes beschrieben ist, kann auch die Anforderung der Vertraulichkeit erfüllt werden. Ob die Anforderungen in der Realität jedoch wirklich erfüllt werden, hängt vor allem von der Auswahl geeigneter Verfahren ab. So kann beispielsweise die Verschlüsselung einer Nachricht mit veralteten Algorithmen oder unzureichenden Schlüssellängen nicht als vertraulich bezeichnet werden.

3.4 Systementwurf

Die bisher erhobenen Anforderungen stellen bereits ein Grundgerüst für die Implementation dar. Es müssen allerdings noch wichtige Designentscheidungen getroffen werden, die in den gegebenen Vorgaben nicht näher spezifiziert wurden und daher Gestaltungsfreiheit besitzen.
Die Erstellung einer Sitzung bedarf einer Person, die die Einladungen versendet. Im weiteren Verlauf der Arbeit wird diese Person als der Initiator der Sitzung bezeichnet. Eine weitere Fragestellung ist die Rolle des Jabber-Servers. Einerseits ist es möglich, alle Nachrichten des Plugins über das Jabber-System zu versenden. Dies erfordert jedoch ein Vertrauen in den Jabber-Server und möglicherweise die Erstellung eines Server-Plugins, falls das Jabber-Protokoll nicht alle benötigten Funktionen abdeckt. Ein anderes Modell ist die Trennung von Plugin-Verkehr und dem regulären IM-Verkehr. In diesem Modell kommunizieren die Plugins auf direktem Wege miteinander und durchlaufen keinen Jabber-Server

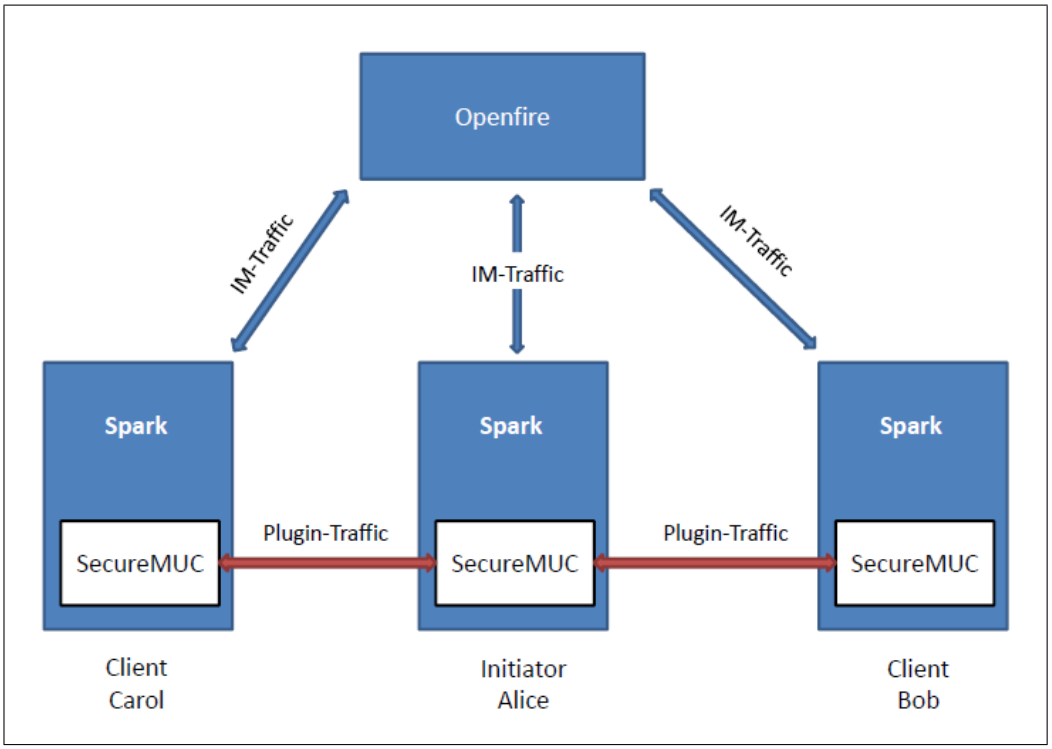

Abbildung 3.1: Systementwurf: Trennung von Plugin- und IM-Verkehr
Quelle: Eigene Darstellung

auf dem Weg der Zustellung. Da bei der Zustellung von Nachrichten jedoch potentiell mehrere Server durchlaufen werden müssen und das Plugin möglichst unabhängig von einem bestimmten Jabber-Server betriebsfähig sein soll, wird im weiteren Verlauf der Ansatz der Trennung von Plugin- und IM-Verkehr verfolgt. Dieses Modell erfordert jedoch die Verwendung einer Plugininstanz als Plugin-Server, der die Kommunikation aller beteiligten Clients übernimmt. Da der Initiator durch die Erstellung der Sitzung bereits Sonderrechte besitzt und damit immer als erster Teilnehmer eine Sitzung betritt, wird die Aufgabe des Plugin-Servers beim Initiator angesiedelt (siehe Abbildung 3.1). Jede Installation des Plugins muss jedoch prinzipiell sowohl die Rolle des Initiators wie auch eines regulären Clients wahrnehmen können.
Mit der Vorgabe die Authentifikation der Nutzer nicht automatisiert, sondern manuell über die Audio-/Video-Verbindung zu realisieren, muss auch eine verifizierende Instanz eingeführt werden, die die Authentifikation von Clients übernimmt. Auch an dieser Stelle bietet sich der Initiator als zuständige Instanz an.

Durch die Einladung von bestimmten Personen weiß der Initiator als einzige Person, wer berechtigt ist an der Sitzung teilzunehmen und durch die Rolle als Plugin-Servers ist der Initiator ohnehin dazu verpflichtet, die entsprechende Infrastruktur zur Authentifikation bereitzustellen.
Die Rolle des Initiators lässt sich demnach folgendermaßen charakterisieren (vgl. Abbildung 3.2):

- **Ersteller der Sitzung:** Der Initiator erstellt eine Sitzung und lädt Personen zur Teilnahme ein.
- **Plugin-Server:** Der Initiator übernimmt die Rolle des Plugin-Servers und ermöglicht dadurch die Kommunikation der Plugins untereinander.
- **Verifizierende Einheit:** Der Initiator übernimmt die Authentifikation der eingeladenen Clients über eine Audio-/Video-Verbindung.
- **Teilnehmer:** Der Initiator besitzt alle Rechte die auch regulärer Client besitzt.

Weitere Fragen, wie etwa die Schlüsselverteilung, den Ablauf der Authentifikation oder das Nachrichtenformat erfordern zunächst eine genauere Betrachtung und Auswahl geeigneter Konzepte und werden in den folgenden Kapiteln vorgenommen.

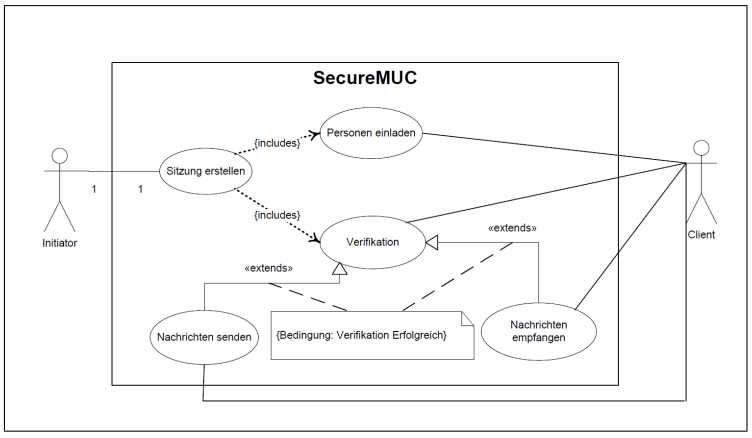

Abbildung 3.2: Anwendungfalldiagramm von SecureMUC *Quelle: Eigene Darstellung*

Kapitel 4

Diffie-Hellman Key Agreement

4.1 Grundlagen

Im Jahr 1976 haben Diffie und Hellman ein gleichnamiges Verfahren zur sicheren Vereinbarung von Schlüsseln (*Key Agreement*) vorgestellt [Egg04]. Die Besonderheit des Verfahrens liegt in der dezentralen Berechnung eines geheimen Sitzungsschlüssels (Session Key) aus einem öffentlichen und einem privaten Schlüssel. Anders als bei anderen Verfahren ist es beim Diffie-Hellman Key Agreement nicht notwendig, einen geheimen Schlüssel über ein unsicheres Medium wie das Internet zu transportieren. Wollen die Personen Alice und Bob einen geheimen Schlüssel miteinander vereinbaren, so werden nach Diffie und Hellman folgende Schritte ausgeführt:

1. Alice erzeugt eine Primzahl `p` und eine Primitivwurzel `g mod p` mit der Bedingung

   $$2 \leq g \leq p - 2 \tag{4.1}$$

   Die entstandenen Zahlen `g` und `p` stellen die Parameter für das Key Agreement dar und müssen nicht geheim gehalten werden.

2. Alice und Bob erzeugen jeweils eine Zufallszahl `a` (Alice) beziehungsweise `b` (Bob) aus der Menge $\{1, ..., p - 2\}$. Die Zahlen `a` und `b` stellen die privaten Schlüssel von Alice und Bob dar und müssen geheim gehalten werden.

3. Anschließend berechnen Alice und Bob die öffentlichen Schlüssel A (Alice) und B(Bob) aus den Parametern g und p sowie dem jeweiligen privaten

Schlüssel a beziehungsweise b:
$A = g^a \bmod p$ und $B = g^b \bmod p$
Die öffentlichen Schlüssel werden daraufhin von Alice und Bob ausgetauscht.

4. Sind die öffentlichen Schlüssel ausgetauscht, können Alice und Bob beide den geheimen Sitzungsschlüssel K berechnen: $K = B^a \bmod p$ beziehungsweise $K = A^b \bmod p$.

Die Sicherheit des Verfahrens wurzelt im diskreten Logarithmus-Problem [Egg04]. Bis zum heutigen Tag existieren keine effizienten Algorithmen um den diskreten Logarithmus zu berechnen [Buc03]. Damit ist die Sicherheit des Verfahrens an ein bis dato ungelöstes mathematisches Problem gebunden.
Eine Authentifizierung oder Schlüssel-Verifikation ist im Diffie-Hellman Key Agreement jedoch nicht vorgesehen. Des Weiteren stellen die öffentlichen und privaten Schlüssel kein asymmetrisches Kryptosystem dar [Egg04], sind also weder für digitale Signaturen noch für die asymmetrische Verschlüsselung zu verwenden. Außerdem ist die Verwendung des Diffie-Hellmann-Verfahrens in der hier beschriebenen Reinform für die Verwendung zwischen zwei Parteien ausgelegt. Daher ist die Verwendung des Key Agreements nicht ohne weiteres für die Absicherung einer Gruppenkommunikation und damit für die Realisierung von SecureMUC zu verwenden.

4.2 Angriffe

Da im Diffie-Hellman Key Agreement keine Überprüfung der Authentizität vorgesehen ist, ist das Verfahren anfällig für *Man-in-the-Middle*-Angriffe [Egg04]. So könnten Alice und Bob wie im Beispielszenario in Abbildung 4.1 davon ausgehen, einen gemeinsamen Schlüssel $K_{A,B}$ miteinander zu vereinbaren. In Wirklichkeit wird das Diffie-Hellman-Verfahren jedoch zwischen Alice und einem Angreifer Eve mit dem resultierenden Schlüssel $K_{A,E}$ sowie Eve und Bob mit dem Schlüssel $K_{B,E}$ durchgeführt. Möchte nun Alice eine Nachricht an Bob senden, wird sie diese mit dem Schlüssel $K_{A,B}$ verschlüsseln, aber ohne es zu bemerken den Schlüssel $K_{A,E}$ verwenden. Der Angreifer Eve ist nun in der Lage die Nachricht von Alice zu entschlüsseln und beliebig zu verändern, weiterzuleiten oder zu verwerfen. Leitet Eve die Nachricht an Bob weiter, wird er dazu den Schlüssel

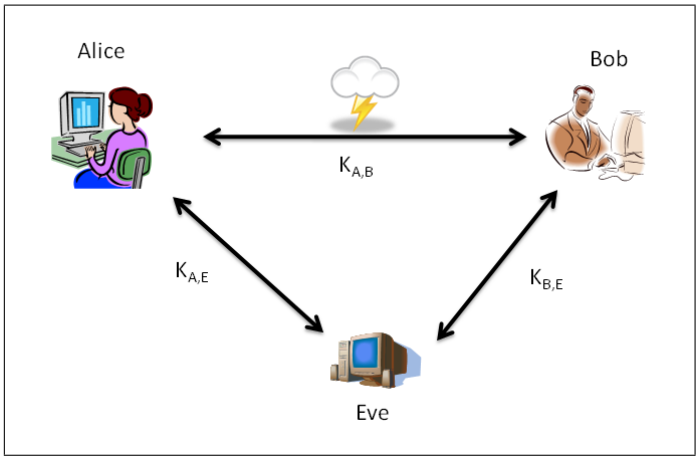

Abbildung 4.1: Beispielzenario eines Man-in-the-Middle-Angriffs *Quelle: [Egg04]*

$K_{B,E}$ verwenden und Bob ist demnach nicht in der Lage, die Manipulation beziehungsweise das Mitlesen von Eve zu bemerken.
Durch die in SecureMUC vorgesehene Authentifikation über die Audio-/Video-Verbindung kann dieses Problem jedoch unterbunden werden. Die Frage nach dem genauen Modell der Authentifikation bleibt jedoch weiter offen und kann im Diffie-Hellman-Verfahren nicht gefunden werden.

4.3 Herausforderungen im Gruppenchat

Wie bereits in den Grundlagen des Kapitels erwähnt wurde, ist das Diffie-Hellman-Verfahren in der Reinform für die Schlüsselvereinbarung von zwei Parteien vorgesehen. Soll Diffie-Hellman hingegen im Kontext einer Gruppenkommunikation eingesetzt werden, steht das Protokoll vor neuen Herausforderungen (siehe Abbildung 4.2) [Mic09]:

- **Member Join:** Ein Neuer Nutzer wird in die Gruppe aufgenommen.
- **Member Leave:** Ein Nutzer verlässt die Gruppe.
- **Mass Join:** Mehrere Nutzer werden gleichzeitig in die Gruppe aufgenommen.
- **Mass Leave:** Mehrere Nutzer verlassen gleichzeitig die Gruppe.

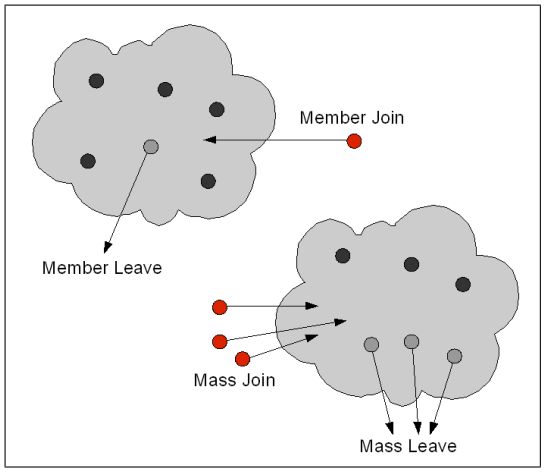

Abbildung 4.2: Herausforderungen des Einsatzes des Diffie-Hellman-Verfahrens im Kontext der Gruppenkommunikation *Quelle: [Mic09]*

Wird über Diffie-Hellman ein gemeinsamer Session-Key für alle Mitglieder der Gruppe ausgehandelt, so haben alle genannten Operationen einen Einfluss auf Berechnung des Session-Keys. Dadurch wird es zwangsläufig zu einem häufigen Wechsel des Session-Keys kommen. Im Sinne einer *Perfect Backward Secrecy* [MDH01] ist ein solches Protokollverhalten zwar dienlich, steigert jedoch den benötigten Rechenaufwand und senkt damit die Performance der Gruppenkommunikation. Soll der entsprechende Session-Key darüber hinaus noch verifiziert werden, ist ein solches Protokoll in der Praxis nicht mehr einsetzbar.

4.4 Anwendungsbeispiel: ZRTP

Das ZRTP-Protokoll wurde von Phil Zimmermann entwickelt und verwendet ein Diffie-Hellman Key Agreement zum Aufbau einer SRTP (Secure Real-Time Transport Protocol)-Sitzung [Phi09].
Daher ist das ZRTP-Protokoll für die Absicherung von VoIP-Verbindungen konzipiert und wird bereits in der Referenzimplementierung *ZFone* [Zfo09] eingesetzt.
Die Besonderheit des Verfahrens ist jedoch nicht der Einsatz von Diffie-Hellman zur Absicherung einer VoIP-Verbindung, sondern die im Protokoll vorgesehene Verifikation des ausgehandelten Session-Keys.
Das ZRTP-Protokoll sieht nach erfolgreichem Abschluss des Diffie-Hellman- Verfahrens den Vergleich eines *Short Authentication String* (SAS) über die VoIP- Ver-

bindung vor. Der SAS wird als Hashwert des Session-Keys berechnet und als kurze Zeichnfolge dargestellt. Ist dieser Hashwert bei beiden Teilnehmern des Protokolls identisch, kann ein Man-in-the-Middle-Angriff ausgeschlossen werden [Phi09].

Kapitel 5

Das Jabber-Protokoll

5.1 Geschichte

Die Grundidee des Jabber Protokolls stammt vom amerikanischen Softwareentwickler Jeremie Miller, der 1999 erstmals das Konzept eines XML-basierten Standards für IM-Systeme veröffentlichte. Ziel sollte ein offener Standard für IM-Systeme sein, der die Interoperabilität verschiedener Systeme ermöglicht und sich damit maßgeblich von der Isolierung proprietärer Systeme unterscheidet [Jab09b]. Unter Leitung von Jeremie Miller wurde daraufhin eine Open Source-Community aufgebaut, die die Weiterentwicklung und Spezifikation des Protokolls übernahm und die 2001 durch die gemeinnützige Stiftung der *Jabber Software Foundation* (JSF) ergänzt wurde.
Der Kern des Protokolls wurde 2004 unter dem Namen *„Extensible Messaging and Presence Protocol“* (XMPP) als IETF-Standard [Int09d] angenommen und wird fortan unter diesem Namen weiterentwickelt. Nach Annahme des Standards unter dem Namen XMPP wurde auch die *Jabber Software Foundation* in *XMPP Standards Foundation* umbenannt und wird heute noch unter diesem Namen geführt [XMP09d].

5.2 Grundlagen

Die wichtigsten Grundlagen des Jabber-Protokolls werden im durch die IETF zugelassenen Standards *XMPP-Core* (RFC 3920) [Int09d] definiert. Die darin be-

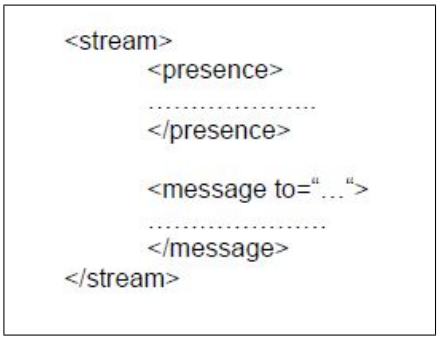

Abbildung 5.1: Aufbau eines einfachen XML-Streams *Quelle: [Int09d]*

schriebenen Vorschriften formen das Grundgerüst aller Jabber-Protokolle und Protokollerweiterungen. Eine detaillierte Spezifikation des Jabber-Protokolls ist jedoch auf drei weitere IETF-Standards verteilt:

- **RFC 3921:** Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence [Int09e]
- **RFC 3922:** Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM) [Int09f]
- **RFC 3923:** End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP) [Int09g]

Neben den offiziellen IETF-Standards existieren noch eine Reihe von Protokollerweiterungen, die zwar durch die XMPP Standards Foundation als offizielle XMPP-Erweiterung zugelassen wurden, aber keinen Standard der IETF darstellen. Voraussetzung für die Zulassung als XMPP-Erweiterung (XMPP-Extension (XEP)) durch die XMPP Standards Foundation stellt neben dem Durchlaufen eines internen Qualitätssicherungsverfahrens jedoch die Umsetzung der IETF-Standards dar.
Grundsätzlich erfolgt die Kommunikation im Jabber-Protokoll über *XML-Streams*. Ein XML-Stream wird dabei als Container für beliebig viele XML-Elemente verstanden, die über ein Netzwerk zwischen zwei beliebigen Entitäten ausgetauscht werden. Die Elemente innerhalb des XML-Streams werden als *XML-Stanzas* bezeichnet und müssen in sich geschlossene semantische Einheiten darstellen [Int09d]. Alle XML-Stanzas die während einer Sitzung zwischen zwei Entitäten ausgetauscht werden stellen somit *Child-Nodes* eines XML-Streams dar (siehe Abbildung 5.1).

Die Adressierung erfolgt im Jabber-Protokoll anhand eines *Jabber Identifier* (JID). Jeder Knoten der mit einem Jabber-Server verbunden ist und XMPP-fähig ist, kann anhand eines JID adressiert werden. Eine JID besteht nach Definition des XMPP-Core immer aus *Node Identifier*, *Domain Identifier* und *Resource Identifier* in der Form „Node Identifier@Domian Identifier/Resource Identifier“ [Int09e].
Eine gültige JID könnte beispielsweise „Alice@uni-koblenz.de/mobile“ sein.
Damit wird die Nutzerin Alice unter dem Jabber-Server der Universität-Koblenz

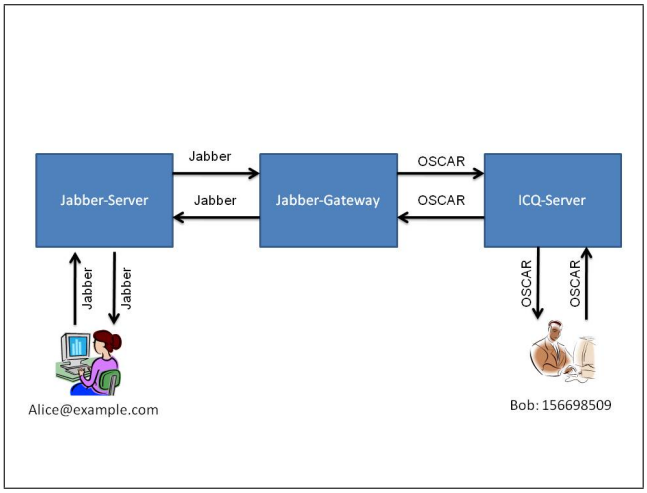

Abbildung 5.2: Aufgabe des Jabber-Gateways *Quelle: [Int09e]*

an der Ressource „mobile“ adressiert.
Die Verwendung von Ressourcen ist eine weitere Besonderheit des Jabber-Protokolls, die es ermöglicht, mit mehreren Jabber-Clients gleichzeitig unter einem Account verfügbar zu sein. So könnte beispielsweise die Nutzerin Alice einen Jabber-Client an ihrem Arbeitsplatz mit dem JID „Alice@uni-koblenz.de/work“ und einen Jabber-Client auf ihrem Mobiltelefon mit dem JID „Alice@uni-koblenz.de/mobile “ gleichzeitig betreiben. Andere Nutzer könnten Alice nun gezielt durch Angabe des jeweiligen JID auf ihrem Mobiltelefon oder am Arbeitsplatz erreichen. Weiterhin ist es möglich die Ressourcen mit Prioritäten zu versehen, anhand derer der ausliefernde Server entscheiden kann, an welche Ressource eine Nachricht versendet werden soll. So könnte Alice beispielsweise ihre Jabber-Clients so konfigurieren, dass die Nachrichten immer auf ihr Mobiltelefon weitergeleitet werden, wenn sie am Arbeitsplatz nicht verfügbar ist [Shi02].
Weiteres Designziel des Jabber-Protokolls war die Interoperabilität mit anderen

IM-Systemen. Demnach sollte es möglich sein, ein Jabber-System sowohl mit anderen Jabber-Systemen als auch mit beliebigen anderen IM-Systemen zu verbinden und die Kommunikation über die Systemgrenzen hinaus zu ermöglichen. Dazu wurde im Jabber-Protokoll ein Service namens *Jabber Transport* bzw. *Jabber-Gateway* vorgesehen. Ein Jabber-Gateway (-Transport) ist ein Service, der Nutzer eines anderen IM-Systems in das Jabber-System integriert und dabei die notwendigen Adress- und Protokollübersetzungen vornimmt [Int09f]. Möchte beispielsweise Alice aus einem Jabber-Netz eine Nachricht an Bob in das ICQ-Netz schicken (siehe Abbildung 5.2), so wird die Nachricht erst an den Jabber Server und von dort an das Jabber-Gateway weitergeleitet. Im Jabber-Gateway wird daraufhin die Nachricht aus dem XMPP-Format in das proprietäre OSCAR-Protokoll übersetzt und anschließend an den jeweiligen ICQ-Server weitergeleitet.

5.3 Jabber-Kernprotokolle

Das Jabber-Protokoll wird zumeist als ein einheitliches Protokoll verstanden, welches sämtliche Interaktion der Teilnehmer definiert. Innerhalb des Jabber-Protokolls existieren jedoch drei Teilprotokolle, die jeweils unterschiedliche Aufgaben erfüllen [Shi02]:

- **Message Protocol:** Dient dem Austausch von Textnachrichten
- **Presence Protocol:** Dient dem Austausch von Präsenz- und Verfügbarkeitsinformationen
- **IQ Protocol:** Dient dem Austausch beliebiger Informationen und der Erweiterbarkeit des Protokolls

Zwischen den einzelnen Teilprotokollen existieren viele Gemeinsamkeiten, doch jedes Teilprotokoll besitzt einen eigenen Regelsatz, der zur jeweiligen Aufgabe innerhalb des gesamten Jabber-Protokolls angepasst ist.
Der Nutzen einer genauen Differenzierung der einzelnen Teilprotokolle ergibt sich insbesondere bei der Entwicklung neuer Anwendungen oder Erweiterungen, da sowohl von Jabber-Servern als auch Jabber-Clients je nach Protokoll ein unterschiedliches Verhalten erforderlich ist.

5.3.1 Message Protocol

```
<message to='bob@example.com/work' from='alice@example.com/mobile' type='chat'>
  <thread>a96534b18164252</thread>
  <subject>Welcome </subject>
  <body>Welcome to jabber, Bob!</ body>
</message>
```

Abbildung 5.3: Beispielnachricht des Message Protocol *Quelle: Eigene Darstellung*

Das *Message Protocol* dient dem Versenden von Textnachrichten und stellt damit das zentrale Element im Jabber-Protokoll dar. Innerhalb des Protokolls existieren fünf unterschiedliche *Message Types*, die über das Message Protocol gesendet werden können:

1. **Groupchat:** Nachrichten die Teil einer Gruppenkonferenz sind
2. **Chat:** Nachrichten die Teil einer *one-to-one*-Konversation sind und in Form eines chronologischen Verlaufs darstellbar sind
3. **Normal:** Einzelne Nachrichten, die nicht zu einer Konversation gehören, sondern selbst eine abgeschlossene Einheit darstellen
4. **Headline:** Nachrichten die durch einen automatisierten Service wie RSS (*Really Simple Syndication* [RSS09]) versendet werden
5. **Error:** Nachrichten die automatisch versendet werden und auf eine Fehlfunktion hinweisen

Die Wurzel des XML-Stanza im Message Protocol stellt das <message>-Element dar, zu dem die Attribute „from“, „to“ und „type“ hinzugefügt werden (siehe Abbildung 5.3):

- **from:** JID des Absenders der Nachricht
- **to:** JID des Empfängers der Nachricht
- **type:** Identifiziert den Message Type der Nachricht

Die Attribute des <message>-Elements stellen die Steuerungsinformationen dar, anhand derer ein Server entscheiden kann, an welchen Teilnehmer eine Nachricht weitergeleitet soll und die ein Client benötigt, um den Absender einer eingehenden Nachricht festzustellen.
Das <message>-Element kann eine Reihe von *Child Nodes* besitzen, in denen der eigentliche Inhalt einer Nachricht transportiert wird:

- **thread:** Tracking-Informationen, die die Zugehörigkeit einer Nachricht zu einer bestimmten Sitzung kennzeichnen.
- **subject:** Thema beziehungsweise Überschrift einer Nachricht
- **body:** Inhalt einer Nachricht

Es existieren in der Spezifikation des Message Protocol zwar Vorschriften welche Elemente als *Child Nodes* des <message>-Elements zulässig sind, aber nicht welche Elemente in welcher Anzahl vorhanden sein müssen beziehungsweise vorhanden sein dürfen. Damit ist sowohl das Fehlen als auch Mehrfachvorkommen der aufgeführten Child Nodes zulässig.

5.3.2 Presence Protocol

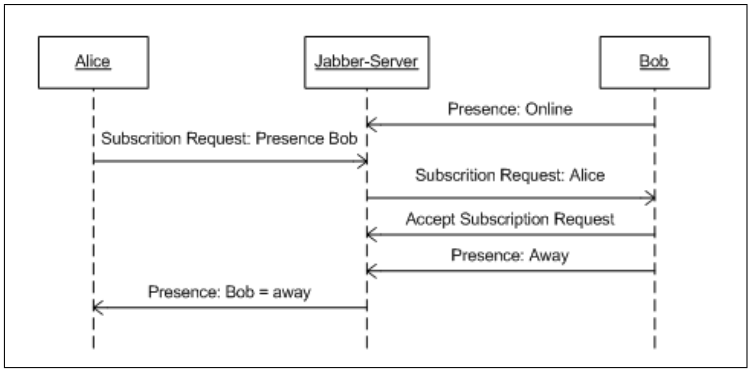

Abbildung 5.4: Funktionsweise des Presence Protocol *Quelle: Eigene Darstellung*

Mit dem Presence Protocol wird das Konzept der „Awareness“ in das Jabber-Protokoll integriert. Auch mit dem Presence Protocol ist es möglich, Nachrichten von einem Client zu einem anderen Client zu senden. In der Spezifikation

des Presence Protocol wird jedoch empfohlen, ein Anmeldungs- und Broadcast-Verfahren zu verwenden (siehe Abbildung 5.4). Möchte beispielsweise Alice die Präsenz- und Verfügbarkeitsinformationen von Bob empfangen, muss Alice zuvor ein *Subscription Request* an Bob senden, um seine Erlaubnis zu erhalten, seinen jeweiligen Status einsehen zu dürfen. Erteilt Bob diese Erlaubnis, wird Alice beim Jabber-Server als *Subscriber* registriert und wird zukünftig vom Server automatisch über Statusänderungen von Bob informiert. Die damit von Bob erteile Erlaubnis an Alice gilt jedoch lediglich einseitig, wodurch Bob auch von Alice eine Genehmigung benötigt, um über deren Status zukünftig informiert zu werden.
Der Aufbau eines XML-Stanza des Presence Protocol erinnert an den Aufbau im Message Protocol, wobei jedoch die Wurzel das <presence>-Element darstellt:

```
<presence>
     <show>away</show>
     <status>Coffee Break</status>
     <priority>1</priority>
   </presence>
```

Anders als jedoch beim Message Protocol werden nicht alle Stanzas mit den Steuerungsinformationen in Form der Attribute „from“, „to“ und „type“ versehen. Diesbezüglich existieren zahlreiche Vorschriften, wann welches dieser Attribute vorhanden sein darf beziehungsweise vorhanden sein muss und welche Semantik es im jeweiligen Kontext besitzt. Informiert beispielweise ein Client den Server über seine Zustandsänderung vom Status „online“ auf den Status „away“, so ist das <presence>-Element alleine ohne Attribute vorgesehen. Versendet der Server hingegen die Zustandsänderung an die Subscriber des Nutzers, so sind die Attribute „from“ und „to“ vorgesehen. Das Attribut „from“ erhält dabei den vollständigen JID des Nutzers, dessen Statusänderung Gegenstand der Nachricht ist. Beim Attribut „to“hingegen wird auf Angabe einer speziellen Ressource verzichtet, da die Statusänderung auf allen Ressourcen des empfangenden Clients verfügbar sein soll [Int09e].
Der Inhalt einer Nachricht im Presence Protocol wird in Form von Child Nodes mit dem <presence>-Element verknüpft. Mögliche Elemente sind:

- **show:** Enthält den Status eines Nutzers aus einer Reihe vordefinierter Stati wie „away“, „dnd“ oder „extended away“
- **status:** Natürlichsprachliche Beschreibung des Zustandes
- **priority:** Setzt die Priorität der Ressource, von der die Statusänderung stammt (*siehe Message Protocol*)

Wie auch beim Message Protocol ist das Vorhandensein der aufgeführten Child Nodes rein fakultativ. So wird beispielsweise ein Fehlen jeglicher Child Nodes als der Status „online“ interpretiert. Mehrfachvorkommen der gleichen Elemente werden jedoch in der Spezifikation des Presence Protocol ausgeschlossen [Int09e].

5.3.3 IQ Protocol

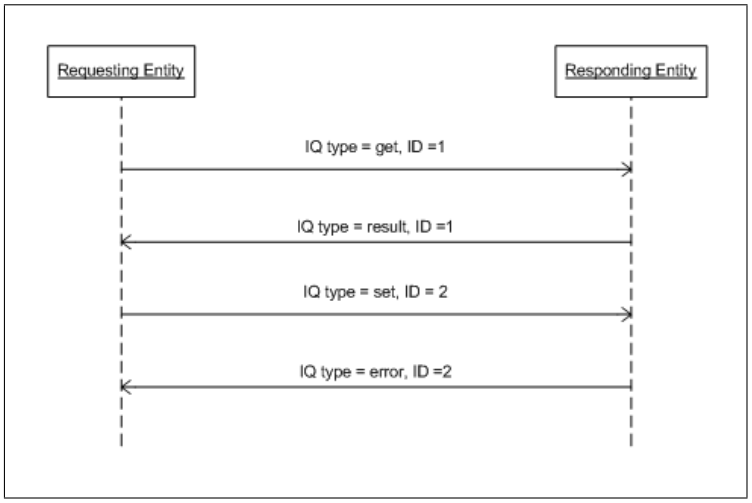

Abbildung 5.5: Beispiel: Ablauf des IQ Protocol *Quelle: [Int09d]*

Das *Information/Query Protocol* (IQ Protocol) stellt ein einfaches Request/Response-Rahmenwerk zur Verfügung, über dieses Informationen beschafft, verteilt und verifiziert werden können. Besonderes Merkmal dabei ist die vielfältige Erweiterbarkeit, da das Protokoll lediglich Regeln für den Request/Response-Mechanismus auf XML-Ebene definiert, konkrete Anwendungen jedoch offen lässt. Typische Anwendungen des IQ Protocol sind beispielsweise die Account-Registrierung, Anbindung an externe Datenbanken oder die Entwicklung eigener Anwendungen. Das Grundprinzip des Protokolls sieht vor, dass eine Entität (*Requesting En-*

tity) eine Anfrage (*Request*) an eine andere Entität (*Responding Entity*) stellt und darauf eine der Request entsprechende Antwort (*Response*) erhält.
Die Wurzel eines IQ-Stanzas stellt das <iq>-Element dar, das mit den bekannten Steuerungsinformationen in Form der Attribute „from“, „to“ „type“ versehen wird:

```
<iq from='alice@uni-koblenz.de' to='bob@uni-koblenz.de'
type='get' id='561245873142'>
<querey xmlns='http://www.uni-koblenz.de/exampleQuery'>
<item jid='carol@uni-koblenz.de/work'>
    </group>
</item>
 </iq>
```

Eine Besonderheit stellt das für IQ-Stanzas verpflichtende „ID“-Attribut dar. Durch das Konzept des Request/Response-Verfahrens müssen Request und Response eindeutig miteinander verknüpfbar sein. Daher muss das „ID“-Attribut mit einem Wert belegt sein, der auch bei parallelen und möglicherweise entgegengesetzten Requests die Zugehörigkeit einer Response zu einer bestimmten Request ermöglicht [Int09d]. Vorgesehen sind 4 unterschiedliche Typen von IQ-Stanzas:

- **Get:** Anfrage der *Requesting Entity* nach bestimmten Information
- **Set:** Setzen beziehungsweise Aktualisierung von Daten durch die Requesting Entity
- **Result:** Liefert das Ergebnis auf eine Anfrage oder bestätigt den erfolgreichen Erhalt von Daten
- **Error:** Signalisiert einen Fehler im Verfahren, beispielsweise wenn die *Responding Entity* das Request der *Requesting Entity* nicht verarbeiten kann

Bei Paketen des Typus „get“ oder „set“ kann dem <iq>-Element daraufhin genau eine beliebige Child Node hinzugefügt werden, die in semantischer Hinsicht eine Request darstellt. Es muss allerdings darauf geachtet werden, dass ein Verweis auf einen XML-Namensraum (*Namespace*) gegeben wird, der die enthaltene Request definiert und dem Empfänger des Stanzas die Verarbeitung ermöglicht. Der Empfänger eines Stanzas kann daraufhin entweder mit einer Nachricht des

Typs „result“ oder des Typs „error“ antworten. Stanzas des Typs „result“ dürfen genau eine oder keine direkte Child Node besitzen.
War eine Request nicht erfolgreich, beispielsweise weil die Responding Entity die Request nicht verarbeiten kann, so muss die Responding Entity mit einem Stanza des Typs „error“ antworten.

5.4 Client-Authentifizierung

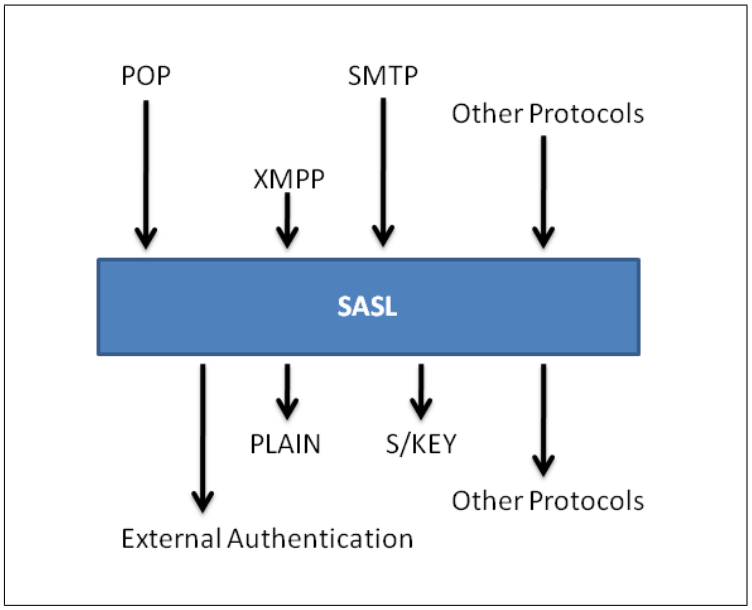

Abbildung 5.6: Die Funktion von SASL *Quelle: [Int09h]*

Mit dem Begriff „Authentifizierung“ wird allgemein der Prozess der Authentifikation bezeichnet, der die behauptete Identität eines Subjekts oder Objekts anhand von charakteristischen Merkmalen überprüft [Egg04]. Bezogen auf die Client-Authentifikation im Jabber-Protokoll bedeutet dies, das der Client gegenüber dem Server einen Nachweis seiner Identität erbringen muss, um Zugang zum System zu erhalten. In vielen Systemen wird zu diesem Zweck die Abfrage von Nutzernamen und Passwort verwendet. Von dieser Strategie weicht auch das Jabber-Protokoll nicht ab, allerdings existieren zwei verschiedene Verfahren der Authentifikation:

1. **Jabber Community:** Die ursprünglich von der Jabber Community vorge-

sehene Authentifizierungsmaßnahme besteht aus dem Abfragen von Nutzernamen und Passwort in Form von Hashwerten oder als Klartext

2. **XMPP-Standard:** Die in den IETF-Standard eingegangene Authentifizierungsmaßnahme schreibt die Verwendung von SASL (*Simple Authentication and Security Layer*) zur Client-Authentifizierung vor

Auch wenn in den letztendlichen Standard der IETF die ursprüngliche Authentifizierungsmethode der Jabber-Community nicht eingegangen ist, so existieren dennoch Systeme die diese Form der Authentifikation als Ausweichmöglichkeit weiterhin anbieten [Jab09a]. Die in den XMPP-Standard eingegangene Technologie SASL stellt ein Framework für verschiedene Verfahren der Authentifizierung zur Verfügung und stellt ebenfalls einen IETF-Standard [Int09h] dar, der sich allerdings noch im Status „proposed“ befindet und damit gegenwärtig noch kein endgültiger Standard ist [Int09b].
Verstanden werden kann SASL als ein Interface, welches eine einheitliche Schnittstelle zwischen Applikationsprotokollen wie dem Jabber-Protokoll und Authentifizierungsprotokollen wie „CRAM-MD5“ [Int09c] oder „S/KEY“ [Int09a] beschreibt (siehe Abbildung 5.6).
Möchte sich ein Client (*Initiating Entity*) gegenüber einem Server (*Receiving Entity*) authentifizieren, so werden folgende Schritte durchlaufen [Int09d]:

1. Der Client sendet ein *SASL Authentication Request* an den Server
2. Der Server sendet eine Liste mit den von ihm unterstützen Authentifikationsverfahren an den Client
3. Der Client wählt ein Verfahren aus der empfangenen Liste aus teilt dem Server das gewünschte Verfahren mit
4. Falls es das ausgewählte Verfahren benötigt, sendet der Server eine *Challenge* an den Client (*Challenge-Response-Verfahren*)
5. Der Client antwortet auf die *Challenge* des Servers mit einer geeigneten *Response*
6. Falls es das Verfahren vorsieht, werden weitere Wiederholungen des Challenge-Response-Verfahrens ausgeführt, bis entweder Client oder Server das Verfahren abbrechen oder der Server die erfolgreiche Authentifizierung mitteilt

Die Verwendung von SASL ermöglicht zwar größere Flexibilität und Interoperabilität von verschiedenen Client/Server-Kombinationen, doch es lässt keine allgemeine Betrachtung der der Client-Authentifizierung im Jabber-Protokoll zu. Die Auswahl eines speziellen Authentifikationsprotokolls erfolgt fallabhängig, je nach Implementation und Konfiguration von Client und Server. Demnach ist die Betrachtung eines konkreten Systems erforderlich, um Aussagen über die Güte der Client-Authentifizierung zu treffen.

5.5 Verschlüsselung

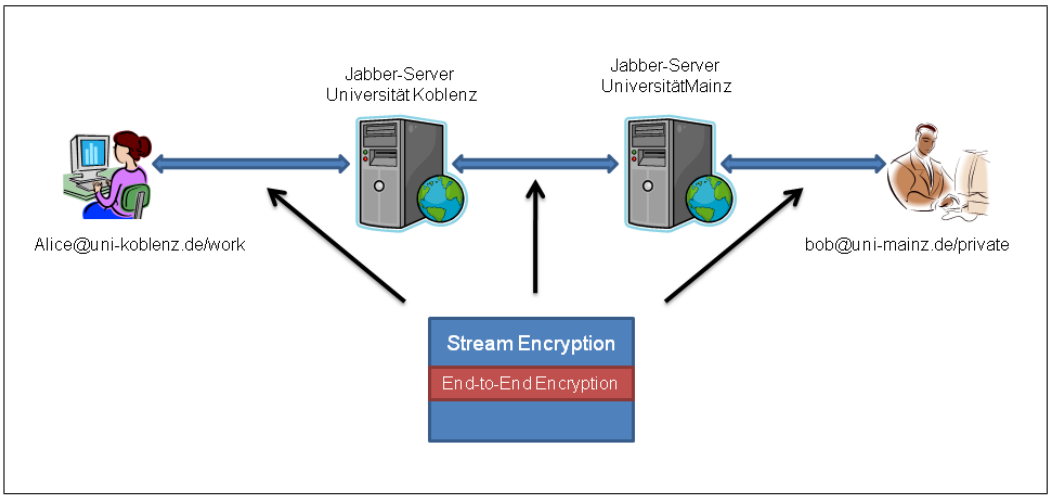

Abbildung 5.7: Die Verschlüsselung im Jabber-Protokoll *Quelle: Eigene Darstellung*

Grundsätzlich können drei unterschiedliche Kanäle durch Verschlüsselung im Jabber-Protokoll geschützt werden:

1. **Client to Server:** Die Verschlüsselung zwischen Client und Server bezieht sich auf den gesamten XML-Stream einer Sitzung und wird häufig mit *Stream Encryption* [XMP09c] bezeichnet.

2. **Client to Client:** Die Verschlüsselung der ausgetauschten Nachrichten zwischen zwei Clients wird häufig als *End-to-End-* oder *Stanza Encryption* bezeichnet. Server, die Nachrichten auf dem Weg der Zustellung durchlaufen,

können lediglich die für die Zustellung notwendigen Steuerungsinformationen einsehen.

3. **Server to Server**: Müssen Nachrichten auf dem Weg der Zustellung mehr als einen Server durchlaufen, so kann auch die Kommunikation der Server untereinander verschlüsselt werden. Auch die Verschlüsselung dieses Kanals fällt unter das Aufgabengebiet der *Stream Encryption*.

Die Verschlüsselung der einzelnen Kanäle schließt sich nicht gegenseitig aus, sondern findet im Idealfall in Ergänzung zueinander statt (siehe Abbildung 5.7). Die Stream Encryption dient dabei als äußere Hülle, die den gesamten Verkehr zwischen zwei Entitäten verschlüsselt, während die übertragenen Stanzas selbst mit End-to-End Encryption geschützt werden können.
Für den Fall der Stream Encryption wird in der Spezifikation des Jabber-Protokolls der Einsatz von *Transport Layer Security* (TLS) empfohlen [Int09d]. Das TLS-Protokoll wurde von der IETF standardisiert [Int09i] und stellt den Nachfolger des etablierten *Secure Socket Layer* (SSL)-Protokolls dar.
Der Fall der Stanza Encryption existiert von Seiten der XMPP Standards Foundation kein einheitlicher Standard. Es wurden zwar bereits Versuche einer Standardisierung unternommen (vgl. [XMP09b]), die jedoch bisher keine Zulassung erreichen konnten.

5.6 Multi User Chat (MUC)

Der *Multi User Chat (MUC)* im Jabber-Protokoll wird über das Message Protocol mit Message Type „Groupchat“ realisiert. In der Erweiterung XEP-0045 [XMP09a]) wurde der MUC weiter spezifiziert und sieht unterschiedliche Rollen wie Moderator, Participant (Teilnehmer) oder Visitor (Besucher) vor. Weiterhin wurden *Room Types* vorgesehen, die entweder permanent oder für die Dauer einer Sitzung auf dem Jabber-Server eingerichtet werden. So existieren beispielsweise permanente öffentliche Räume, die mit der Arbeitsweise von IRC vergleichbar sind oder „Members-Only Rooms“, die auf Einladungen basieren.
Die gewünschten Funktionalitäten mit Hinblick auf Client-Authentifizierung oder End-to-End Encryption für SecureMUC werden jedoch nicht unterstützt.

Kapitel 6

Spark IM

6.1 Grundlagen

Der Jabber-Client *Spark* ist ein Open-Source-Projekt des amerikanischen Softwareunternehmens *Jive Software* in der Programmiersprache Java. Die dazugehörige Community wird unter dem Namen „Ignite Realtime“ geführt und betreut unterschiedliche XMPP-basierte Projekte. Dazu gehört insbesondere auch der im Rahmen dieser Arbeit eingesetzte XMPP-Server *Openfire* und die XMPP-API *Smack*.
Die Lizenzierung von Spark erfolgt unter der *GNU Lesser General Public License* (LGPL) [Fre09]. Diese Lizenzform wurde von der *Free Software Foundation* entwickelt und soll den frei zur Verfügung gestellten Quellcode vor unfreier Vereinnahmung durch ein „Copyleft“ schützen [Geo09]. Grundsätzlich muss ein LGPL-lizenziertes Programm zusammen mit seinem Quellcode vertrieben werden und darf sowohl kommerziell wie auch privat verwendet werden.
Der Quellcode darf dabei beliebig verändert und verteilt werden, doch jede neue Version muss durch das Copyleft auch unter der LGPL lizenziert werden. Die externe Nutzung des Quellcodes, beispielsweise durch Verwendung von LGPL-lizenzierten Bibliotheken, hat jedoch keinen Einfluss auf die Lizenzierung.
Mit Hinblick auf die Entwicklung des Plugins SecureMUC bedeutet dies, dass die Bibliotheken von Spark und Smack verwendet werden können, ohne das lizenzrechtliche Einschränkungen für SecureMUC entstehen, da die Bibliotheken nur extern in Form dynamischer Verlinkungen verwendet werden. Würde SecureMUC hingegen als fester Bestandteil von Spark entwickelt, so müsste die Lizenzierung unter der LGPL erfolgen.

Die Implementierung aller Ignite Realtime-Projekte sind konform zum XMPP-Standard (RFC 3920) und sind damit unabhängig voneinander einsetzbar.

6.2 Aufbau

Grundsätzlich müssen bei der Betrachtung des Aufbaus von Spark zwei unterschiedliche Bibliotheken betrachten werden:

1. **Spark-API:** Enthält die grafische Oberfläche und die Logik sowie Verarbeitung von XMPP-Paketen
2. **Smack-API:** Enthält die Implementierung des XMPP-Protokolls

Die Smack-API ist dabei unabhängig von Spark nutzbar und gilt im Rahmen der Community als eigenständiges Projekt, welches unabhängig von Spark verwendet werden kann. Der Spark-Client selbst verwendet jedoch ausschließlich die Smack-API zur Implementierung des XMPP-Protokolls.
Das Konzept innerhalb der Spark-API ist die Aufteilung der API in Manager, Event-Handler, Event-Listener und Components [Ign09a]:

- **Manager:** Enthalten die Logik und ermöglichen den Zugang zum System.
- **Event-Listener:** Überwachen den Datenverkehr und die grafische Oberfläche um auf bestimmte Ereignisse reagieren zu können.
- **Event-Handler:** Führen bei Ereignissen bestimmte Funktionen aus.
- **Components:** Teile der grafischen Oberfläche.

Die Manager sind somit die übergeordneten Instanzen, die die anderen Komponenten Verwalten und deren Interaktion definieren. Um beispielsweise ein Plugin in Spark verfügbar zu machen, muss das Plugin vom *PluginManager* geladen werden. Möchte man hingegen vom Plugin Zugriff auf das System, so muss man sich die Instanz des entsprechenden Managers besorgen.

6.3 Erweiterbarkeit

Die Entwicklung von Erweiterungen in Form von Plugins ist eine wesentliche Funktion von Spark, der auch in der Community durch zahlreiche Beispiele und

Anleitungen besondere Beachtung geschenkt wird. Um ein Programm jedoch als Plugin in Spark verfügbar zu machen, müssen eine Reihe von Bedingungen erfüllt werden. Grundsätzlich müssen Plugins als JAR-Archive zur Verfügung gestellt werden, in denen sich ein Verzeichnis „libs“ und eine Datei „plugin.xml“ befinden muss. Das Verzeichnis „libs“ enthält die kompilierten class-Dateien und in der Datei „plugin.xml“ werden Angaben zum Plugin in XML-Form gegeben:

```
<plugin>
   <name>SecureMUC</name>
   <class>de.unikoblenz.fdietz.SecureMUC</class>
   <author>Florian Dietz</author>
   <version>1.0</version>
   <description>Implments [...] </description>
   <email>fdietz@uni-koblenz.de</email>
   <minSparkVersion>2.0.6</minSparkVersion>
 </plugin>
```

Die Informationen dieser Datei werden einerseits zur Installation des Plugins benötigt, da im <class>-Element die Angabe der Klasse gegeben wird, die der PluginManager zur Initialisierung des Plugins laden soll. Andererseits sind aber auch Informationen wie Autor und Funktionsbeschreibung enthalten, die lediglich Informationen für den Nutzer des Plugins darstellen.
Weiterhin existiert ein Plugin-Interface, welches von der Klasse implementiert werden muss, die im <class>-Element der Datei „Plugin.xml“ adressiert wurde:

```
public interface Plugin{

 public void initialize();

 public void shutdown();

 public boolean canShutdown();

 public void uninstall();

}
```

Wird ein Plugin vom PluginManager geladen, so wird die Methode initialize() aufgerufen. Im implementierenden Plugin muss demnach dort die Initialisierung des Plugins gestartet werden. Die Methode shutdown() wird bei jedem Plugin vor dem Beenden von Spark aufgerufen und dient der Persistenz der Daten. Befinden sich im Plugin beispielsweise noch nicht gespeicherte Daten, so können diese nun gespeichert werden, um den Datenverlust zu verhindern. Im Zusammenhang dazu steht die Methode canShutdown(), da ein Plugin anhand dieser Funktion signalisieren kann, ob es sich in einem persistenten Zustand befindet und damit bereit zum Beenden von Spark ist. Soll ein Plugin hingegen von Spark deinstalliert werden, so wird die Methode uninstall() aufgerufen. Jedes Plugin ist dabei selbst dafür verantwortlich, dass alle vom Plugin erzeugten Dateien und Spuren selbstständig entfernt werden.
Nur wenn die aufgeführten Bedingungen erfüllt werden, ist ein Plugin innerhalb von Spark lauffähig. Um die Erstellung von Plugins jedoch zu vereinfachen, wurde von Ignite Realtime ein *Sparkplug Kit* [Ign09b] veröffentlicht, in der bereits eine vordefinierte Struktur und Skripte zum kompilieren von Plugins enthalten sind.

Kapitel 7

Implementierung

7.1 Überblick

Zur Realisierung der gewünschten Funktionalität von SecureMUC werden verschiedene vorhandene Konzepte miteinander verbunden. Grundgedanke ist die Verwendung des Initiators als vertrauenswürdige Instanz, die die Verifikation aller Teilnehmer übernimmt und für die sichere Verteilung eines Session-Keys sorgt. Um dies zu erreichen baut der Initiator mit jedem Teilnehmer eine direkte Verbindung auf und führt ein Diffie-Hellma Key Agreement durch. Der ausgehandelte Schlüssel wird - ähnlich wie bei ZRTP (siehe Kapitel 4.4) - durch Vergleich des Hashwerts über eine Audio-/Video-Verbindung des Schlüssels verifiziert. Erst nach erfolgreicher Verifikation eines Clients erhält der entsprechende Client den Session-Key der Sitzung. Um den Ablauf des Plugins zu verdeutlichen und zu konkretisieren wird im Folgenden der Ablauf in vier verschiedene Phasen unterteilt und getrennt voneinander betrachtet.

7.1.0.1 Phase 1: Versand der Einladungen

Der Initiator einer Sitzung ist für den Versand der Einladungen verantwortlich und als einziger Teilnehmer dazu berechtigt. Dazu kann der Initiator Personen aus seiner Contact List auswählen und die Einladungen gesammelt versenden. Nach dem Versand der Einladungen öffnet sich die Initiatorenoberfläche, die für den Initiator sowohl für administrative Aufgaben wie auch für die Darstellung der eigentlichen Konferenz verwendet wird. Auf Client-Seite werden Einladun-

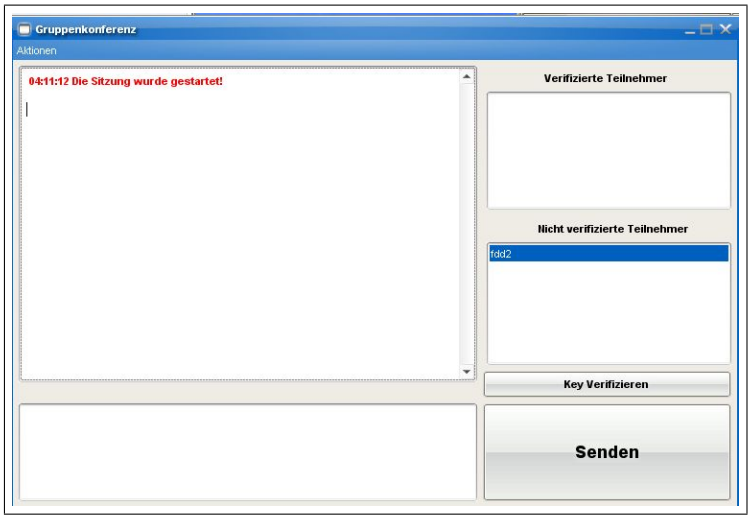

Abbildung 7.1: Aufbau der Initiatorenoberfläche *Quelle: Eigene Darstellung*

gen in Form von Popups dargestellt, die Informationen über den Initiator und das Thema der Konferenz beinhalten. Eine Einladung kann vom Client entweder angenommen werden, wodurch sich die Clientoberfläche von SecureMUC öffnet und automatisch mit dem Verbindungsaufbau und Key Agreement begonnen wird, oder abgelehnt werden. Auch wenn eine Einladung vorerst abgelehnt wird, werden Einladungen zu einer Sitzung im Client gespeichert und ermöglichen ein nachträgliches Betreten einer Sitzung.
Nach Abschluss von Phase 1 haben demnach alle vom Initiator ausgewählten Clients eine Einladung erhalten und der Initiator hat die Initiatorenoberfläche als aktuelle Ansicht.

7.1.0.2 Phase 2: Verbindungsaufbau und Key Agreement

Hat ein Client die Einladung angenommen, so wird automatisch der Verbindungsaufbau zum Initiator gestartet es öffnet sich auf Client-Seite die Clientoberfläche. Der Initiator überprüft bei Verbindungsaufbau eines neuen Clients zunächst ob der verbundene Client eine Einladung erhalten hat und damit berechtigt ist an der Sitzung teilzunehmen. Ist dies der Fall, so wird zwischen Client und Initiator ein Diffie-Hellman Key Agreement ausgeführt und der Hashwert des ausgehandelten Schlüssels berechnet. Erst wenn das Key Agreement erfolgreich abgeschlossen wurde wird dem Client die erfolgreiche Verbindung mitgeteilt und

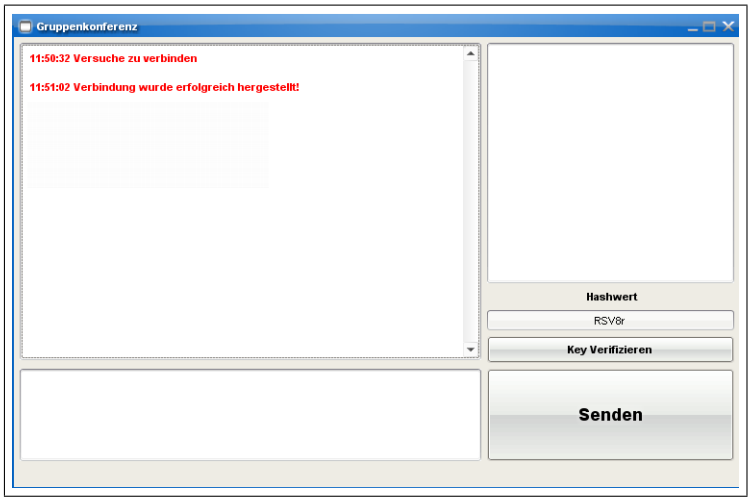

Abbildung 7.2: Aufbau der Clientoberfläche *Quelle: Eigene Darstellung*

dem Initiator die Verbindung zu einem nicht verifizierten Teilnehmer gemeldet. Alle Schritte dieser Phase werden automatisch und ohne manuelle Aktionen von Client oder Initiator ausgeführt. Nach Abschluss der Phase haben Initiator und Client ein individuelles Schlüsselpaar ausgehandelt und den Hashwert für die Verifikation des Schlüssels berechnet.

7.1.0.3 Phase 3: Verifikation

Abbildung 7.3: Aufbau der Verifikationsoberfläche *Quelle: Eigene Darstellung*

Die Verifikation des Diffie-Hellman-Schlüssels muss vom Initiator der Sitzung für jeweils einen einzelnen Client gestartet werden. Dazu wählt der Initiator einen noch nicht verifizierten Teilnehmer aus der gleichnamigen Liste aus und

kann die Verifikation starten. Daraufhin wird vom Initiator ein *Verification Request* an den Client gesendet wodurch sich bei Initiator und Client die Verifikationsmaske zusammen mit einer Audio-/Video-Verbindung öffnet, die über den Diffie-Hellman-Schlüssel verschlüsselt wird. Über die Audio-/Video-Verbindung lesen sich nun Initiator und Client den berechneten Hashwert gegenseitig vor. Stimmt der Hashwert bei Initiator und Client überein, müssen beide Teilnehmer die Korrektheit des Hashwerts in der Verifikationsmaske bestätigen. Von diesem Zeitpunkt an gilt der Client als verifiziert und erhält vom Initiator den Session-Key der Sitzung. Der Session-Key wird dabei mit dem verifizierten Diffie-Hellman-Schlüssel verschlüsselt übertragen.

7.1.0.4 Phase 4: Betrieb

Wurde ein Client verifiziert und hat den Session-Key erhalten, werden über eine *Member Join Message* alle anderen Clients informiert. Von diesem Zeitpunkt an kann der neu hinzugekommene Client mit allen anderen Teilnehmern verschlüsselt kommunizieren.

7.2 Spezifikation

7.2.1 Kryptographie

Die Auswahl der kryptografischen Algorithmen wurde stark durch die Verfügbarkeit innerhalb der Java Virtual Machine (JVM) beeinflusst. Aus exportrechtlichen Gründen der USA ist es dem Unternehmen SUN Microsystems nicht gestattet, die JVM standardmäßig mit unbeschränkten kryptografischen Algorithmen auszuliefern [SUN09]. Daher ist beispielsweise der AES-Algorithmus auf eine Schlüssellänge von 128-Bit begrenzt.
Um unbeschränkte kryptografische Algorithmen in der SUN-JVM nutzen zu können, wird der manuelle Download und die Installation der „Unlimited Strength Jurisdiction Policy Files“ benötigt, die auf der Homepage von SUN kostenlos angeboten werden. Da Manipulationen an der JVM jedoch für den Betrieb des Plugins ausgeschlossen werden, wurden die standardmäßigen enthaltenen Algorithmen der JVM verwendet:

- **Verschlüsselung:** Die Verschlüsselung erfolgt mit dem AES-Algorithmus

und einer Schlüssellänge von 128-Bit.

- **Signaturen:** Digitale Signaturen werden mit dem RSA-Algorithmus und einer Schlüssellänge von 1024-Bit erstellt.
- **Hashwerte:** Hashwerte werden mit dem SHA1-Algorithmus erstellt.

Sollten in Weiterentwicklungen stärkere Algorithmen benötigt werden, bietet sich beispielsweise eine eigene Implementation der kryptografischen Algorithmen an.

7.2.2 Verbindungsaufbau und Key Agreement

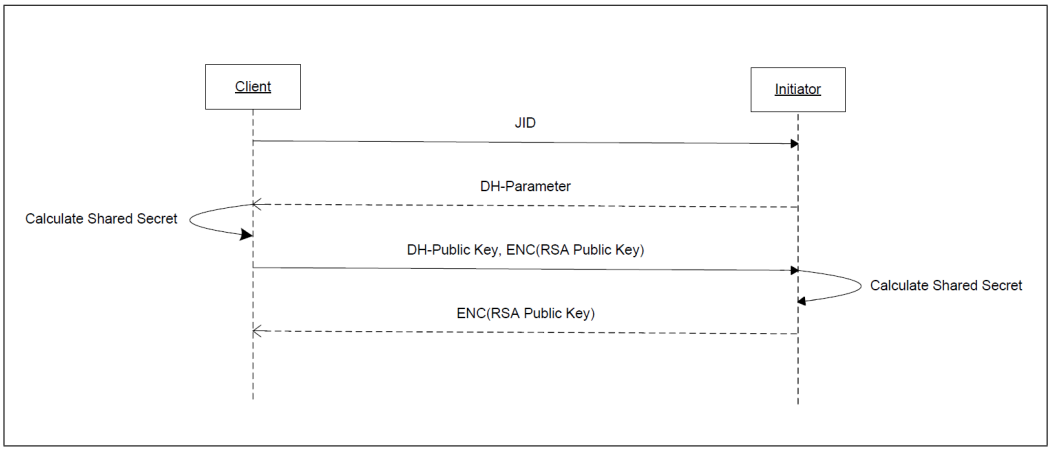

Abbildung 7.4: Verbindungsaufbau und Key Agreement *Quelle: Eigene Darstellung*

Der Verbindungsaufbau muss von einem Client ausgehen. Die dazu notwendigen Parameter kann der Client aus der erhaltenen Einladung zur SecureMUC-Sitzung entnehmen. Um den Verbindungsaufbau und das Key Agreement erfolgreich abzuschließen, sind folgende Schritte auf Seite des Clients und des Initiators notwendig (siehe Abbildung 7.4):

1. Der Client sendet dem Initiator seine JID. Anhand der JID des Clients überprüft der Initiator, ob der Client zu der Sitzung eingeladen wurde. Ist dies nicht der Fall, so wird der Verbindungsaufbau verweigert.
2. Ist der Client zu der Sitzung eingeladen worden, antwortet der Initiator mit den DH-Parametern und seinem DH-Public Key.

3. Der Client berechnet mit den empfangenen Daten seinen öffentlichen DH-Schlüssel DH_{pub} sowie den DH-Session-Key $DH_{session}$ und wählt ein neues RSA-Schlüsselpaar (RSA_{pub}, RSA_{priv}) aus. Anschließend verschlüsselt der Client seinen öffentlichen RSA-Schlüssel mit dem DH-Session-Key:

$$ENC(K_{DH}, RSA_{pub}) \tag{7.1}$$

Ist dies geschehen, sendet der Client den verschlüsselten RSA-Schlüssel und den DH-Public Key an den Initiator:

$$SEND(ENC(K_{DH}, RSA_{pub}), DH_{pub}) \tag{7.2}$$

Der Client ist nun im Besitz des vorläufigen Session-Keys, der zur Verifikation benötigt wird.

4. Der Initiator verwendet den empfangenen DH-Public Key des Clients um den DH-Session-Key zu berechnen. Ist dies geschehen, wird mit dem DH-Session-Key der empfangene RSA-Public Key des Clients entschlüsselt.
Um dem Client den erfolgreichen Abschluss des Key Agreement mitzuteilen, verschlüsselt der Initiator seinen öffentlichen RSA-Schlüssel mit dem DH-Session Key versendet ihn an den Client.

5. Der Client entschlüsselt den empfangenen RSA-Public Key des Initiators mit dem DH-Session-Key.

Nach Abschluss dieser Phase haben Initiator und Client einen gemeinsamen DH-Session-Key vereinbart und die RSA-Public Keys ausgetauscht, die zur späteren Verifikation von Signaturen benötigt wird.

7.2.3 Verifikation

Die Verifikation eines Clients kann nur vom Initiator manuell gestartet werden und läuft in folgenden Schritten ab (siehe Abbildung 7.5):

1. Der Initiator sendet ein *Verification Request* an den Client und es öffnet sich die Verifikationsmaske.

2. Der Client reagiert auf ein eintreffendes Verification Request mit der dem Aufbau der Audio-/Video-Verbindung und dem öffnen der Verifikations-

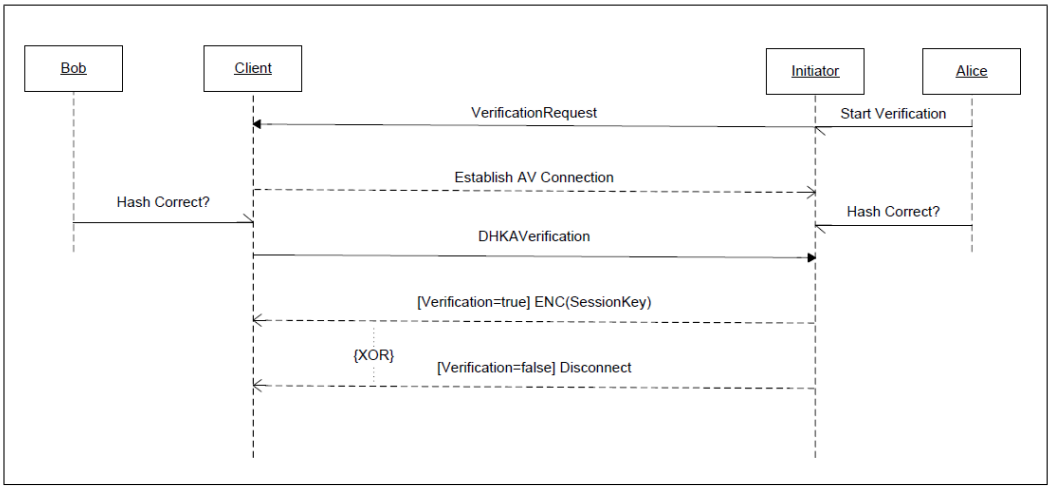

Abbildung 7.5: Ablauf der Verifikation *Quelle: Eigene Darstellung*

maske. Der Aufbau der Audio-/Video-Verbindung geschieht durch die Auslösung eines Events an Schnittstellen und ist nicht Bestandteil von SecureMUC.

3. Über die aufgebaute Audio-/Video-Verbindung vergleichen Initiator und Client den Hashwert des DH-Session-Key. Der Hashwert wird als String aus fünf Zeichen dargestellt und kann Buchstaben in Groß- und Kleinschreibung sowie Zahlen enthalten. Der Initiator der Sitzung ist in dieser Phase zusätzlich dafür verantwortlich, die Identität des Clients überprüfen.

4. Ist der Hashwert korrekt, so müssen Client und Initiator die Korrektheit in der Verifikationsmaske bestätigen. Ist dies geschehen, schließt sich die Verifikationsmaske und die Audio-/Video-Verbindung wird durch das Auslösen eines entsprechenden Events beendet.

5. Der Client sendet bei erfolgreicher Verifikation eine DHKA-Verification an den Initiator, die mit dem DH-Session-Key verschlüsselt ist.

6. Der Initiator versendet bei erfolgreicher Verifikation seinerseits und der Ankunft der DHKA-Verification des Clients den Session-Key der Sitzung an den Client. Der Session-Key wird zur Übertragung mit dem verifizierten DH-Session-Key verschlüsselt übertragen. Abschließend sendet der Initia-

tor eine Nachricht an alle bereits verifizierten Teilnehmer über die erfolgreiche Verifikation des gerade verifizierten Clients.

Nach Abschluss dieser Phase haben demnach Initiator und Client die Verifikation abgeschlossen und der Client hat den Session-Key der Sitzung vom Initiator erhalten. Von diesem Zeitpunkt an gilt der Client als verifizierter Teilnehmer und kann aktiv an der Konferenz teilnehmen.

7.2.4 Nachrichtenformat

Der Austausch von Daten wird in SecureMUC über das Jabber-Protokoll realisiert. Dabei werden prinzipiell das *Message Protocol* und das *IQ-Protocol* eingesetzt. Da die benötigten Funktionen im Jabber-Protokoll jedoch nicht vorgesehen sind, mussten zu diesem Zweck Protokollerweiterungen vorgenommen werden. Im Falle des IQ-Protocol stelle sich dies als wenig problematisch heraus, da Protokollerweiterungen expliziter Bestandteil des IQ-Protocol sind. Daher werden alle Daten, die nicht Bestandteil des Chats sind, in SecureMUC über das IQ-Protocol versendet.
Der Versand von Nachrichten des Chats hingegen konnte nicht über die vorhandenen Möglichkeiten des Message Protocol realisiert werden beziehungsweise hätte nicht XMPP-konforme Techniken erfordert. Daher wurde für den Versand von Nachrichten der *Message Type* „eGroupchat“ erstellt:

```
<Message From="alice@..." To="bob@..." Type="eGroupchat">
  <TreadID>73f646DH</TreadID>
  <EncryptedData>
    <CipherData>
      <CipherValue>BASE64EncodedData</CipherValue>
    </CipherData>
  </EncryptedData>
  <Signature>
    <SignatureMethod Algorithm="RSA" />
    <DigestMethod Algorithm="SHA1" />
    <SignatureValue>BASE64EncodedData</SignatureValue>
  </Signature>
</Message>
```

Der Aufbau des <Message>-Elements ist an den regulären Aufbau innerhalb des Message Protocol angelehnt. Unterschiede existieren lediglich durch die Elemente <EncryptedData> und <Signature>. Im <EncryptedData>-Element werden die verschlüsselten Nachrichten des Chats transportiert und im <Signature>-Element sowohl der Wert der Signatur selbst wie auch Angaben zum verwendeten Signaturverfahren getroffen.
Auch wenn die Erstellung des Message Types „eGroupchat“ nicht XMPP-konform ist, so wurde diese Lösung dennoch als „näher“ am XMPP-Standard angesehen als etwaige Alternativen, wie etwa eine Abwicklung des Chat-Verkehrs über das IQ-Protocol.

7.3 Realisierung in Java

Ausgangspunkt der Realisierung von SecureMUC in Java ist Klasse *SMUC*. In dieser Klasse wird das vom Spark-Pluginmanager benötigte Plugin-Interface implementiert und damit die Intialisierung des Plugins gestartet. Übergeordnete Anwendungen, die die SecureMUC-Funktionalität importieren möchten, müssen daher eine Instanz der Klasse *SMUC* erstellen und die enthaltene initialize()-Methode ausführen. Die Klasse SMUC dient innerhalb des Plugins als zentrale Einheit, die die anderen Komponenten verwaltet und deren Funktion überwacht. Die visuelle Darstellung des Plugins geschieht durch Instanzen der Klasse *JFrame* beziehungsweise *JDialog*, die im Paket *Views* zusammengefasst werden. Bei der Darstellung des Konferenzfensters wird zwischen Initiator und Client unterschieden, da auf der Initiatorenoberfläche auch administrative Komponenten untergebracht werden müssen. Auch die Verbindungsklassen unterscheiden sich bezüglich der Implementation zwischen Client und Initiator:

- **ClientConnectionManager:** Enthält Komponenten zum Verbindungsaufbau zu (genau) einem Initiator.
- **ServerConnectionManager:** Enthält Komponenten zur Annahme und Verwaltung mehrerer Clients zur gleichen Zeit. Ermöglicht wird dies durch den Einsatz von Multithread-Programmierung.

Die Klassen *ClientConnectionManager* und *ServerConnectionManager* sind demnach im Betrieb direkt miteinander verbunden, tauschen *XML-Files* miteinander aus

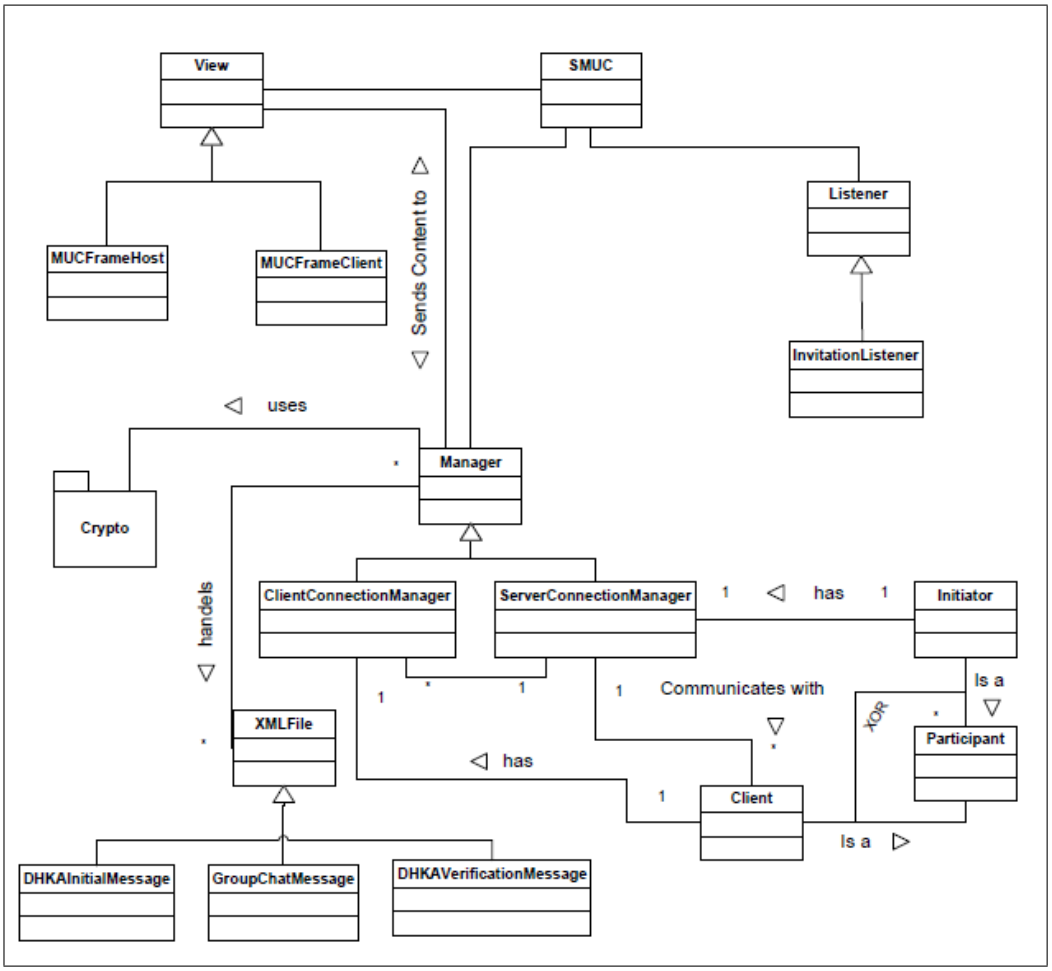

Abbildung 7.6: Statische Struktur der Implementation in Java *Quelle: Eigene Darstellung*

und leiten den Inhalt an die entsprechende visuelle Klasse zur Darstellung weiter. Da Nutzer nur mit den Views des Plugins interagieren, beziehen die Verbindungsklassen von diesen jedoch auch den zu sendenden Inhalt (siehe Abbildung 7.6). Die Oberklasse *XMLFile* wird für die Realisierung aller Jabber-Nachrichten verwendet und garantiert das Vorhandensein der XMPP-Steuerungsinformationen und die Möglichkeit der Serialisierung.
Weitere Besonderheit ist die Klasse *Participant*, die als temporärer Speicher für nutzerbezogene Informationen verwendet wird. Jeder Nutzer innerhalb des Plugins wird als eine Instanz der Klasse Participant repräsentiert. Darin sind beispielsweise Variablen und Methoden zur Speicherung und Abfrage von JID, Public Key

oder auch Socket(-Identifier) enthalten.
Die Realisierung von Schnittstellen für die Audio-/Video-Verbindung wurde in Form einer Registrierung von Komponenten und die Benachrichtigung über Events realisiert. Ein VoIP-Client kann sich beispielsweise in der Klasse *SMUC* registrieren und wird daraufhin über alle relevaten Events des Plugins informiert.

7.4 Analyse

7.4.1 Performance

Neben menschlichen Faktoren, wie etwa der unbekannte Ablauf des Plugins, entstehen aus der gewählten Implementation Limitierungen, die sich hinsichtlich der Gesamtperformance negativ auswirken. Der Initiator einer Sitzung stellt als zentrale Instanz, die als Server fungiert und den Ablauf des Plugins koordiniert, den Flaschenhals der Implementation dar. Bei genauerer Betrachtung kristallisierien sich insbesondere die folgenden Faktoren heraus:

- **Verbindungsvorbereitung:** Die Vorbereitung der Verbindung findet zwar nur einmalig zu Beginn des Ablaufs statt, ist durch das Starten der verschiedenen Server-Threads und die Berechnung von Verbindungsparametern sehr zeitintensiv. Da der Verbindungsaufbau von Clients jedoch möglichst verzögerungsfrei beginnen soll, ist die Verbindungsvorbereitung zu Beginn der Sitzung für den Initiator unerlässlich.

- **Key Agreement:** Da jeder Client auf Seite des Initiators innerhalb eines eigenen Threads bearbeitet wird, geschieht auch das Key Agreement innerhalb dieses Threads. Abhängig von der Ressourcenzuteilung des Schedulers kann es geschehen, dass die Berechnungen des Key Agreements über mehrere Zyklen verteilt werden und damit den Verbindungsaufbau eines Clients verzögern.

- **Verifikation:** Der Initiator muss die Verifikation für jeden Teilnehmer einzeln durchführen und ist nicht in der Lage, mehrere Clients gleichzeitig zu verifizieren. Verbinden sich zeitnah mehrere Clients, entstehen demnach zwangsläufig Wartezeiten bis die Verifikation vom Initiator gestartet wird. Die Verifikation selbst ist abhängig von einer funktionierenden

Audio-/Video-Verbindung. Entstehen an dieser Stelle Verzögerungen oder Fehler, so verzögert sich beziehungsweise scheitert auch die Verifikation.

Engpässe entstehen somit größtenteils in den Phasen des Verbindungsaufbaus und der Verifikation. Geht man von einer kleineren Anzahl von Teilnehmern aus, sollten im weiteren Betrieb des Plugins jedoch keine nennenswerten Verzögerungen auftreten.

7.4.2 Sicherheit

Die Erreichung einer sicheren Gruppenkommunikation war das wichtigste Designziel bei der Entwicklung von SecureMUC. Daher sollte auch der Aspekt der Sicherheit des Plugins erneut mit den Erkenntnissen der Implementation betrachtet werden.
Durch die Integration eines Authentifikationsmechnismus in das Diffie-Hellman Key Agreement in Kombination mit digitalen Signaturen und Nachrichtenverschlüsselung wird ein Schutz vor einem breiten Spektrum potentieller Angriffe abgedeckt. Dennoch sind weiterhin Sicherheitsrisiken enthalten, die aus Fehlverhalten von Teilnehmern resultieren können. Denkbar wäre beispielsweise die unerlaubte Weitergabe des Session-Keys durch einen verifizierten Teilnehmer. Weiterhin ist die Sicherheit der gesamten Gruppenkommunikation von der verantwortungsvollen Ausübung der Rolle des Initiators durch einen Teilnehmer abhängig. Wird die Aufgabe der Verifikation nicht gewissenhaft ausgeführt, greifen die im Laufe der Arbeit beschriebenen Maßnahmen nicht und werden damit wirkungslos.
Bei korrekter Handhabung können die Sicherheitsanforderungen Integrität, Authentizität, Verfügbarkeit und Vertraulichkeit jedoch als ausreichend erfüllt betrachtet werden.

Kapitel 8

Fazit

Die vorangegangenen Ausführungen haben sich mit IM-Systemen allgemein und im speziellen mit dem Einsatz von IM-Systemen in sicherheitskritischen Szenarien beschäftigt. Deutlich sollte geworden sein, das IM heutzutage ein großes Potential zur Unterstützung unterschiedlichster Szenarien bietet. Dabei werden von den heute am Markt verfügbaren Systemen aber noch längst nicht alle Anwendungsgebiete in ausreichendem Maße bedient, wodurch Erweiterbarkeit und Standardkonformität der eingesetzten Systeme eine hohe Relevanz erhalten. Insbesondere proprietäre Systeme lassen diese Eigenschaften jedoch schmerzlich vermissen und schließen sich damit selbst für viele denkbare Szenarien aus. Das im Rahmen dieser Arbeit eingesetzte System bestehend aus dem Jabber-Server Openfire und dem Jabber-Client Spark hat sich im Gegensatz dazu als geeignete Ausgangsbasis für sicherheitskritische Anwendungen erwiesen. Defizite, wie etwa die Begrenzung der Schlüssellängen der eingesetzten kryptographischen Algorithmen ergeben sich zumeist aus Limitierungen der zugrundeliegenden Java Virtual Machine (JVM) und können in der gezeigten Art und Weise für einen Produktivbetrieb umgangen werden. Daher lässt sich für das verwendete System ein positiver Gesamteindruck festhalten, der maßgeblich durch die Erweiterbarkeit und den frei zugänglichen Quellcode sämtlicher Komponenten geprägt ist.
Das entstandene Plugin SecureMUC erfüllt die Anforderungen an spontane aber sichere Gruppenkommunikation, ist jedoch stark von der Rolle des Initiators abhängig und damit nur für kleinere Personengruppen geeignet.

Literaturverzeichnis

[Ana05] IDC Market Analysis. Worldwide Enterprise Instant Messaging Applications 2005-2009 Forecast and 2004 Vendor Shares: Clearing the Decks for Substantial Growth. 2005.

[Buc03] Johannes Buchmann. *Einführung in die Kryptographie, 3. Auflage*. Springer Verlag, Heidelberg, 2003.

[Bun09] Bundesamt für Sicherheit in der Informationstechnik. Voipsec - studie zur sicherheit von voice over internet protocol, 2005, http://www.mussichnochfinden.com, Zugriffsdatum : 05.01.2009.

[Egg04] Claudia Eggert. *IT-Sicherheit: Konzepte - Verfahren - Protokolle, 3. Auflage*. Oldenbourg Wissenschaftverlag, München, 2004.

[Fre09] Free Software Foundation. GNU LESSER GENERAL PUBLIC LICENSE Version 3, 2007, http://www.gnu.org/licenses/lgpl-3.0.txt, Zugriffsdatum : 22.03.2009.

[Geo09] Georg Jakob. Freie Softwarelizenzen - Rechtliche Aspekte der GPL, LGPL und BSDL, 2003, http://www.users.sbg.ac.at/~jack/legal/wu_fsl/fs_lizenzen.pdf, Zugriffsdatum : 22.03.2009.

[Hei09] Heise Online. News: Microsoft und Yahoo wollen bei Instant Messaging zusammenarbeiten, 2005, http://www.heise.de/newsticker/Microsoft-und-Yahoo-wollen-bei-Instant-Messaging-zusammenarbeiten-U
meldung/64822, Zugriffsdatum : 22.01.2009.

[Ign09a] Ignite Realtime. Sparkplug Development Guide, 2009, http://www.igniterealtime.org/builds/sparkplug_

kit/docs/latest/sparkplug_dev_guide.html, Zugriffsdatum : 22.03.2009.

[Ign09b] Ignite Realtime. Sparkplug Kit 207, 2009, http://www.igniterealtime.org/projects/spark/sparkplug-kit.jsp, Zugriffsdatum : 22.03.2009.

[Int09a] Internet Engineering Task Force. RFC 1760: The S/KEY One-Time Password System, 1995, http://tools.ietf.org/html/rfc1760, Zugriffsdatum : 05.01.2009.

[Int09b] Internet Engineering Task Force. The Internet Standards Process - Revision 3, 1996, ftp://ftp.rfc-editor.org/in-notes/bcp/bcp9.txt, Zugriffsdatum : 03.01.2009.

[Int09c] Internet Engineering Task Force. RFC 2195: IMAP/POP AUTHorize Extension for Simple Challenge/Response, 1997, http://tools.ietf.org/html/rfc2195, Zugriffsdatum : 05.01.2009.

[Int09d] Internet Engineering Task Force. RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core, 2004, http://www.ietf.org/rfc/rfc3920.txt, Zugriffsdatum : 05.01.2009.

[Int09e] Internet Engineering Task Force. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, 2004, http://www.ietf.org/rfc/rfc3921.txt, Zugriffsdatum : 09.01.2009.

[Int09f] Internet Engineering Task Force. Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM), 2004, http://www.ietf.org/rfc/rfc3922.txt, Zugriffsdatum : 09.01.2009.

[Int09g] Internet Engineering Task Force. End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP), 2004, http://www.ietf.org/rfc/rfc3923.txt, Zugriffsdatum : 09.01.2009.

[Int09h] Internet Engineering Task Force. RFC 4422: Simple Authentication and Security Layer (SASL), 2006, http://tools.ietf.org/html/rfc4422, Zugriffsdatum : 17.01.2009.

[Int09i] Internet Engineering Task Force. The Transport Layer Security (TLS) Protocol Version 1.2, 2008, http://tools.ietf.org/rfcmarkup/5246, Zugriffsdatum : 03.01.2009.

[Jab09a] Jabber Software Foundation. JEP-0078: Non-SASL Authentication, 2004, http://xmpp.org/extensions/attic/jep-0078-1.8.html, Zugriffsdatum : 17.01.2009.

[Jab09b] Jabber Software Foundation. Jabber philosophy overview, 2009, http://www.jabber.org/web/Philosophy_overview, Zugriffsdatum : 05.01.2009.

[Joe08] Joe Hildebrand. Nine IM accounts and counting. http://www.acm.org/literat/bsi_standard/, letzter Aufruf: 15.02.2009, 2008.

[MDH01] Pierre Paradinas(Hrsg.) Michel Dupuy (Hrsg.). *Trusted Information: The New Decade Challenge*. Springer Verlag, Heidelberg, 2001.

[Mel09] Anastasia Meletiadou. Instant messaging systeme als plattform für elektronisches wählen. In Patrik Horster, editor, *Wird erscheinen in Proceedings of D.A.CH Security 2009*. syssec, 2009.

[Mic09] Michael Steiner, Gene Tsudik, Michael Waidner. CLIQUES: A New Approach to Group Key Agreements, 1998, http://www.isi.edu/div7/publication_files/cliques_a_new.pdf, Zugriffsdatum : 12.03.2009.

[Phi09] Phil Zimmermann. ZRTP: Media Path Key Agreement for Secure RTP, 2009, http://tools.ietf.org/html/draft-zimmermann-avt-zrtp-15, Zugriffsdatum : 01.04.2009.

[RSS09] RSS Advisory Board. RSS 2.0 Specification, 2009, http://www.rssboard.org/rss-specification, Zugriffsdatum : 15.01.2009.

[Rup07] Chris Rupp. *Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis, 4. Auflage*. Hanser Fachbuchverlag, München, 2007.

[Sch05] Bruce Schneier. *Angewandte Kryptographie - Protokolle, Algorithmen und Sourcecode in C, 1. Auflage*. Pearson Education, München, 2005.

[Shi02] Iain Shigeoka. *Instant Messaging with java - The Jabber Protocols, 1. Auflage*. Manning Publications, Greenwich, 2002.

[Sie02] Johannes Siedersleben. *Softwaretechnik: Praxiswissen für Softwareingenieure, 2. Auflage*. Hanser Fachbuchverlag, München, 2002.

[SUN09] SUN Microsystems. Readme: Unlimited Strength Java(TM) Cryptography Extension Policy Files for the Java(TM) Platform, Standard Edition Development Kit v6, 2009, https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=jce_policy-6-oth-JPR@CDS-CDS_Developer, Zugriffsdatum : 15.01.2009.

[TG07] Michael Herczeg (Hrsg.) Tom Gross, Michael Koch. *Computer Supported Cooperative Work, 1. Auflage*. Oldenbourg Wissenschaftsverlag, München, 2007.

[XMP09a] XMPP Standards Foundation. XEP-0045: Multi-User Chat, 2008, http://xmpp.org/extensions/xep-0045.html#connections, Zugriffsdatum : 05.01.2009.

[XMP09b] XMPP Standards Foundation. XEP-0200: Stanza Encryption, 2008, http://xmpp.org/extensions/xep-0246.html#encryption, Zugriffsdatum : 05.01.2009.

[XMP09c] XMPP Standards Foundation. XEP-0246: End-to-End XML Streams, 2008, http://xmpp.org/extensions/xep-0246.html#encryption, Zugriffsdatum : 05.01.2009.

[XMP09d] XMPP Standards Foundation. About the xmpp standards foundation, 2009, http://xmpp.org/xsf/, Zugriffsdatum : 05.01.2009.

[Zfo09] Zfone Project. The Zfone Project, 2009, http://zfoneproject.com/index.html, Zugriffsdatum : 01.04.2009.