UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Konzeption und Realisierung einer sicheren Audio-/Video-gruppenkommunikation

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science
im Studiengang Informatik

vorgelegt von

Matthias Boll

Erstgutachter: Prof. Dr. Rüdiger Grimm
Institut für Wirtschafts- und Verwaltungsinformatik,
Fachbereich Informatik,

Zweitgutachter: Anastasia Meletiadou
Institut für Wirtschafts- und Verwaltungsinformatik,
Fachbereich Informatik,

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)

Zusammenfassung

Diese Bachelorarbeit behandelt die Erweiterung eines Instant Messenger um die Funktion einer verschlüsselten Audio-/Videokonferenz. Es werden verschiedene zu diesem Zweck nutzbare Techniken beschrieben und gegenübergestellt. Besonderer Wert wird darauf gelegt, die Verbindung dementsprechend zu sichern, dass auch vertrauliche Informationen übermittelt werden können. Letztlich wird die Implementierung in Java für den Instant Messenger „Spark“ sowie die Integration der Konferenz in ein eVoting-Plugin erläutert.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Einleitung und Motivation

Viele Firmen und Behörden besitzen heutzutage mehrere Standorte oder Filialen. Um eine gute Abstimmung zwischen allen Standorten zu erreichen, sind regelmäßige Konferenzen notwendig. Doch sämtliche Teilnehmer an einem Ort zu versammeln ist oft mit einem hohen finanziellen und logistischen Aufwand verbunden. Daher benutzen viele Firmen mittlerweile zum Beispiel Telefonkonferenzen, um diesen Aufwand zu sparen. Doch diese haben den Nachteil, dass man die Gesprächspartner nicht zu Gesicht bekommt.
Geeigneter, um die Situation und Effektivität einer realen Konferenz zu ersetzen, ist eine Videokonferenz.
Manche Firmen nutzen zur internen Kommunikation bereits Instant Messenger. Ziel dieser Bachelorarbeit ist, einen existierenden Instant Messenger um die Funktion einer mit geringem Aufwand aufzubauenden Videogruppenkommunikation in kleineren Gruppen zu erweitern.

1.2 Fragestellungen

Bevor mit der Entwicklung eines solchen Systems begonnen werden kann, müssen verschiedene Fragen geklärt werden. Es gibt organisatorische, wie zum Beispiel: „Wie können die Teilnehmer über einen Konferenztermin benachrichtigt werden?“.

Andere Probleme sind technischer Natur: „Über welche Protokolle kommuniziert die Videoverbindung?“, „Wie werden die bei Videos entstehenden Datenmengen übertragen?“, „Mit welchen Codecs können diese Daten komprimiert werden?“. Außerdem muss auch die Sicherheit eine große Rolle spielen: „Kann die Konferenz beispielsweise vor Lauschangriffen geschützt werden?“, „Ist eine Verschlüsselung sinnvoll, und wenn ja welche Art von Verschlüsselung?“, „Bestehen Angriffsmöglichkeiten, um eine Videokommunikation zu sabotieren?“.
Diese Bereiche sollen in der Arbeit angesprochen werden. Größte Beachtung findet allerdings die technische Umsetzung und wie diese den aktuellen Anforderungen der IT-Sicherheit gerecht werden kann. Es müssen Protokolle ausgewählt und vereinbart werden, um die Kommunikation möglich zu machen. Die Ordnungsstruktur einer echten Konferenz muss auf die virtuelle Konferenz abgebildet werden. Dabei sollen Sicherheitskonzepte, die zur Zeit und auch noch in der nahen Zukunft als zuverlässig gelten, unterstützt werden.

Kapitel 2

Konzept

2.1 Beispielszenario

Ein mögliches Szenario für eine Audio-/Videokonferenz ist folgende, in Abbildung 2.1 graphisch dargestellte, Situation:
Herr K. ist Vorstandsvorsitzender eines großen internationalen Konzerns. Die Mitglieder des Vorstandes sind über den ganzen Erdball verteilt: Herr A. sitzt in Peking, Frau B. in London und Herr C. macht gerade Urlaub auf den Malediven. Nur Herr D. arbeitet momentan wie Herr K. vom Firmensitz in Koblenz aus. Der Konzern nutzt zur unkomplizierten internen Kommunikation ein Instant Messaging System mit integrierter Videokonferenzfunktion.
Der 5-köpfige Vorstand hat ein Meeting um eine mögliche Übernahme der XYZ-Gruppe zu beratschlagen. Daher wählt Herr K. die Vorstände A.,B.,C. und D. in der Kontaktliste seines Instant Messengers aus und lädt diese zu einer Videokonferenz ein. Nachdem er jeden Kontakt persönlich verifiziert hat, wird die verschlüsselte Konferenz aufgebaut. Die Vorstandsmitglieder besprechen die Vor- und Nachteile, die die Übernahme für den Konzern hätte. Bei der Durchsicht der Marktanalyse wundern sie sich über eine ungewöhnlich hohe Prozentzahl. Um sicher zu gehen, dass diese korrekt ist, wird Frau M., die Leiterin des Projektes, welches die Situation der XYZ-Gruppe genauestens analysiert hat, ebenfalls kurzfristig in die Konferenz eingeladen. Sie bestätigt die Zahlen und versichert, dass alles mehrfach überprüft wurde. Danach verlässt sie die Konferenz wieder. Nachdem der Vorstand seine Diskussion beendet hat, stimmen die Mitglieder über das weitere Vorgehen bezüglich XYZ ab. Herr K. wünscht allen noch einen

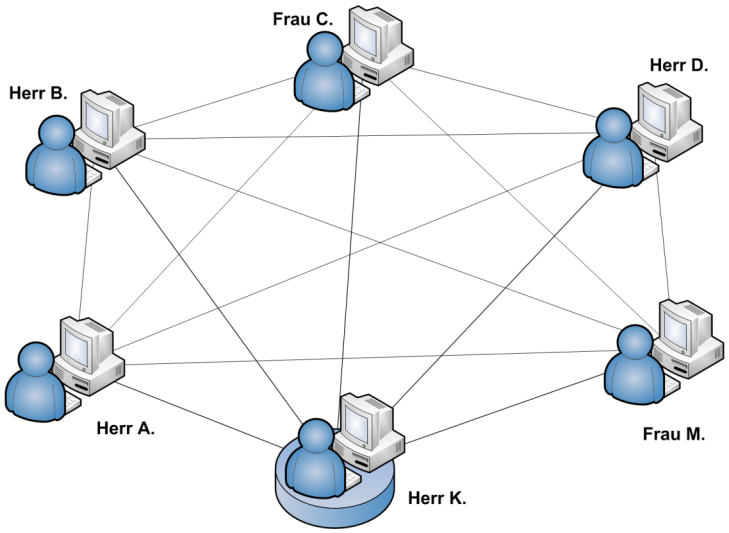

Abbildung 2.1: Ein mögliches Szenario

schönen Tag bzw. eine gute Nacht und schließt die Sitzung.

2.2 Zielsetzung

Die Videokonferenzanwendung muss den Austausch von Videodaten zwischen zwei oder mehr Kontakten ermöglichen. Ein Teilnehmer soll die Aufgaben eines Leiters haben. Er kann die Konferenz eröffnen und hat die Möglichkeit, neue Kontakte zu einer bestehenden Konferenz hinzuzufügen. Die Kommunikation soll zeitgemäß gesichert sein. Das heißt, dass die Integrität und Vertraulichkeit der Audio-/Videoübertragung, die Authentizität der Kontakte und die Verfügbarkeit des Videokonferenzsystems durch die Anwendung geschützt wird[1].
Skalierbarkeit und optimale Performance für Gruppenkommunikation mit einer größeren Zahl Teilnehmer kann vernachlässigt werden.
Des Weiteren soll die Videokonferenz als Teil eines elektronischen Abstimmverfahrens für Instant Messenger verwendet werden können[2]. Dafür wurde bereits ein Multiuser-Chat entwickelt, der die zur Verschlüsselung benutzbaren Schlüssel aushandelt und die Videoverbindung initiiert[3]. Zur Integration müssen ent-

[1] detaillierte Informationen hierzu finden sich in Kapitel 5
[2] weitere Information hierzu in [Mel08]
[3] SecureMUC, siehe dazu [Die09]

sprechend Schnittstellen vereinbart und implementiert werden.

2.3 Verteilung der Audio und Videodaten

Es gibt mehrere Varianten, wie die Daten an die verschiedenen Teilnehmer übermittelt werden könnten. Um eine auszuwählen werden verschiedene Netzwerktopologien[4] miteinander verglichen.
Zum einen gibt es die *Stern*-Variante, dann die *Peer-to-Peer*-Variante, die *lineare* Variante und dann noch die Möglichkeit einer *Tree*-Topologie. In den folgenden Abschnitten bezeichnet *n* immer die Anzahl der Teilnehmer im Netzwerk. Mit *Verbindung* wird der Empfang oder das Senden eines Audio-/Videostroms bezeichnet. Wenn also beispielsweise von A nach B zwei Audio-/Videoströme gesendet werden und A von B einen Audio-/Videostrom empfängt, hat A drei Verbindungen.

2.3.1 Peer-to-Peer

Bei der Nutzung von Peer-to-Peer-Verbindungen sieht die Netzwerkstruktur der Konferenzteilnehmer aus wie in Abbildung 2.1. Peer-to-Peer (P2P) ist eine Spezialversion der *Mesh*-Topologie. In einem Mesh ist jedes Netzwerkmitglied mit einem oder mehr Netzwerkmitgliedern verbunden. Bei P2P bestehen Verbindungen von jedem Netzwerkmitglied (=Peer) zu *allen* anderen Peers. Dies bedeutet, dass bei *n* Teilnehmern jeder Teilnehmer *n-1* sendende Verbindungen für seine Videodaten hat. Gleichzeitig sind auch *n-1* eingehende Verbindungen vorhanden, über die die Daten der anderen Teilnehmer empfangen werden. Jeder Peer hat also die gleiche Anzahl Verbindungen. Geht die Verbindung zu einem Peer verloren, bestehen die Verbindungen der restlichen Peers zueinander weiterhin.

2.3.2 Tree

In einer Treestruktur ist das Netzwerk hierarchisch geordnet. Ein Netzknoten ist der oberste Knoten, mit dem ein oder mehrere Unterknoten verbunden sind. Die-

[4] siehe [Mit08] und [lea08]

se Unterknoten sind wiederum mit ihren Unterknoten verbunden. Im Gegensatz zu einem Mesh sind die Netzwerkteilnehmer aber nur mit ihrem Elternknoten und ihren Unterknoten verbunden, nicht noch weiter untereinander. Wie viele sendende Verbindungen ein Teilnehmer hierbei zu verwalten hätte, hängt von der Anzahl seiner Unterknoten ab und wie tief sich der Knoten in der Baumstruktur befindet. Empfangen werden müssen immer n-1 Kanäle, pro anderes Konferenzmitglied einer. Bei einer Tree-Topologie wie in Abbildung 2.2 hat zum Beispiel Knoten *B* neun sendende Verbindungen und n-1=4 empfangende Verbindungen. Neun ausgehende Übertragungen deshalb, weil die Signale von A,E und das eigene jeweils an C und an D gesendet werden müssen (2*3 Verbindungen); außerdem ist das eigene Signal und die Weiterleitungen von C und D an A zu übermitteln. Knoten E muss dagegen nur einen Datenstrom senden (den eigenen). In einer Tree-Topologie herrscht also eine ungleiche Verteilung der Übertragungslast. Wenn ein Teilnehmer ausfällt, verlieren alle seine Unterknoten die Verbindung zum restlichen Tree, die Verbindung mit ihren eigenen Unterknoten besteht allerdings noch.

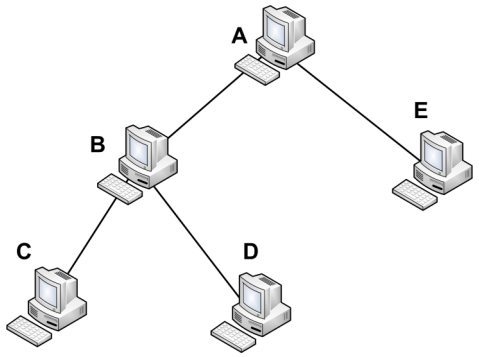

Abbildung 2.2: Tree-Netzwerktopologie

2.3.3 Linear

Die lineare Variante (siehe Abbildung 2.3) soll nur kurz angesprochen werden. Hierbei sind alle Teilnehmer nacheinander in einer Reihe miteinander verbunden. Empfangen müssen alle Knoten wieder *n-1* Kanäle. Zu Senden sind bei allen „innen“- liegenden Netzwerkmitgliedern *n-2* Kanäle (die der direkten Nachbarn fallen weg). Netzwerkmitglieder ganz außen müssen nur einen Kanal senden.

Die Last ist also (mit Ausnahme der beiden am Rand befindlichen Teilnehmer) gleichmäßig verteilt. Bei Ausfällen wird das Netzwerk an der Ausfallstelle gespalten.

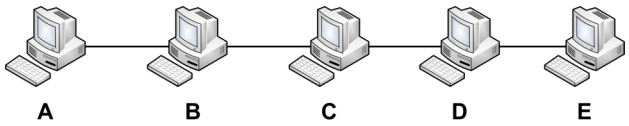

Abbildung 2.3: lineare Netzwerktopologie

2.3.4 Stern

Bei einem Sternaufbau mit zentralem Server übernimmt einer der Netzwerkteilnehmer den größten Teil der Arbeit. Die sternförmige Struktur ist in Abbildung 2.4 mit vier Teilnehmern dargestellt. Der *Server* (Mitte der Abbildung) stellt die einzige Verbindung der anderen Netzwerkknoten untereinander dar. Die Knoten müssen nur ihre eigenen Daten an den Server senden (eine ausgehende Verbindung) und dieser leitet an jeden Teilnehmer die Daten der anderen Kontakte weiter. Daher benötigt der als Server fungierende Teilnehmer *(n-1)*(n-1)* ausgehende Verbindungen. Der Vergleich dieser Zahlen verdeutlicht, dass der Server den bei Weitem größten Teil der Übertragungsleistung verrichtet. Wenn der Server ausfällt, würde die gesamte Konferenz abgebrochen.

2.3.5 Auswahl einer Topologie

Für das Videokonferenzsystem über IM wurde die *Stern*-Topologie ausgewählt. Dies hat mehrere Gründe: Es ist auch mit einer relativ schlechten Internet-/Netzwerkanbindung möglich, einer Videokonferenz beizutreten, da der Server die Weiterleitung der Datenströme übernimmt. Außerdem vereinfacht die zentralisierte Struktur die Verwaltung der Konferenz und ihrer Teilnehmer. Die Funktion eines realen Konferenzleiters kann gut auf die Administrationsfunktionen eines Servers abgebildet werden. Zwar bricht die gesamte Konferenz zusammen falls der Server ausfällt, doch wenn ein anderer Teilnehmer als der Server ausfällt ist die Verbindung der restlichen untereinander davon in keiner Weise betroffen. Es muss also nur ein Teilnehmer (der Konferenzleiter/Server) vor einem Ausfall ge-

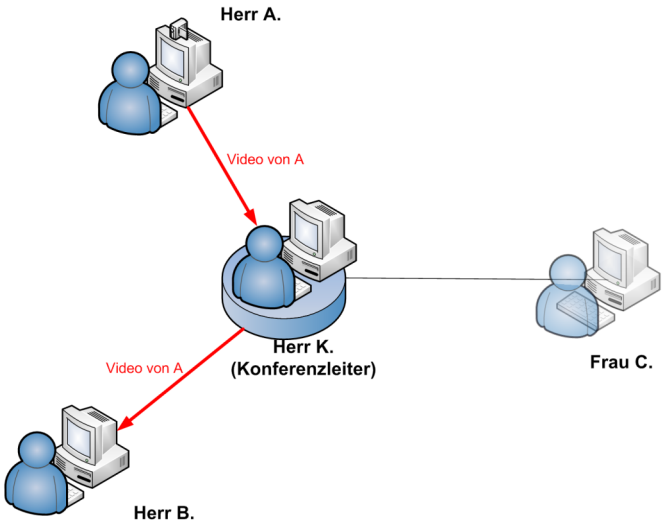

Abbildung 2.4: Stern-Struktur: Weg der Daten von A nach B

schützt werden[5] um eine allgemein stabile Konferenz zu garantieren.
Insgesamt benötigen alle Varianten die gleiche Anzahl von $2n^2 - 2n$ Verbindungen, $n*(n-1)$ eingehende und $n*(n-1)$ ausgehende. Sie unterscheiden sich nur in der Verteilung der Verbindungen auf die Knoten. Gegen Peer-to-Peer oder eine lineare Struktur spricht die fehlende Hierarchie, die eine Verwaltung der Verbindungen und Teilnehmer erschwert. Dafür ist hier die Belastung nahezu gleichmäßig verteilt. Eine Tree-Struktur ist zwar geordnet, allerdings existieren hier viele verschiedene Belastungsstufen, je nach Position im Baum. Dazu muss auch erst einmal eine sinnvolle Methode zum Aufbau der Struktur ausgewählt werden[6]. Da das Verwaltungsproblem stärker gewichtet wurde als die Belastung fiel die Wahl auf die Stern-Topologie.
Jeder Gesprächsteilnehmer ist also immer nur mit dem Konferenzleiter verbunden. Wenn zum Beispiel ein Gespräch zwischen Konferenzleiter (Person K) und 2 weiteren Teilnehmern stattfindet (Personen A und B), erhält B die Videodaten von A nur dadurch, dass K diese an B weiterleitet (Abbildung 2.4).

[5] durch leistungsfähige Hardware und andere Maßnahmen, die hier nicht weiter beschrieben werden sollen

[6] geographische Nähe wäre zum Beispiel möglich

IP-Adresse des Kontaktes
Port für den Empfang des Videokanal
Port für den Empfang des Audiokanal
Port für das Senden des Videokanal
Port für das Senden des Audiokanal
Sender- oder Empfängerflag
ID des Kontaktes, dessen Daten übertragen werden
Verschlüsselungsalgorithmus
Schlüssel

Tabelle 2.1: vereinbarte Parameter mit einem Kontakt

2.4 Verbindungsaufbau

Ein wichtiger Bestandteil der Konferenz ist der Verbindungsaufbau, da hier vorab alle nötigen Parameter ausgehandelt werden müssen, um die eigentliche Videoverbindung herstellen zu können.
Der Initiator der Verbindung zwischen den ersten beiden Konferenzteilnehmern wird automatisch der Konferenzleiter. Um mit der Konferenz verbunden werden zu können, muss eine Einladung des Konferenzleiters erfolgen. Wenn diese Einladung vom Empfänger bestätigt wurde, werden zwischen dem neuen Gesprächsteilnehmer und dem Konferenzleiter die Verbindungsparameter ausgehandelt. Dazu gehören die IP-Adressen, die zu nutzenden Ports, die Anzahl der Video- und Audiostreams sowie die Art der Verschlüsselung. Außerdem müssen Leiter und Teilnehmer bei jeder Verbindung markieren, ob sie Sender oder Empfänger sind, und die Daten welches Konferenzmitgliedes übertragen werden. Dies ist nötig, wenn der Server die Verbindungen zentral an alle anderen Teilnehmer weiterleitet. Eine genaue Übersicht, welche Informationen ein Kontakt für einen Audio-/Videostream benötigt, ist in der Tabelle 2.1 zu finden.
Die zweite Art des Verbindungsaufbau tritt auf, wenn ein neuer Teilnehmer einer Konferenz beitritt. In diesem Fall müssen alle schon bestehenden Teilnehmer von diesem Ereignis benachrichtigt werden. Aufgrund der Entscheidung für ein *sternförmiges Netzwerk* in 2.3.5 ist es nun Aufgabe des Servers, mit jeweils jedem Teilnehmer Verbindungsparameter auszuhandeln um diesem die Audio-/Videodaten des neuen Teilnehmers zu übermitteln.

2.5 Verschlüsselung

Die Audio- und Videodaten werden mit symmetrischen Schlüsseln verschlüsselt. Da die Datenmengen verhältnismäßig groß sind, ist eine asymmetrische Verschlüsselung aus Performancegründen nicht sinnvoll. Die Schlüssel werden mit dem Verfahren von Diffie und Hellman vereinbart[7]. Nachdem ein Teilnehmer sich authentifiziert hat, wird der Diffie-Hellman-Schlüssel ausgewechselt und ein globaler, für alle Konferenzteilnehmer gleicher Sessionkey benutzt.
Die Vereinbarung der Schlüssel wird in der *Bachelorarbeit von Florian Dietz* [Die09] behandelt.

[7] siehe dazu 3.9

Kapitel 3

Programme, Protokolle und Bibliotheken

Mit den folgenden Seiten soll eine Implementierung des oben beschriebenen Konzeptes einer Audio-/Videogruppenkommunikation für Instant Messenger vorgestellt werden, mit der sich ein wie in 2.1 beschriebenes Szenario durchführen lässt. Hierzu wurde als Instant Messenger *Spark* ausgewählt, dieser sowie weitere benutzte Komponenten und Protokolle werden in diesem Kapitel vorgestellt, bevor in Kapitel 4 dann auf die Implementierung eingegangen wird.

3.1 Spark IM

Spark IM ist ein in Java geschriebener quelloffener Instant Messenger der Firma *Jive Software*. Er nutzt das ebenfalls freie XMPP-Protokoll. Der Basisclient kann mit verschiedenen, teils auch kommerziellen, Plugins um viele Funktionen erweitert werden. Für Support und Weiterentwicklung von Spark IM hat Jive Software die Online-Community *ignite realtime* gegründet. Hier werden auch verschiedene Schwesterprojekte von Spark IM betrieben. Zum Beispiel der Jabber-Server Openfire oder der Jabber-Webclient SparkWeb. Aufgrund der Quelloffenheit, der aktiven Community und der Pluginmöglichkeit wurde Spark IM zur Implementierung des eVoting-über-IM ausgewählt.

3.2 XMPP (Jabber)

Das *Extensible Messaging and Presence Protocol* (XMPP) ist eine Sammlung von XML-Technologien zur Kommunikation und Statusbenachrichtigung mit Kontakten in Echtzeit. XMPP wurde im Oktober 2004 als *Internet Standard RFC 3920* [SA04] veröffentlicht. In RFC 3920 befinden sich nur die Kernelemente, weitere Standards zu XMPP stehen in den RFCs 3921,3922,3923,4854,4979 und 5122. Das Protokoll entstand aus dem seit 1999 entwickelten *Jabber*-Protokoll. XMPP wird von der *XMPP Standards Foundation* (XSF), einer Non-Profit-Organisation der XMPP-Community, ständig verbessert und neue Protokollerweiterungen werden hinzugefügt. Auch momentan werden RFC 3920 und 3921 wieder von einer Arbeitsgruppe der *Internet Engineering Task Force* (IETF) überarbeitet[1].
Hauptsächlich wird XMPP als Protokoll für *Instant Messaging*-Dienste genutzt. Es sind aber auch andere Anwendungsbereiche möglich, zum Beispiel in *Spielen*, im *Cloud Computing* oder zur allgemeinen *Übertragung von XML-Daten*. Die *XMPP Extension Protocols* (XEPs) genannten Erweiterungen des XMP-Protokolls sind auch größtenteils bekannte Komfortfunktionen für Instant Messaging, wie Multi-User-Chats, Nutzerprofile oder Dateiübertragung[2].

3.3 Smack

Die Bibliothek *Smack*[3] ist eine XMPP-Implementierung in Java. Sie stammt von derselben Community, die auch Spark entwickelt und wird auch zur Kommunikation per XMPP in Spark benutzt. Die eigentlichen Audio- und Videodaten werden nicht durch XML-codierte Nachrichten verschickt, da dies einen viel zu großen Overhead und dadurch eine unnötig hohe Bandbreitenbelastung verursachen würde. Allerdings werden über XMPP-Nachrichten die Verbindungsparameter ausgehandelt. Dazu wird das bereits in Spark integrierte Smack benutzt.

[1] Spezifikationsentwurf [SA09]
[2] Auflistung unter [XMP09]
[3] siehe [Rea]

3.4 Red5 und Java Media Framework

Es wurden verschiedene Möglichkeiten zur Übertragung und Darstellung der Videostreams untersucht.
Zur Zeit im Betastadium befindet sich die Integration von *red5* in Openfire und Spark. red5 ist ein seit 2005 existierendes OpenSource-Projekt (wie Spark in Java implementiert), welches Streaming von Flashvideos ermöglicht. Der Quellcode des Openfire-Plugins und des Spark-Plugins ist allerdings momentan noch nicht veröffentlicht, was eine für Gruppenkommunikation nötige Anpassung verhindert. Außerdem wäre die Verbindung zwangsläufig nur über Openfire als Jabber-Server möglich gewesen. Ein weiterer Grund, der gegen Red5 spricht ist die Tatsache, dass es nur auf Flash als Codec beschränkt ist und dass die Videodaten im Prinzip in einem integrierten Webbrowser angezeigt werden und nicht nativ in Spark.
Für Java existiert zur Verarbeitung von Videodaten das „Java Media Framework“ (JMF). Diese Bibliothek wurde von Sun (in Partnerschaft mit IBM) entwickelt, aber seit 2003 ist die Entwicklung auf Seiten von Sun eingestellt worden. Dies macht sich vor allem in der nicht vorhandenen Unterstützung aktueller Codecs bemerkbar. Da JMF eine proprietäre Bibliothek von Sun ist, ist es auch nicht möglich, das Framework selbstständig weiterzuentwickeln. Trotzdem stellt die JMF-API immer noch die komfortabelste Möglichkeit dar, um in Java Videodaten abzuspielen. Was JMF noch viel interessanter für die Videokonferenz macht, ist die integrierte RTP-Unterstützung, mit der Videodaten auch per Stream übertragen und empfangen werden können. Durch die Möglichkeit, den RTP-Videodatenstrom vor dem Senden noch einmal manuell zu verändern, lässt sich auch die Verschlüsselung günstig realisieren. Daher wurde zur Implementierung der Videokommunikation das Java Media Framework in der aktuellen Version 2.1.1 verwendet.

3.5 FMJ und lti-civil

Durch die oben erwähnten Probleme mit JMF und dem fehlenden Support von Sun wurde das Projekt „freedom for media in java“ (FMJ) gestartet. Dieses OpenSource-Projekt will eine Alternative zum JMF entwickeln, dabei allerdings vollständig kompatibel zur vorhandenen JMF-API bleiben. Dadurch kann bei existierenden JMF-Anwendungen einfach das JMF ohne große Änderungen durch FMJ

ersetzt werden. Aktuell sind die meisten Grundfunktionen von JMF bereits implementiert. FMJ bemüht sich zusätzlich besonders um die Integration aktueller Videocodecs (z.B. Theora).
Das Schwesterprojekt lti-civil wird zur Aufnahme von Webcams oder anderer Videoquellen benutzt. lti-civil ist momentan allerdings noch nicht eigenständig plattformübergreifend, sondern dient nur als Wrapper für die Bibliotheken der jeweiligen Betriebssysteme (DirectShow in Windows, Video4Linux in Linux, Quicktime in MacOS). Zum jetzigen Zeitpunkt sind FMJ und lti-civil noch nicht ausgereift genug um das JMF zu 100% zu ersetzen, allerdings sind sie eine aussichtsreiche Perspektive für den produktiven Einsatz der Videokonferenz in Spark.

3.6 Theora

Theora ist ein kostenloser, quelloffener Videocodec der *Xiph.org Foundation*[4]. Er ist vergleichbar mit MPEG-4-Codecs wie zum Beispiel *DivX* oder *H.264*. Für Audiodaten wird der ebenfalls von Xiph.org stammende *Vorbis*-Codec empfohlen. Durch die hohe Kompressionsrate eignet sich Theora wesentlich besser für die Gruppenkommunikation als die standardmäßig von JMF unterstützten, veralteten Codecs.

3.7 RTP

3.7.1 Allgemeines

Das *Real-Time Transport Protocol* (RTP) ist ein Protokoll zur Übertragung von Daten in Echtzeit, zum Beispiel von Videostreams. Die aktuelle Version ist in *RFC 3550* [SJ03] standardisiert.
RTP stellt nicht sicher, dass jedes Paket beim Empfänger ankommt. Verlorengegangene Pakete werden also nicht erneut versendet. Ebenfalls in [SJ03] standardisiert ist das *RTP Control Protocol* (RTCP). Dieses ermöglicht eine minimale Qualitätsüberprüfung und Verbindungskontrolle für RTP. Oft wird für RTP als Transportprotokoll UDP und IP als Netzwerkprotokoll verwendet, es sind aber auch andere Protokolle möglich. UDP hat den Vorteil, dass durch die in UDP-Paketen

[4] [Xip09] Entwicklung von Theora

vorhandene Prüfsumme fehlerhaft übertragene Pakete erkannt werden können. Es wird empfohlen, verschiedene Streams (zum Beispiel einmal Video und einmal Audio) in verschiedenen RTP-Sitzungen zu übertragen. Eine Motivation dafür ist zum Beispiel, dass es bei Audio-/Videoverbindungen möglich ist, nur die weniger Bandbreite benötigende Audioübertragung zu empfangen.
RTP wurde als Protokoll ausgewählt, da es für diese Art von Anwendung entwickelt wurde. Bei einer Audio- und/oder Videoverbindung ist es vernachlässigbar, dass manche Pakete verloren gehen. Dies resultiert im Extremfall nur in einem kurzen Bildausfall. Dafür werden die Daten aber möglichst zeitnah übertragen, Audio und Video sind nur minimal zeitlich verschoben zwischen Sender und Empfänger.

3.7.2 Struktur eines RTP-Paket

Der Header eines RTP-Paketes hat nach [SJ03] folgende Bestandteile:

Version Die Versionsnummer der verwendeten Protokollspezifikation (momentan aktuell ist *Version 2*).

Padding Dieses Bit gibt an, ob am Ende der Payload noch Bytes ohne Inhalt angehängt wurden, damit eine bestimmte Blockgröße erreicht wurde. Dies ist zum Beispiel wichtig wenn Blockchiffren benutzt werden zur Verschlüsselung der Payload.

Extension Wenn dieses Bit gesetzt ist, folgen direkt nach dem Header noch Informationen zu einer benutzten RTP-Erweiterung.

CSRC Zähler Gibt die Anzahl der *contributing source*-Bezeichner, die nach dem festen Teil des Headers stehen, an.

Markierung Diese Markierung kann benutzt werden um eine applikationsspezifische Markierung zu RTP-Paketen hinzuzufügen. Zum Beispiel können in einer Videoübertragung bestimmte Frames hiermit gekennzeichnet werden.

Payloadtyp Dieser Eintrag gibt den Typ der Payload-Daten im Paket an, zum Beispiel Videodaten im Format JPEG.

Sequenznummer Hiermit werden einzelne Pakete nach und nach durchnummeriert. Der Empfänger hat dadurch die Möglichkeit, angekommene Pakete noch nachträglich zu sortieren.

Zeitstempel Mit dem Zeitstempel ist es möglich, Daten genau zu synchronisieren, beispielsweise separat übermittelte Audio- und Videostreams.

SSRC SSRC bedeutet *synchronization source* . Dies ist eine zufällig gewählte Indentifikationsnummer zur Unterscheidung der verschiedenen Teilnehmer.

CSRC Liste Mit dieser Liste der *contributing sources* können verschiedene Datenquellen in einem RTP-Paket identifiziert werden, maximal sind 15 Einträge möglich.

Nach dem Header mit diesen Daten folgt die Payload des Paketes, wobei diese innerhalb der ersten Bytes durchaus noch Teile der Extensions oder andere Zusatzinformationen beinhalten kann.

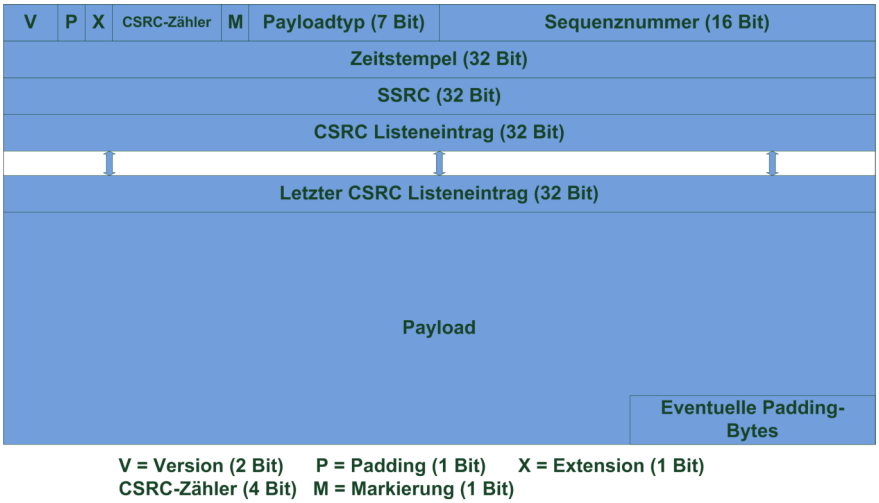

Abbildung 3.1: RTP-Paket

3.8 Advanced Encryption Standard

Der *Advanced Encryption Standard*[5] (AES) ist ein im Jahr 2000 festgelegter Verschlüsselungsstandard des US-amerikanischen *National Institute of Standards and Technology* (NIST). In einem Wettbewerb[6] wurde der Rijndael-Algorithmus ausgewählt, als neuer Standard für die Verschlüsselung von Staatsdokumenten zu dienen.
Rijndael ist ein Blockchiffrieralgorithmus mit symmetrischem Schlüssel. Er unterstützt in der AES-Spezifikation Schlüssellängen von 128,192 und 256 Bit.

3.9 Diffie-Hellman

Diffie-Hellman[7] ist ein Verfahren zur Vereinbarung eines gemeinsamen Schlüssels zwischen zwei Parteien. Durch den Austausch weniger Parameter und Zahlen können beide den gleichen Schlüssel errechnen, ohne dass eine Angreifer, der die ausgetauschten Werte mithört, ebenfalls den Schlüssel rekonstruieren kann. Eine Man-in-the-Middle Attacke[8] auf das Verfahren ist möglich, kann aber durch anschließende gegenseitige Verifikation ausgeschlossen werden.

[5] [Nat01]
[6] [oSN00]
[7] [Res99]
[8] siehe 5.2.1

Kapitel 4

Realisierung

4.1 Implementierung

Das in Kapitel 2 beschriebene Konzept wurde unter Nutzung der in Kapitel 3 beschriebenen Protokolle und Bibliotheken implementiert. Benutzt wurde die Programmiersprache Java und das System als Plugin „**SecureVC**“ für den in Kapitel 3.1 beschriebenen Instant Messenger *Spark* realisiert. Schließlich wurde das Plugin noch in ein übergeordnetes Plugin für eVoting-Verfahren[1], und zwar in den Teil *SecureMUC*[2], integriert.

4.1.1 Klassenübersicht

Die im Plugin enthaltenen Java-Klassen haben folgende Aufgaben:

SecureVC
: ist die Hauptpluginklasse, die das Plugin initialisiert.

ParameterManager
: verwaltet die Parameter der bestehenden Videoverbindungen. Des Weiteren verschickt und verarbeitet sie die Einladungen zu neuen Verbindungen.

VCParameters
: kombiniert alle für eine RTP-Verbindung nötigen Parameter in einer Klasse.

[1] [Mel08]
[2] siehe [Die09]

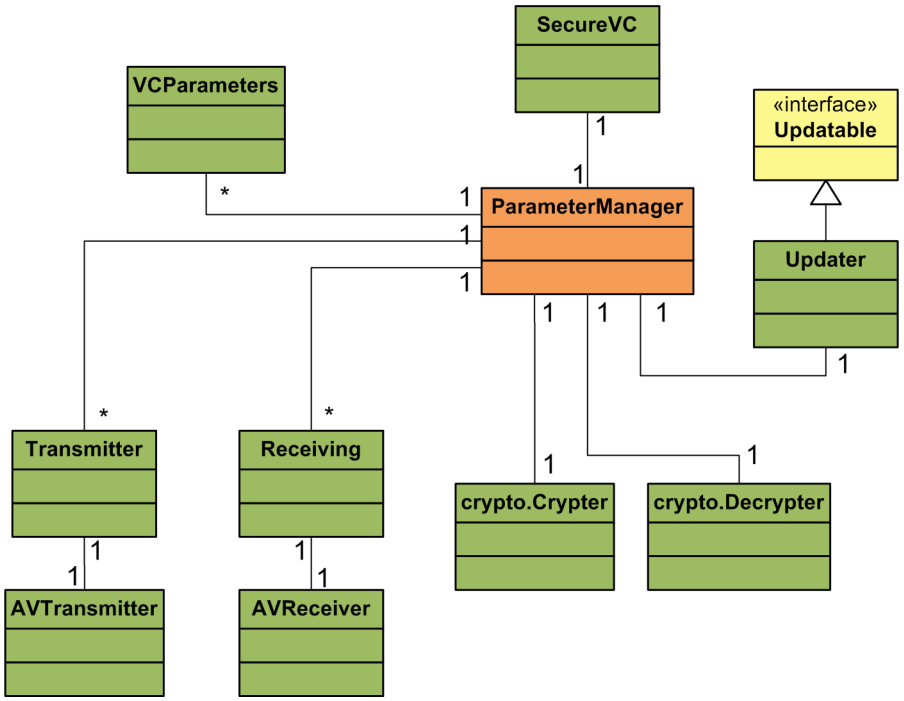

Abbildung 4.1: UML-Klassendiagramm

AVReceiver/AVTransmitter
: übernehmen das Empfangen beziehungsweise Senden der RTP-Datenströme.

Receiving/Transmission
: werden als eigene Threads gestartet für jede bestehende Videoverbindung.

Updater
: ist die Implementierung des Interface Updatable, um Kommunikation mit anderen Plugins (in diesem Fall SecureMUC) zu ermöglichen.

DualHashMap
: funktioniert wie eine java.util.HashMap, arbeitet aber mit zwei Schlüsseln statt einem. Wird benutzt um die Verbindungsparameter zu verwalten. Als eindeutiger Schlüssel dient eine Kombination aus der JID des Kontaktes und der Angabe, wessen Audio-/Videodaten über diese Verbindung übertragen werden.

crypto.Crypter
: verschlüsselt die Payload von RTP-Paketen mit AES-128.

crypto.Decrypter
entschlüsselt die RTP-Payloads wieder.

tools.IPRetriever
wird benutzt um die öffentliche IP herauszufinden, damit diese den Kontakten mitgeteilt werden kann.

tools.JIDTools
enthält Methoden, die mit der Jabber-ID arbeiten und die von mehreren verschiedenen Klassen benötigt werden.

In Abbildung 4.1 sind die wichtigsten Klassen noch einmal in Diagrammform dargestellt.

4.1.2 Allgemeines zum Verbindungsaufbau

Über einen Kontextmenüeintrag in der Kontaktliste, beziehungsweise im Falle der Integration in das eVoting-Plugin über den Verifizierungsbutton, wird eine „Einladung zur Videokonferenz“ an die JID des ausgewählten Kontaktes gesendet. JID ist der *Jabber Identifier*, welcher Jabber-Nutzer eindeutig mit Adressen der Form *name@jabberserver/resource* identifiziert. Der Kontakt kann diese Anfrage ablehnen, in diesem Fall geschieht nichts weiter. Falls der angefragte Kontakt zustimmt, vereinbaren Client und Server (dieser entspricht dem *Initiator* der Konferenz) automatisch über XMPP-Nachrichten welche Ports zum Aufbau der RTP-Verbindungen genutzt werden.
Es wird jeweils ein eigener Port für jeden Video- und jeden Audiodatenstrom benötigt. Im Falle einer Konferenz zwischen 4 Teilnehmern benötigt also jeder Teilnehmer 2 Ports zum Senden der eigenen Daten sowie 3*2 Ports für die Streams der anderen Teilnehmer. Standardmäßiger Basisport ist 10200, von dort werden die Portnummern immer in 2-er Schritten erhöht. Dies geschieht damit die jeweils ungeraden Portzahlen für RTCP-Verbindungen genutzt werden können.
Zur Verschlüsselung der RTP-Daten wird standardmäßig AES-128 vereinbart. Dies kann aber auch durch andere Algorithmen ersetzt werden. Der nötige Schlüssel ist dem Sender und dem Empfänger bereits bekannt.
Danach wird die Videoverbindung aufgebaut.

4.1.3 Protokoll zur Parametervereinbarung

Tritt ein neuer Client der Konferenz bei, handelt er mit dem Server über mehrere XMPP-Nachrichten die im Konzept genannten Parameter aus. Diese werden mit Hilfe der *Smack API* als *properties* in XMPP-Nachrichten eingebettet. Die folgende Auflistung erklärt die benutzten *property*-Namen und ihre Zuordnung zu Verbindungsparametern.

isSecureVCMessage
: identifiziert eine XMPP-Nachricht als zum Videokonferenzplugin zugehörend. Nur mit dieser *property* ausgestattete Nachrichten werden vom Plugin weiterverarbeitet.

messageTypeID
: gibt den aktuellen Status der Verhandlung an. Der initiale Zustand bei einer Einladung ist null. Die verschiedenen SecureVC-messageTypes werden in Tabelle 4.1 vorgestellt.

serverIP
: gibt die IP des Servers, also des Konferenzleiters an.

serverSVideoPort
: gibt den Port an, über den der Server bei dieser Verbindung den Videokanal sendet.

serverSAudioPort
: gibt den Port an, über den der Server bei dieser Verbindung den Audiokanal sendet.

serverRVideoPort
: gibt den Port an, über den der Server bei dieser Verbindung den Videokanal des Kontaktes empfängt.

serverRAudioPort
: gibt den Port an, über den der Server bei dieser Verbindung den Audiokanal des Kontaktes empfängt.

clientIP
: ist die IP des Client.

clientRVideoPort
ist der Port, über den der Client die Videodaten vom Server empfängt.

clientRAudioPort
ist der Port, über den der Client die Audiodaten vom Server empfängt.

clientSVideoPort
ist der Port, über den der Client seine Videodaten an den Server sendet.

clientSAudioPort
ist der Port, über den der Client seine Audiodaten an den Server sendet.

videoOf
gibt an, wessen Audio- und Videostrom übertragen wird. Hiermit kann der Client A die beiden Quellen, die er vom Server empfängt, auseinanderhalten (die Daten des Servers und die über den Server weitergeleiteten Daten von Client B).

Die IP ist im Normalfall die Internet-IP der Nutzer. Die benutzten Ports müssen, wenn sich einer der Teilnehmer hinter einem Router befindet, weitergeleitet werden. Falls sich 2 Nutzer im gleichen lokalen Netzwerk befinden, wird nicht die Internet-IP sondern die lokale IP ausgetauscht. Dadurch wird die Verbindung auch nur im lokalen Netzwerk aufgebaut und die Internetverbindung nicht unnötig belastet. Der grundsätzliche Ablauf ist in Tabelle 4.1 bereits zu erkennen, soll hier aber nochmals erläutert werden. Zuerst verschickt der Server die Einladung an einen Kontakt (*INVITE*). Dieser antwortet mit einer Zustimmung (*INV_ACK*) oder lehnt es ab der Konferenz beizutreten (*INV_DENY*). Erhält der Server eine Ablehnung als Antwort, wird der Verbindungsaufbau abgebrochen. Im Falle einer Bestätigung errechnet der Server freie Ports für die Übertragung seiner Daten und den Empfang der Daten des neuen Kontaktes. Diese werden zusammen mit der IP-Adresse des Servers mit einer *INV_SERVERDATA*-Nachricht an den Client geschickt. Der Client trägt die empfangenen Daten in seiner Verbindungstabelle ein, errechnet die eigenen Ports für die Verbindung und bestimmt seine IP. Diese Informationen schickt er dann mit einer als *INV_CLIENTDATA* markierten Nachricht an den Server zurück. Dieser übernimmt die Portinformationen. Außerdem überprüft er anhand der Client-IP, ob sich beide im gleichen lokalen Netzwerk befinden. Ist dies nicht der Fall, bestätigt er dem Client die Verbindung mit *INV_APPROVED* und baut auf seiner Seite die RTP-Verbindungen

Tabelle 4.1: messageTypeIDs in SecureVC

Typnummer	messageTypeID	Status der Verhandlung
0	INVITE	Empfang einer Einladung
-1	INV_DENY	Einladung wurde abgelehnt
1	INV_ACK	Einladung wurde angenommen
2	INV_SERVERDATA	Portinformationen des Servers
3	INV_CLIENTDATA	Portinformationen des Client
4	INV_SERVERNEWIP	Es muss auf die lokale IP gewechselt werden
5	INV_CLIENTNEWIP	Die lokale IP des Client
6	INV_APPROVED	Bestätigung vom Server, Daten komplett
10	NEW_SERVERDATA	Neuer Kontakt mit Ports des Server für diesen Kanal
11	NEW_CLIENTDATA	Ports des Client für den neuen Kanal
12	NEW_APPROVED	Bestätigung vom Server, Daten komplett

auf. Wenn der Client die Bestätigung erhalten hat, beginnt dieser ebenfalls die Übertragung und den Empfang der Audio-/Videostreams.
Befinden sich Server und Client in einem lokalen Netzwerk, verschickt der Server noch keine Bestätigung. Stattdessen sendet er eine *INV_SERVERIP*-Nachricht mit seiner lokalen IP. Der Client antwortet per *INV_CLIENTIP*, die seine lokale IP enthält. Auf diese Weise können nun beide sich verbinden ohne einen Umweg über das Internet machen zu müssen. Nach Erhalt der neuen Client-IP sendet der Server nun das *INV_APPROVED* und die Verbindungen werden auf beiden Seiten aufgebaut. Der Ablauf ist auch nochmal in Abbildung 4.2 dargestellt.

4.1.4 Events

Während einer Konferenz können verschiedene *Events* auftreten, die von den Konferenzteilnehmern verarbeitet werden müssen. Diese Events werden vom Multiuser-Chat-Bestandteil[3] des eVoting-Plugins an SecureVC übergeben. Es folgt

[3]SecureMUC

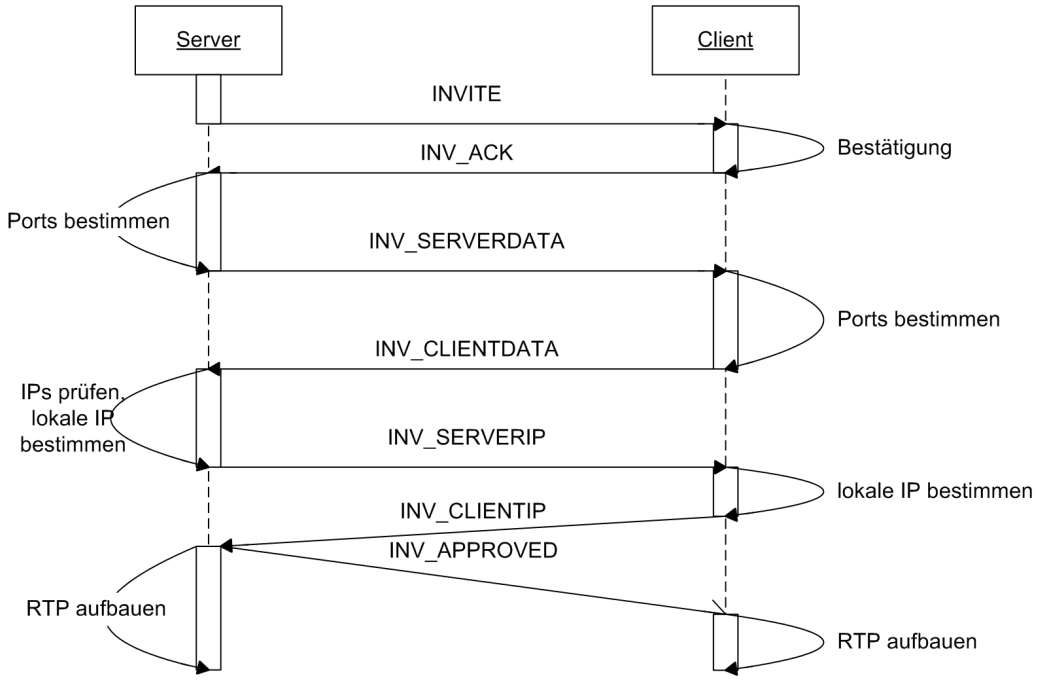

Abbildung 4.2: Verbindungsaufbau im lokalen Netzwerk

eine Auflistung der Events und die Reaktionen von SecureVC darauf.

serverseitige Events:

IsInitiator:

Die Plugininstanz fungiert nun als Initiator/ Konferenzleiter/ Server. Dazu wird ein entsprechendes Flag im ParameterManager gesetzt.

MemberJoin:

Ein neuer verifizierter Teilnehmer ist der Konferenz beigetreten. Seine JID wird als Argument mitübergeben. Es müssen alle anderen Teilnehmer benachrichtigt werden, dass ein neuer Kontakt besteht und es müssen jeweils Verbindungen mit den bestehenden Kontakten ausgehandelt werden, um die Daten des neuen für diese bereitzustellen.

MemberLeave:

Wenn ein Teilnehmer die Konferenz verlässt, muss der Server selbst die Verbindung mit diesem beenden, sowie alle anderen Teilnehmer benachrichtigen, dass diese ebenfalls die Verbindungen, über die sie die Daten

des ehemaligen Kontaktes empfangen haben, schließen.

VerificationStarted:
Eine Benachrichtigung, dass die Verifikation eines Teilnehmers begonnen hat. Der Server muss nun mit dem neuen Kontakt eine Verbindung aushandeln. Die Verbindung wird mit dem vereinbarten DH-Schlüssel verschlüsselt. Nachdem diese aufgebaut ist, kann der Kontakt anhand eines Hashwertes, der über die Audio-/Videoverbindung verglichen wird, verifiziert werden.

VerificationSuccessful:
Dieses Event wird ausgelöst, wenn der oben genannte Hashwert als korrekt bestätigt wird. Dadurch gilt der Teilnehmer als verifiziert und die Verschlüsselung wechselt auf den globalen Sessionkey. Es wird *MemberJoin* ausgelöst, um die anderen Teilnehmer zu benachrichtigen.

EndOfSession:
Die Sitzung wird beendet. Alle Verbindungen werden geschlossen.

clientseitige Events:

IsClient:
Die Plugininstanz ist Client.

SessionKeyReceived:
Nach erfolgreicher Verifikation des Clients erhält dieser vom Server den SessionKey (noch verschlüsselt mit dem DH-Schlüssel). Nun wechselt der Client die Verschlüsselung auf den SessionKey.

MemberJoin:
Ein neuer verifizierter Teilnehmer ist beigetreten. Es wird die Anfrage des Servers zur Vereinbarung der Übertragungsparameter abgewartet. Sind diese vorhanden, wird die Verbindung zum Empfang der Audio-/Videostreams des neuen Kontaktes aufgebaut.

MemberLeave:
Die bestehenden Verbindungen, die Daten des die Sitzung verlassenden Teilnehmers enthalten, werden getrennt.

Disconnected:

Wird ausgelöst wenn der Konferenzleiter die Konferenz beendet, wenn der Client die Sitzung selbstständig verlässt oder wenn durch einen Verbindungsfehler die Verbindung abbricht. Noch vorhandene Verbindungen werden geschlossen.

4.1.5 Verschlüsselung

Die Payload der RTP-Ströme wird symmetrisch mittels AES[4] verschlüsselt. Dazu wird ein Schlüssel der Länge 128 Bit benutzt. Längere Schlüssel werden zur Zeit noch nicht vom Package *javax.crypto* unterstützt[5]. Die ausgetauschten DH-Schlüssel beziehungsweise der Sessionkey werden als *Salt* benutzt um die AES-Schlüssel zu erstellen.

4.2 Anleitung

4.2.1 Installation

Um eine verschlüsselte Videokonferenz nutzen zu können, wird folgendes benötigt:

- Java (ab Version 6)
- Spark IM (ab Version 2.5.8)
- eVoting-Plugin für Spark
- der für die Videoübertragung genutzte Codec (zum Beispiel Quicktime oder Theora)
- eine Webcam

Das Plugin installiert man in Spark IM, indem man die jar-Datei in den Unterordner „plugins/“ im Sparkverzeichnis kopiert. Nun ist das Plugin nach einem Neustart von Spark IM verfügbar. Über das eVoting-Menü kann man in den *Einstellungen* nun optional noch den zu verwendeten Basisport einstellen. Außerdem

[4] Advanced Encryption Standard[Nat01]
[5] siehe [Sri03]

kann eine Videodatei ausgewählt werden, die gesendet werden soll, falls keine Webcam vorhanden ist.

4.2.2 Aufbau einer Gruppenkommunikation

Um eine Verbindung wie im Beispielszenario 2.1 aufzubauen, müssen folgende Schritte durchgeführt werden. Sämtliche Teilnehmer müssen Spark nach der obigen Anleitung installieren.
Falls noch kein Jabber-Account vorhanden ist, besteht die Möglichkeit einen neuen Account anzulegen (entsprechende Schaltfläche im Loginformular von Spark auswählen). Der Nickname kann beliebig gewählt werden, als Server kann der zu Testzwecken installierte *openfire.it-risk.iwvi.uni-koblenz.de* eingetragen werden[6]. Nun kann eine Person die Konferenz starten (dieser Teil gehört zu SecureMUC). Wenn in der Konferenz mit der Verifikation eines Kontaktes begonnen wird, wird die Videoverbindung aufgebaut. Es müssen nach und nach alle Kontakte verifiziert werden, bis die komplette Gruppenkommunikation mit allen gewünschten Kontakten aufgebaut ist.

4.3 Integration in das eVoting-Plugin

SecureVC sollte in Kombination mit *SecureMUC* in ein *eVoting*-Plugin für Spark IM integriert werden. Da die Videokonferenz von SecureVC nur durch Aufruf aus SecureMUC aufgebaut wird, musste ein Interface zur Kommunikation mit SecureMUC vereinbart werden. Dadurch entstand das Interface Updatable mit der Methode „**public void** update(String evt, Object args)“. Über die Variable evt wird der Eventname übergeben, mit dem Objekt args werden verschiedene Parameter zum Event weitergereicht. Eine Auflistung und Erklärung der verschiedenen Events wurde bereits in Kapitel 4.1.4 gegeben.
Für das Hinzufügen neuer Kontakte, beziehungsweise für das Starten der Verifikation, implementiert SecureVC das Interface VoipClient mit den Methoden „**public void** startCall(String jid)“ und „**public void** endCall(String jid)“. Diese Methoden rufen dann den eigentlichen Verbindungsaufbau auf.

[6]Eine umfangreiche Serverliste findet sich unter [jab]

Kapitel 5

Sicherheitsanalyse der Implementierung

In diesem Kapitel werden die verschiedenen Eigenschaften charakterisiert, die eine Implementierung des in Kapitel 2 vorgestellten Konzeptes erfüllen sollte. Zuerst werden wichtige IT-Sicherheitseigenschaften genannt, danach eine Auswahl bekannter *Angriffe* auf diese Anforderungen vorgestellt. In den *Sicherheitsmaßnahmen* werden Maßnahmen erklärt, mit denen die zur dieser Arbeit gehörende Implemtierung *SecureVC* diese Angriffe verhindert beziehungsweise abwehrt, damit die Audio-/Videogruppenkommunikation den *Sicherheitsanforderungen* gerecht wird.
Prinzipiell kann die Implementierung an verschiedenen Stellen angegriffen werden. Sollte ein Angreifer Kontrolle über ein benutztes IM-System haben, kann die Implementierung einfach durch einfach durch eine ersetzt werden, welche über *Backdoors* dem Angreifer alle sicherheitskritischen Daten offenlegt.
Ein Angriffspunkt von außerhalb des Systems ist das Protokoll zum Aushandeln der Verbindungsparameter, die andere Angriffsmöglichkeit ist die Übertragung der RTP-Datenströme.
Ebenfalls denkbar sind Schwachstellen in der Implementierung von *Spark IM* oder in der *Java Virtual Machine*. Diese letztgenannten Stellen liegen allerdings außerhalb des Fokus dieser Arbeit und werden daher nicht behandelt.

5.1 Sicherheitsanforderungen

5.1.1 Integrität

Die Integrität einer Nachricht besagt, dass eine Nachricht nicht in ihrem Inhalt verändert wurde. Der Empfänger erhält folglich genau den gleichen Nachrichtentext beziehungsweise die gleichen Daten, die der Sender auch gesendet hat.
Bezüglich der Videokommunikation ist die Integrität dann verletzt, wenn ein Angreifer die Möglichkeit hat in den Videostream oder den Audiostream einzugreifen und diesen zu stören. Dies könnte geschehen, indem er die Daten entschlüsselt, verändert (zum Beispiel in zufällige Bits um das Videobild zu zerstören) und an den Empfänger schickt.

5.1.2 Vertraulichkeit

Die Vertraulichkeit einer Nachricht ist gewährleistet, wenn nur der Sender und der Empfänger den Inhalt der Nachricht kennen. Dies kann zum Beispiel durch ein modernes Verschlüsselungsverfahren erreicht werden, welches von Angreifern nicht decodiert werden kann.
Die Vertraulichkeit ist gestört, wenn zum Beispiel die Audiodaten mitgelesen und vom Angreifer entschlüsselt werden können. Auf die Verbindungsparameter erstreckt sich die Vertraulichkeit nicht, da diese Daten unverschlüsselt übermittelt werden. Ein Angreifer kann also leicht erfahren, wer wann mit wem kommuniziert hat. Der Inhalt der Kommunikation wird allerdings geschützt.

5.1.3 Authentizität

Die Authentizität einer Nachricht beziehungsweise eines Nachrichtenkontaktes ist sichergestellt, wenn nachgewiesen werden kann, dass ein Sender die Person ist, für die er sich ausgibt, und wenn zusätzlich die Integrität der Nachricht gewährleistet ist. Durch Überprüfen der Authentizität kann verhindert werden, dass andere Personen unter falschem Namen Nachrichten verschicken.
Diese Eigenschaft bezieht sich hauptsächlich auf Textnachrichten, da bei einer Videoübertragung zwischen 2 Leuten, die sich kennen, sofort bemerkt wird dass der andere nicht die Person ist, für die sie sich ausgibt. Bei der Integration wird

die Authentizität der Kontakte von SecureMUC geprüft, indem über die Videoverbindung ein Hash-Wert verglichen wird.

5.1.4 Verfügbarkeit

Die Verfügbarkeit eines Systems muss möglichst 100% sein und ein Ausfall desselben muss vermieden werden. Ein sicheres kryptographisches Verfahren, welches aber nur über ein selten verfügbares System möglich ist, kann nicht zuverlässig genutzt werden. Für die Videokonferenz ist die Verfügbarkeit nicht gewährleistet, wenn der Server (der aufgrund der gewählten Infrastruktur wichtigstes Glied der Konferenz ist) von Angreifern mit einfachen Mitteln außer Betrieb gesetzt werden kann.

5.2 Angriffsmöglichkeiten und Gegenmaßnahmen

5.2.1 Man-in-the-Middle

Bei einem *Man-in-the-Middle* Angriff (MITM), versucht ein Angreifer (hier: Eve) sich zwischen zwei miteinander Kommunizierende zu schalten (hier: Alice und Bob).
Eve fängt die Nachrichten der Beiden füreinander ab und sendet eigene in deren Namen. Zum Beispiel schickt Alice die Nachricht „Wie alt bist du?“ an Bob, und Eve fängt diese ab. Eve sendet jetzt „Wie ist deine Handynummer?“ an Bob, gibt sich aber dabei als Alice aus. Bob gibt seine Handynummer nur wenigen Leuten, vertraut aber Alice und schickt ihr seine Nummer. Doch seine Antwort wird ebenfalls von Eve abgefangen, und sie weiß jetzt ohne Bobs Wissen und Zustimmung dessen Handynummer. Nun antwortet Eve noch -als Bob- auf Alices Frage nach Bobs Alter.
Durch dieses Zwischenschalten hat sich Eve unberechtigt Informationen verschafft, ohne dass Alice oder Bob etwas davon bemerkt haben. Dieser Angriff (dargestellt in Abbildung 5.1) verletzt Authentizität und Vertraulichkeit der Kommunikation.
Solch ein Angriff kann bei SecureVC auf die Aushandlung der Parameter erfolgen. Ein *„mitm“* könnte sich gegenüber einem Server *S* als Client ausgeben und gegenüber einem Jabberclient *C* als SecureVC-Server, der eine Einladung ver-

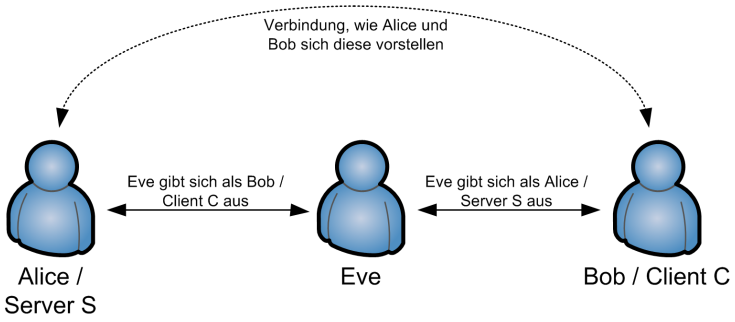

Abbildung 5.1: Man-in-the-Middle

schickt. Um die vom Server geschickte XMPP-Nachricht an eine bestimmte JID[1] abzufangen beziehungsweise umzuleiten bedarf es allerdings eines tiefen Eingriffs in die Kommunikation zwischen den betroffenen Jabber-Servern. Gelingt dies, kann der Angreifer in der Rolle eines Servers mit dem Client *C* eine Verbindung aushandeln und ebenso, in Clientrolle, mit dem Server *S* eine Videoverbindung vereinbaren und aufbauen. Diese beiden Verbindungen leitet er aneinander weiter und die Kontakte *C* und *S* denken vorerst, sie würden miteinander ohne Zwischenstation kommuniziern. Nun allerdings kommt die in SecureMUC implementierte Verifikation hinzu. Wenn diese durchgeführt wird, bemerken *C* und *S*, dass sie unterschiedliche Hashwerte haben und daher nicht direkt miteinander verbunden sein können.
Falls also ein *Man-in-the-middle*-Angriff durchgeführt werden kann, wird er spätestens durch die Verifikation entdeckt.

5.2.2 Denial of Service

Ein Denial of Service Angriff richtet sich gegen die Verfügbarkeit eines Systems. Hierbei wird an das System eine so große Menge an Anfragen gerichtet, dass die Leistung des Systems zu schwach ist, um diese alle zu bearbeiten. Dadurch kann entweder das ganze System zum Absturz gebracht werden oder auch die Verarbeitung von richtigen Anfragen blockiert werden, da diese sich in eine lange Warteschlange einreihen müssen.
Ein IT-System kann diesen Angriff nur begrenzt verhindern. Eine Möglichkeit ist, eine Erkennung für große Anfragemengen in kurzer Zeit einzubauen, und diese

[1] siehe 4.1.2

Anfragen dann zu verwerfen. Dabei kann es allerdings passieren, dass echte, innerhalb dieses Zeitraums getätigte Anfragen ebenfalls nicht bearbeitet werden. Natürlich kann auch die Leistung eines Systems erhöht werden um die maximal verarbeitbare Menge an Anfragen zu erhöhen.
Gegen *Denial-of-Service* wird in SecureVC durch einen Sicherungsmechanismus geschützt. Um das System nicht zu überlasten wird der Empfang zeitweise blockiert, falls innerhalb weniger Sekunden die *SecureVC*-XMPP-Paketen eine bestimmte Mengenzahl überschreiten. Dadurch kann unter Umständen auch das Herstellen zusätzlicher echter Verbindungen verhindert werden, in jedem Fall ist aber zumindest das Weiterbestehen vorhandener Verbindungen gesichert.

5.2.3 Brute-Force Entschlüsselung

Als Brute-Force bezeichnet man ein Verfahren zur Entschlüsselung von Daten ohne Kenntnis des Schlüssels. Es werden sequentiell alle möglichen Kombinationen, die ein Schlüssel haben kann, ausprobiert, bis der richtige Schlüssel gefunden wurde. Die Aufwendigkeit des Verfahrens hängt wesentlich von der Länge des Schlüssels und der Anzahl der verwendbaren Zeichen ab. Kann ein solcher Angriff erfolgreich sein, ist die Vertraulichkeit der Nachrichten nicht mehr gewährleistet.
Um die Entschlüsselung geheimer Daten mittels Brute-Force zu verhindern, muss nur ein ausreichend komplexer Schlüssel gewählt werden, für den Brute-Force mit heutiger Rechenleistung nicht durchführbar ist.
Brute-Force Attacken auf die Implementierung sind möglich, da mit geeigneter Rechenleistung das in dieser Implementierung verwendete AES-128 entschlüsselt werden kann[2]. Allerdings kann dieser Algorithmus für den produktiven Einsatz durch einen stärkeren bzw. einen mit längerem Schlüssel ersetzt werden.

5.2.4 Entschlüsselung mit bekanntem Klartext

Wenn der Klartext oder ein Stück des Klartextes zu einem Geheimtext bekannt ist, vereinfacht dies bei Verwendung kurzer Schlüssel erheblich das Herausfinden des benutzten Schlüssels um den Rest des Geheimtextes zu entschlüsseln. Daher sollte vermieden werden, dass Angreifer Rückschlüsse auf Teile des Klar-

[2] der Grund für die Verwendung wird in 4.1.5 genannt

textes ziehen können. Teile des Klartextes können, wenn der Kontext bekannt ist, unter Umständen schnell herausgefunden werden (zum Beispiel die Grußformel in einem Brief).
Deshalb ist es zum Beispiel sinnvoll, dass die Headerdaten von Kommunikationsprotokollen nicht mitverschlüsselt werden, weil der Aufbau und Inhalt dieser Daten Angreifern höchstwahrscheinlich bekannt ist. Da Videodaten im Gegensatz zu Text auch relativ chaotisch sind und es schwer ist, darin Muster zu finden, ist ein solcher Angriff nicht einfach durchzuführen.
Daher verschlüsselt SecureVC (da die Implementierung nur Schlüssel der Länge 128 Bit verwendet) nur die Payload der RTP-Pakete, also nur die Audio- und Videodaten selbst werden verschlüsselt. Diese Methode verrät Angreifern, die die Verbindung abhören zwar die verschiedenen Verbindungsparameter und zum Beispiel auch die beteiligten Kontakt-JIDs, verhindert aber das Brechen der Verschlüsselung durch bekannte Klartextteile. Mit starken Algorithmen und Schlüssellängen kann ein solcher Angriff allerdings sicher verhindert werden.

5.2.5 Social Engineering

Social Engineering greift eines der schwächsten Glieder in der IT-Sicherheit an: die menschlichen Benutzer selbst. Dies kann auf viele verschiedene Weisen geschehen. Ein häufig gewähltes Beispiel ist, dass ein Angreifer sich am Telefon als Bankangestellter ausgibt, der dringend die PIN für das Konto des Opfers benötigt, um eine fälschlicherweise stattgefundene Abbuchung rückgängig zu machen. Oft werden die Opfer hierbei in eine Drucksituation versetzt. Unter anderem wird durch Social Engineering die Authentizitätseigenschaft verletzt.
Normalerweise ist die einzige Möglichkeit, ein System gegen diesen Angriff zu sichern, die Benutzer auf die Problematik aufmerksam zu machen, damit sie ihre Logindaten oder andere vertrauliche Informationen nicht weitergeben.
Bezüglich der Videokonferenz funktioniert es allerdings nicht, mit durch *Social Engineering* beschafften Logindaten an der Konferenz teilzunehmen, sich als eine andere Person auszugeben[3] und bei der Konferenz mitzuhören. Denn durch die Audio-/Videoübertragung kann direkt die Identität der Gesprächspartner überprüft und ein unerwünschter Teilnehmer erkannt werden. Eine Ausnahme, bei der dieser Angriff trotzdem möglich ist, wäre hier eine erfolgreiche Verkleidung

[3] Maskerade

mit Gesichtsmaske und Stimmenimitation, allerdings kann dies nicht über IT-Sicherheitsmaßnahmen verhindert werden.

Kapitel 6

Fazit

Es wurde erfolgreich eine sichere Audio-/Videogruppenkommunikation für einen Instant Messenger entwickelt. Mit dieser Arbeit konnte gezeigt werden, dass ein solches System prinzipiell möglich ist und welche Komponenten es benötigt. Das System wurde nach dem derzeitigen Stand der Technik gegen die potientellen Gefahren in der Informationstechnik abgesichert. Die Abläufe einer realen Konferenz wurden auf die virtuelle Videoübertragung abgebildet. Das erarbeitete Konzept kann je nach Fokussierung auch auf andere Situationen angepasst werden. Für den produktiven Einsatz empfielt es sich zum Beispiel, noch bestehende Sicherheitsprobleme zu beheben (z.B. die 128-Bit Begrenzung).
Allgemein muss gesagt werden, dass gerade Systeme, die in Instant Messengern integriert werden, gute Chancen haben häufig genutzt zu werden. Denn im Privatleben und auch immer mehr im Geschäftsalltag nutzen Menschen Instant Messenger um miteinander zu kommunizieren.

Literaturverzeichnis

[Die09] Florian Dietz. Absicherung und Verifikation einer Instant Messaging Kommunikation über DH-Key Agreement, 2009.

[jab] jabberes.org. Jabber/xmpp server list. http://www.jabberes.org/servers/servers_by_times_online.html.

[lea08] learn-networking.com Team. A Guide to Network Topology. *Learn Networking*, Januar 2008.

[Mel08] Anastasia Meletiadou. E-Voting auf Grundlage eines Instant Messaging Systems. *Beherrschbare Systeme - dank Informatik. Lecture Notes in Informatics 133*, Bd. 1, 2008.

[Mit08] Bradley Mitchell. Network Topologies. *About.com*, 2008.

[Nat01] National Institute of Standards and Technology (NIST). Advanced Encryption Standard. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.

[oSN00] National Institute of Standards and Technology (NIST). Commerce Department Announces Winner of Global Information Security Competition. http://www.nist.gov/public_affairs/releases/g00-176.htm, 2000.

[Rea] Ignite Realtime. Smack API. http://www.igniterealtime.org/projects/smack/index.jsp.

[Res99] E. Rescorla. Diffie-Hellman Key Agreement Method. RFC 2631 (Proposed Standard), Juni 1999.

[SA04] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard), Oktober 2004.

[SA09] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core Internet Draft. RFC 3920 Internet Draft, März 2009.

[SJ03] Frederick Schulzrinne, Casner and Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), Juli 2003.

[Sri03] Rags Srinivas. Using AES with Java Technology. *Sun Developer Network*, Juni 2003.

[Xip09] Xiph.org Foundation. The Theora Videocodec. http://www.theora.org, 2009.

[XMP09] XMPP Standards Foundation. XMPP Extension Protocols. http://www.xmpp.org/extensions, 2009.