Hochschule für Technik und Wirtschaft des Saarlandes University of Applied Sciences

Design einer mobilen Anwendung zur verschlüsselten Sprachkommunikation auf Basis des Android Betriebssystems

Sebastian Weisgerber

30. September 2011

Prüfer: Prof. Dr. Reinhard Brocks

Zweitprüfer: Prof. Dr.-Ing. Damian Weber

Betreuer: Dr. Dirk Leinenbach

Betreuer: Dipl.-Inf. Tim Bautz

Sperrvermerk

Die vorliegende Arbeit beinhaltet interne vertrauliche Informationen der Firma consistec Engineering & Consulting GmbH. Die Weitergabe des Inhalts der Arbeit im Gesamten oder in Teilen sowie das Anfertigen von Kopien oder Abschriften – auch in digitaler Form – sind grundsätzlich untersagt. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma consistec Engineering & Consulting GmbH.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Master-Thesis selbständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet habe. Dieses Thema bzw. diese Arbeit wurde bisher weder in gleicher noch in ähnlicher Form in einem anderen Prüfungsverfahren vorgelegt.

Ort und Datum: ..... Name: .....

Danksagung

Danken möchte ich Herrn Prof. Reinhard Brocks für die sehr gute Betreuung meiner Arbeit und insbesondere für die Beratung im Bereich graphischer Modellierungssprachen.

Vielen Dank an Herrn Prof. Damian Weber für die Betreuung meiner Arbeit.

Ein großer Dank geht auch an Dr. Dirk Leinenbach, für die erstklassige Betreuung, Beratung und Unterstützung während der gesamten Entwicklungszeit dieser Master-Thesis.

Danken möchte ich auch Tim Bautz für seine Betreuung und die Idee zu dieser Master-Thesis.

Ein besonderer Dank geht auch an meine Eltern für die wunderbare Unterstützung während meiner gesamten Studienzeit.

Ein Dank geht auch an Matthias und Markus für die Beratung und die Gespräche bei heißem Kaffee während der Entwicklungszeit dieser Thesis.

Danke an Stefan für den Support bei Software PBXs.

Danke an Sina für das Korrekturlesen.

Danke an die gesamte consistec GmbH für die Mittagstische und die erstklassige Arbeitsatmosphäre.

Abstract

Die unterschiedlichen Möglichkeiten und die Realisierung der abhörsicheren und verschlüsselten, mobilen Sprachkommunikation auf Basis des Android Betriebssystems, sind zentraler Bestandteil dieser Master-Thesis.

Private und abhörsichere Kommunikation lässt sich in einem direkten Gespräch ohne größeren Aufwand realisieren. Bei indirekter Sprachkommunikation, wie einem Telefongespräch, muss erheblich mehr Aufwand betrieben werden, um abhörsicher und vertraulich kommunizieren zu können. Die Verwendung von Verschlüsselungsmechanismen ist hierfür eine Option zur Realisierung von abhörsicherer und privater Sprachkommunikation.

In dieser Arbeit werden die verbreitetsten Protokolle und Technologien beschrieben und evaluiert, mit deren Hilfe man verschlüsselte Sprachkommunikation technisch realisieren kann. Die Technologie-Evaluation wird kategorisiert nach Netzzugangstechnologie, Audiocodec, Signalisierung, Medientransport und Schlüsselverwaltung durchgeführt. Dies geschieht unter Berücksichtigung von Angriffen und Sicherheitslücken, den Besonderheiten der Medientransportebene bei drahtloser mobiler Datenübertragung und den Beschränkungen, denen mobile Endgeräte und der mobile Internetzugang unterworfen sind.

Neben der Evaluation der Technologien wird auch der Einsatz und die Integration von Smartcards in Sprachkommunikationssoftware zur Verschlüsselung und Zertifikatsspeicherung diskutiert.

Aufbauend auf der Analyse der existenten Protokolle und Technologien wird das Konzept einer Anwendung entwickelt, mit der die verschlüsselte Sprachkommunikation auf dem Android Betriebssystem realisiert werden kann. Dabei wird durch die Implementierung einzelner Teile des Konzepts, eine Machbarkeitsstudie durchgeführt.

Inhaltsverzeichnis

Teil I

Einleitung

In der vorliegenden Master-Thesis werden die unterschiedlichen Möglichkeiten untersucht, wie man abhörsichere Sprachkommunikation unter Android realisieren kann. Die Arbeit ist in vier Teile strukturiert, die wiederum in einzelne Kapitel untergliedert sind.

In Teil I werden die Grundlagen erläutert, die notwendig sind um die vorliegende Arbeit besser zu verstehen und die getroffenen Entscheidungen nachvollziehen zu können. Zunächst wird die Motivation der Arbeit in Kapitel 1 dargelegt. Im darauf folgenden Kapitel 2 wird das Android Betriebssystem in Teilen vorgestellt, soweit es für das Verständnis der Implementierung notwendig ist. Im letzten Kapitel des Einleitungsteiles wird auf digitale Sprachkommunikation im Allgemeinen eingegangen und deren Prinzipien vorgestellt.

Teil II befasst sich mit der Evaluation von Technologien die für die Umsetzung von abhörsicherer Sprachkommunikation unter Android in Frage kommen. Zunächst wird ein Anforderungskatalog aufgestellt, anhand dessen die Technologien bewertet und verglichen werden (Kapitel 4). Nach der Evaluation der mobilen Netzzugangstechnologien (Kapitel 5) und der Begründung weshalb unter Android die GSM-Sprachdaten nicht direkt abgesichert werden können (Kapitel 6), werden zuerst Audiocodecs untersucht die sich zur Kodierung von Audiodaten eignen könnten (Kapitel 7). In Kapitel 8 werden Medientransportprotokolle untersucht um Sprachdaten zu transportieren, gefolgt von Kapitel 9 in dem Protokolle zur Signalisierung miteinander verglichen werden. Bedingt durch das ausgewählte Medientransportprotokoll werden in Kapitel 10 Protokolle evaluiert, mit deren Hilfe die Schlüsselverwaltung für das Medientransportprotokoll durchgeführt werden kann.
In Kapitel 11 wird in einer Zusammenfassung das Ergebnis der Evaluation aller Technologien dargestellt.

Teil III befasst sich mit dem Design (Kapitel 12) und der Implementierung (Kapitel 13) der Anwendung zur abhörsicheren Sprachkommunikation unter Android.

Im letzten Teil der Arbeit (Teil IV) wird nach einem Resümee (Kapitel 14), ein möglicher Ausblick auf zukünftige Erweiterungen gegeben (Kapitel 15).

Da in der vorliegen Master-Thesis auch der praktische Einsatz und die Relevanz von graphischen Modellierungssprachen in der Software- und Systementwicklung untersucht wurde, wird in Anhang A auf die Sprache User Requirements Notation und in Anhang B auf den Einsatz von SysML eingegangen und deren Verwendung in der Praxis bewertet.
Ergänzend werden in Anhang C kryptographische Grundlagen erläutert und in Anhang D der Mitschnitt eines Netzwerkprotokollanalysators von einem ZRTP-Handshake dargestellt.

1 Privatsphäre durch Verschlüsselung

In diesem Kapitel wird die zugrunde liegende Motivation dieser Arbeit erläutert. Es wird dargelegt weshalb die Verschlüsselung von Sprachkommunikation sinnvoll und in vielen Fällen sogar notwendig ist. Dabei wird auch auf die Schwächen des GSM-Sprachkanals und weshalb dieser trotz der Verwendung von Verschlüsselung nicht sicher ist, eingegangen.

1.1 Wahrung der Privatsphäre, Schutz von Geheimnissen

Die Notwendigkeit und der Wunsch zur privaten und persönlichen Kommunikation ist ein Bedürfnis, das jedem Menschen bekannt ist. Mehr noch, das Recht auf Privatsphäre gilt sogar als von den Vereinten Nationen festgelegtes Menschenrecht:

> "Niemand darf willkürlichen Eingriffen in sein Privatleben, seine Familie, seine Wohnung und seinen Schriftverkehr oder Beeinträchtigungen seiner Ehre und seines Rufes ausgesetzt werden. Jeder hat Anspruch auf rechtlichen Schutz gegen solche Eingriffe oder Beeinträchtigungen."[1]

Das Recht auf Privatsphäre ist neben der Deklarierung als Menschenrecht auch im deutschen Grundgesetz verankert:

> "Jeder hat das Recht auf die freie Entfaltung seiner Persönlichkeit, soweit er nicht die Rechte anderer verletzt und nicht gegen die verfassungsmäßige Ordnung oder das Sittengesetz verstößt."[2, Art. 2 Abs. 1]
>
> "Das Briefgeheimnis sowie das Post- und Fernmeldegeheimnis sind unverletzlich."[2, Art. 10 Abs. 1]

Die Wahrung der Vertraulichkeit setzt aber nicht nur das Recht auf Privatsphäre einer Einzelperson durch. Sie dient auch dem Schutz von Firmen- und Regierungsgeheimnissen.

In Zeiten von Indect[3, 4, 5, 6], Echelon[7] und anderen elektronischen und voll automatisierten Data-Mining- und Überwachungssystemen ist es bei weitem nicht mehr ausreichend sich auf sein Recht auf Privatsphäre zu berufen. Um sich vor dem

Abhören der Sprachkommunikationen durch diese Systeme vor privaten Angreifern und vor Regierungen zu schützen, ist proaktives Handeln erforderlich. Nur so kann man sein *Recht auf Privatsphäre* und der vertraulichen Telekommunikation wahren und durchsetzen.

Die Benutzung von Verschlüsselungsmechanismen ist dabei neben der Steganographie[8] eine Möglichkeit das Recht auf Privatsphäre umzusetzen.
Der Schutz von vertraulichen Daten durch Verschlüsselung ist für Unternehmen und Regierungen Alltag. Privatpersonen hingegen fehlt zudem meist das Bewusstsein für die Notwendigkeit von Verschlüsselung. Dabei ist Verschlüsselung insbesondere bei der Verwendung von Mobiltelefonen und der Internettelefonie besonders wichtig. Da diese im Gegensatz zur Festnetztelefonie mit weniger Aufwand auch von Privatpersonen abgehört werden können[9, 10, 11] und dies trotz verschlüsselter Datenübertragung auf der Luftschnittstelle. Festnetztelefonie kann von Regierungsbehörden noch leichter abgehört werden. Dank der rechtlichen Grundlagen, der *Lawful Interception*, müssen Netzbetreiber auf eigene Kosten für Regierungsbehörden technische Schnittstellen bieten um sich in laufende Gespräche einklinken zu können[12]. Diese Schnittstellen zur *Lawful Interception* sind von der ETSI[13] und IETF[14] spezifiziert. Im Jahr 2009 wurden in Deutschland 17.208 Überwachungsanordnungen[15] nach § 100a Strafprozessordnung[16] durchgeführt.

Durch die enorme Verbreitung von Mobiltelefonen[17, 18] und dem abnehmenden Markt für Festnetztelefonie[19] ist es daher naheliegend, die verschlüsselte Kommunikation in erster Linie für Mobiltelefone zu realisieren.

1.2 GSM und die Privatsphäre

Das Global System for Mobile Communications (GSM)[20] ermöglicht bereits die verschlüsselte Kommunikation auf der Funkstrecke. Die verschlüsselte Kommunikation wird vom GSM-Standard allerdings nur zwischen dem Benutzergerät (User Equipment - UE) und der Base Station (BS) spezifiziert, um ein einfaches abhören auf der Luftschnittstelle auszuschließen. Ende-zu-Ende Verschlüsselung bietet der GSM-Standard nicht.

Die beiden von GSM spezifizierten Algorithmen A5/1 und A5/2[21] bieten auf dem Weg der Luftschnittstelle neuerdings keinen ausreichenden Schutz. Beide Algorithmen können aufgrund von zahlreichen Schwachstellen in Echtzeit ohne Kenntnis des Schlüssels mit Hilfe von Rainbowtables entschlüsselt werden [9, 10, 11]. Rainbowtables sind Tabellen mit vorausberechneten kryptographischen Hashketten, mit deren Hilfe man auf Kosten des Speicherplatz schneller Passworthashes entschlüsseln kann. Gegen einen weiteren Algorithmus, bekannt unter dem Namen Kasumi, der

in A5/3 Verwendung findet, sind ebenfalls bereits Attacken veröffentlicht (Related Key Attack)[22, 23].

Aufgrund der Tatsache dass die GSM Verschlüsselung zwischen UE und Base Station unzureichend ist, sowie bedingt durch die fehlende Ende-zu-Ende Verschlüsselung, ist es dringend empfohlen, (*zusätzliche*) Verschlüsselung auf die Sprachdaten anzuwenden um die Privatsphäre und Vertraulichkeit der Kommunikation für Privatpersonen, Unternehmen und Regierungsmitglieder zu gewährleisten (weiterführende Informationen unter [24]).

2 Android OS

In diesem Kapitel werden die Konzept- und Implementierungsrelevanten Eigenschaften des Android Betriebssystems (*Android OS*) erläutert. Neben einer kurzen Historie und einem Architekturüberblick wird auch explizit auf die Sicherheitskonzepte des Android OS eingegangen. Zusätzlich wird begründet weshalb das Android Betriebssystem für die Konzepterstellung und Implementierung verwendet wird. Sofern nicht explizit angegeben beziehen sich alle Erläuterungen, die Implementierung und Bezüge zu Android, auf die aktuelle Version 2.3.4.

2.1 Geschichte und Begründung

Das Android Betriebssystem (Android OS) wurde von Google und den 34 Mitgliedern der Open Handset Alliance (OHA) am 05. November 2007 der Öffentlichkeit vorgestellt[25]. Die Gründungsmitglieder bestanden unter anderem aus HTC, Sony, Motorola, Nvidia, Intel, T-Mobile, der Broadcom Corporation und Texas Instruments.

Kurze Zeit später, am 12. November 2007 wurde das erste Android SDK in Version 1.0 der Öffentlichkeit vorgestellt[26]. Seit diesem Zeitpunkt ist die Anzahl der Mitglieder der Open Handset Alliance auf fast 90 Mitglieder angewachsen (*Stand: 15.09.2011*) und Android hat in dem Zeitraum von 3,5 Jahren eine weite Verbreitung gefunden. Wie in Abbildung 2.1 dargestellt konnten im vierten Quartal 2010 mit 33,3 Millionen Geräten erstmals mehr Android Geräte pro Quartal abgesetzt werden als Geräte von Nokia, die auf 31 Millionen verkaufte Geräte kamen[27].

Die Statistiken von Gartner für das Jahr 2011, sowie die Prognose bis 2015 zeigen ebenfalls, dass Android das verbreitetste mobile Betriebssystem ist und für die nahe Zukunft auch bleiben wird[18, 17, 28]. Das iPhone und damit auch dessen Betriebssystem iOS von Apple kann im gleichen Zeitraum zwar ebenfalls einen starken Wachstum vorweisen, die absoluten Zahlen sind allerdings um mehr als 50% geringer als die von Android.

OS vendor	Q4 2010 shipments (millions)	% share	Q4 2009 shipments (millions)	% share	Growth Q4'10/Q4'09
Total	101.2	100.0%	53.7	100.0%	88.6%
Google*	33.3	32.9%	4.7	8.7%	615.1%
Nokia	31.0	30.6%	23.9	44.4%	30.0%
Apple	16.2	16.0%	8.7	16.3%	85.9%
RIM	14.6	14.4%	10.7	20.0%	36.0%
Microsoft	3.1	3.1%	3.9	7.2%	-20.3%
Others	3.0	2.9%	1.8	3.4%	64.8%

*Note: The Google numbers in this table relate to Android, as well as the OMS and Tapas platform variants
Source: Canalys estimates, © Canalys 2011

Abbildung 2.1: Weltweiter Smartphone Markt, Verkaufsanteile Q4 2010 (Quelle: [27])

2.2 Systemarchitektur

Das Android Betriebssystem basiert, sehr vereinfacht dargestellt, auf einem modifizierten und für den Einsatz auf mobilen Endgeräten optimierten Linux Kernel der Serie 2.6 und einer von Google eigens entwickelten Java VM mit Namen Dalvik Virtual Machine (Dalvik VM). Die Optimierungen für mobile Endgeräte waren notwendig, da diese mit beschränkten Ressourcen, wie geringer Akkukapazität und langsamen Prozessoren auskommen müssen. Das Android Application Framework setzt auf die Dalvik VM und den Linux Kernel auf und bildet zusammen mit zusätzlichen Systembibliotheken die Laufzeitumgebung für Anwendungen unter Android.

Smartphones, die für den Betrieb mit Android entwickelt wurden, verfügen meist über zwei separate Prozessoren. Einen *Application Processor* für Anwendungen und Software und einen *Communication Processor* für GSM[29]. Sofern die Trennung nicht physisch durch separate Prozessoren oder Kerne realisiert wurde, laufen immer zwei separate Software-Stacks. Dieses Detail ist wichtig, da der GSM Telefonstack bei Android Geräten nicht im Quellcode vorliegt. Gepaart mit der Android Systemarchitektur bedeutet das, dass auf die GSM Sprachdaten nicht zugegriffen werden kann, was aber bei der späteren Evaluation von Bedeutung ist (siehe Kapitel 6).

Die sparsame Verwendung von Systemressourcen ist eines der zentralen Designgrundlagen des Android Systems. Dazu zählen neben der Minimierung des CPU- und Speicherverbrauchs auch die Minimierung der Peripherieverwendung um Strom zu sparen.
Das Konzept der Ressourcenschonung fängt bei den Linux Kernelmodifikationen und der eigens für Android entwickelten Dalvik Java VM an, erstreckt sich über Binder, einen Kerneltreiber zur sparsamen Interprozesskommunikation, bis hin zu Bionic, einer minimalen C-Bibliothek, die, unter Aufgabe der *vollständigen* Kompatibilität

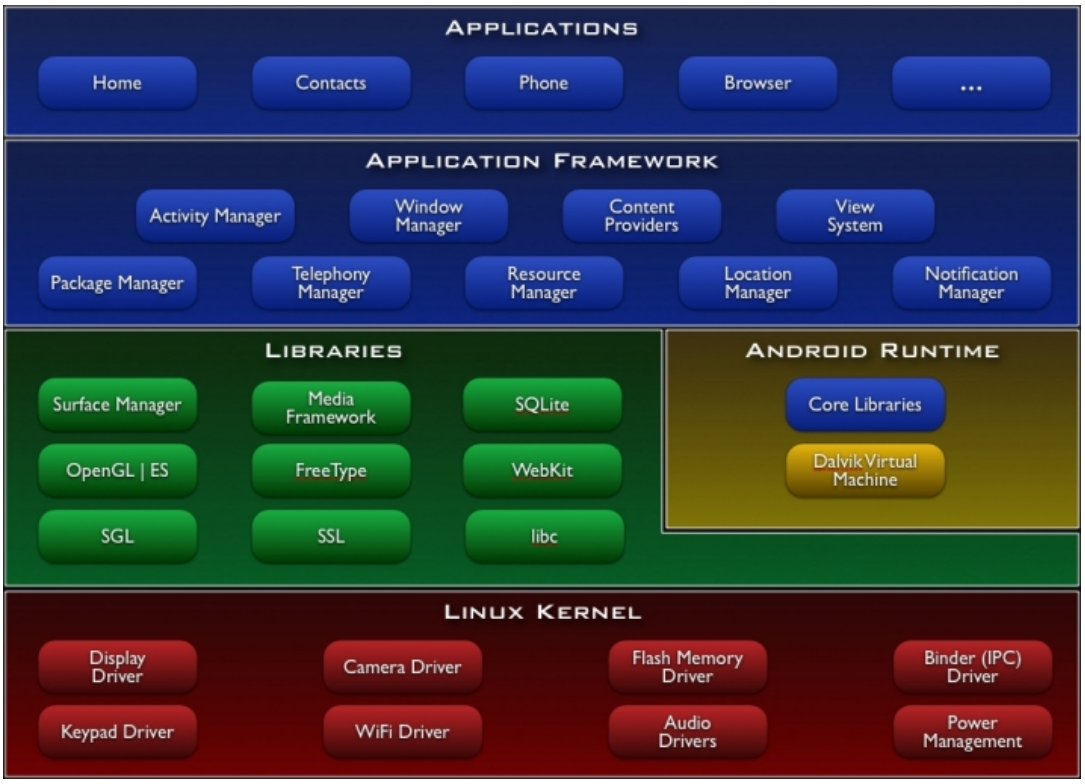

Abbildung 2.2: Android Systemarchitektur (Quelle: [31])

zur Glibc, weniger Systemressourcen verbraucht.
Bionic stellt dabei die meisten Funktionen der Glibc bereit. In Bionic ist eine eigens an Binder angepassten Posix-Threads[30] (Pthreads) Implementierung integriert. Bionic wird auch deshalb anstatt der Glibc verwendet, um nicht an deren Lizenz gebunden zu sein, die aufgrund der Copyleft-Klausel der GPL, lizenzrechtliche Probleme für Systementwickler verursachen könnte, sofern der Quellcode nicht veröffentlicht werden soll. Die Verwendung von Quellcode mit Copyleft-Lizenz in einem Binärprogramm erfordert die Veröffentlichung des Quellcodes des Binärprogramms bei dessen Verbreitung. Die Android Systemarchitektur ist in Abbildung 2.2 dargestellt.

Android Anwendungen können in den Programmiersprachen Java, C und C++ entwickelt werden. Java Anwendungen werden mit Hilfe des Android Software Development Kits (SDK) entwickelt. C und C++ Anwendungen werden mit Hilfe des Android Native Development Kits (NDK) realisiert. Mittels der Java Technologie Java Native Interface (JNI) können C und C++ Funktionen von Java aus aufgerufen werden und vice versa. Seit Android NDK Revision 5 vom Dezember 2010, können zudem Anwendungen, einschließlich der graphischen Benutzeroberflächen und deren Steuerung, auch ausschließlich in C oder C++ programmiert werden.

2.3 Sicherheit, IPC und Speichermanagement

Android Anwendungen laufen alle in einer eigenen Sandbox ab, einem abgeschotteten Speicherbereich im System, auf den nur die Anwendung selbst oder andere explizit berechtigte Anwendungen zugreifen können. Anwendungen agieren in einer eigenen Dalvik VM pro Applikation, verwenden einen eigenen Linux-Benutzeraccount und speichern die anwendungsrelevanten Daten in Ordnern, auf die nur der Benutzeraccount der Anwendung Zugriff hat. Dadurch werden alle Anwendungen strikt voneinander abgeschottet.

Um die Interprozesskommunikation (IPC) unter Android ressourcenschonend umzusetzen tauscht der IPC Kerneltreiber *Binder* nur Referenzen auf Objekte aus, die in einem Shared Memory Bereich abgelegt sind. Dadurch können zwei Anwendungen miteinander kommunizieren, ohne die Daten zu kopieren. Um die Zugriffsberechtigungen kümmert sich dabei ebenfalls ein spezieller Kerneltreiber mit Namen Ashmem.

Zu den Aufgaben von Ashmem kommt, neben der Rechteverwaltung beim Zugriff auf Speicherbereiche, noch eine weitere Funktion hinzu. Bei knapp werdendem Speicher beendet Ashmen Programme, die zwar noch geladen sind, aber schon länger nicht mehr vom Benutzer in den Vordergrund geladen wurden automatisch. Diese Funktion ist unbedingt erforderlich, da Android zwar eine virtuelle Speicherverwaltung nutzt um den physischen Speicher zu verwalten, allerdings keinen Swappingbereich zusätzlich zum Arbeitsspeicher bietet. Daher müssen im Fall von akutem Arbeitsspeichermangel notwendigerweise wenig oder ungenutzte Anwendungen beendet werden.[32]

2.4 Offenheit

Android ist ein offenes Betriebssystem und wurde von Beginn an unter Berücksichtigung dieses Kriteriums entwickelt. Offenheit bedeutet nicht nur, dass das System quelloffen und unter einer liberalen BSD-Lizenz[33] veröffentlicht ist, es bedeutet auch, dass man als Entwickler vollen Zugriff auf die Systemfunktionen und Systemanwendungen hat. Dies umfasst den vollständigen Quellcode des Android OS. Dadurch ist es unter Android möglich, das mitgelieferte Adressbuch oder die Telefonapplikation durch eigene Anwendungen zu ersetzen und diese nahtlos ins System zu integrieren. Selbst Modifikationen an Systemkomponenten des Betriebssystems sind möglich, ohne bei der Verbreitung der Software den Quellcode mitzuliefern. Die einzige Prämisse ist, dass der Copyright Vermerk im Quellcode erhalten bleibt.

Die Offenheit von Android ist nicht auch zuletzt einer der Grundsteine für den Er-

folg der Plattform Android. Durch den vollständigen Zugriff auf das System und den Quellcode ist sie für Gerätehersteller und Distributoren sehr attraktiv, da sie mit Android auf eine solide und potente Plattform aufsetzen können und eine vollständige Anpassung des Betriebssystems an die eigenen Erfordernisse möglich ist.

Die Verfügbarkeit des Quellcodes des Android Betriebssystems bedeutet für die Anwendung zur sicheren Sprachkommunikation, dass man sie anstatt als Android Anwendung, auch das Android System direkt modifizieren kann. Die Verwendung des SCTP-Protokolls[34], um ein Beispiel zu nennen, könnte dadurch mit Hilfe eines Kernelmoduls realisiert werden oder einzelne Aufgaben einer Anwendung könnte man durch die Modifikation und Entwicklung von Android-Systembibliotheken realisieren.

Die Implementierung als Android Systemmodifikation hat allerdings den entscheidenden Nachteil, dass die Anwendung als Android Systemimage für jedes Gerät separat gepackt und konfiguriert werden müsste. Dies bringt einen enormen Implementierungs- und Verwaltungsaufwand mit sich, der proportional zur Anzahl der unterstützten Android-Geräte steigt.

Die Konzeption und Implementierung der verschlüsselten Sprachkommunikation als Android Systemmodifikation scheidet daher aus den möglichen Konzeptideen aus.

2.5 Fazit

Bedingt durch das starke Marktwachstum und die Marktdurchdringung des Android OS sowie die Offenheit des Systems, wird das in dieser Master-Thesis erstellte Konzept, auf Basis des Android OS erstellt.

Der aktuelle Android Release für Mobiltelefone ist Android 2.3.4 (*Codename Gingerbread*), für Android Tabletgeräte ist die aktuelle Version Android 3.2 (*Codename Honeycomb*) (Stand 01.09.2011). Da die Anwendungen zur sicheren Sprachkommunikation Mobiltelefone als Zielplattform hat und Android 3.2 abwärtskompatibel zu Android 2.3.4 ist, wird die Evaluation, das Konzept und die Implementierung für das Android Release 2.3.4 (Android SDK API Level 10) entwickelt.

3 Digitale Sprachkommunikation

In diesem Kapitel wird zunächst der allgemeine Ablauf von analoger und digitaler Sprachkommunikation erläutert. Die Digitale Sprachkommunikation wird dabei in unterschiedliche Phasen untergliedert aus denen sich die, für die spätere Evaluation und Konzeption notwendige Modularisierung, ergibt.

3.1 Telekommunikation und deren Ablauf

Der Ablauf analoger Sprachkommunikation im Public Switched Telephone Network (PSTN) lässt sich in drei Phasen untergliedern (Vgl. [35, S. 9 f]) und ist in Abbildung 3.1 dargestellt:

Verbindungsaufbau Signalisierung des Kommunikationswunsches zum Kommunikationspartner und Aufbau der Verbindung.

Nachrichtenaustausch Austausch von analog kodierten Sprachsignalen.

Verbindungsabbau Beenden des Austausches von Sprachsignalen und Freigabe der verwendeten Ressourcen.

Diese Untergliederung in drei Phasen lässt sich auf den Ablauf digitaler Sprachkommunikation ebenfalls anwenden, wie sie beispielsweise beim ISDN[37, 35, S. 391 ff] (Integrated Services Digital Network) oder auch bei der Sprachkommunikation im GSM-Netz abläuft. ISDN wird in diesem Kapitel stellvertretend für ein digitales Telekommunikationsnetz behandelt.
Betrachtet man analoge und digitale Sprachkommunikation als Applikation, dann unterscheiden sich die beiden nur in der unterschiedlichen Implementierung der drei Phasen. Sowohl das analoge Telefonnetz als auch ISDN operieren leitungsorientiert.

Im digitalen Telekommunikationsnetz ISDN werden die Übertragung von Signalisierungsdaten (ISDN B-Kanal) und den Sprachdaten (ISDN D-Kanal und Signalling System #7[38]) im Gegensatz zum PSTN, getrennt voneinander übertragen. Die Trennung von Signalisierungskanal und Datenkanal wird als *Outband-Signaling* bezeichnet. Dem gegenübergestellt ist *Inband-Signaling,* bei dem dem Signalisierungs- und Nutzdaten über den gleichen Kanal übertragen werden.

Digitale Sprachkommunikation kann neben dem leitungsorientierten ISDN auch über paketvermittelnde Netze transportiert werden. Die verbreitetste Realisierung

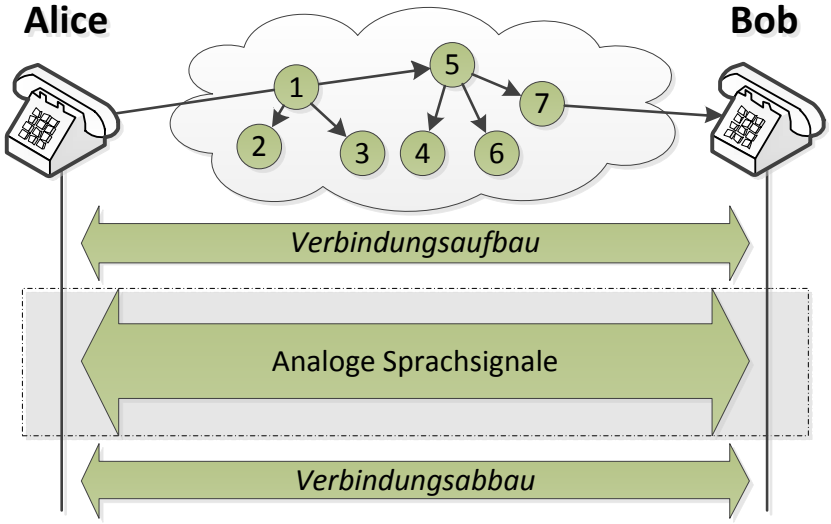

Abbildung 3.1: Analoge Sprachkommunikation (Vgl. [36])

von digitaler Sprachkommunikation über ein paketvermittelndes Netz ist Voice over IP (VoIP). Betrachtet man H.323[35, 39, 1082 ff], einen VoIP-Standard der ITU-T, sieht man, dass auch bei VoIP Sprachdaten und Signalisierungsdaten getrennt voneinander übertragen werden. An dieser Stelle muss darauf hingewiesen werden, dass auch VoIP-Lösungen existieren, die Inband-Signaling erlauben, wie beispielsweise das Inter-Asterisk eXchange Version 2 Protokoll[40], das allerdings nicht sehr verbreitet ist.

Da die Paketweiterleitung in IP-Netzen in der Regel nach dem *Best Effort*- Prinzip erfolgt, werden zusätzliche Mechanismen zur Messung und Steuerung des Datentransports nötig (Medientransportkontrolle). Die Digitale Sprachkommunikation über paketvermittelnde Netze ist in Abbildung 3.2 dargestellt.

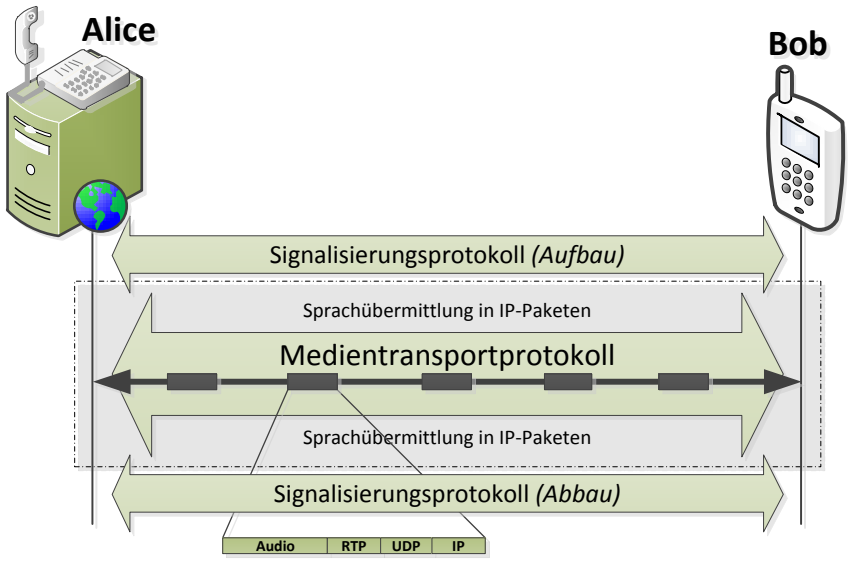

Abbildung 3.2: Digitale Sprachkommunikation in Paketnetzen (Vgl. [36])

Basierend auf den Eigenschaften von paketvermittelnden Netzen, lassen sich die aufeinanderfolgenden Phasen der digitalen Sprachkommunikation noch detaillierter untergliedern:

Verbindungsaufbau

Signalisierung Signalisieren des Kommunikationswunsches an den Kommunikationspartner.

Quittierung Bestätigung des Verbindungswunsches, beziehungsweise Bestätigung des Kommunikationswunsches durch den Empfänger.

Sprachkommunikation Die Subphasen der Sprachkommunikation laufen alle kontinuierlich und parallel ab.

Verbindungsaufbau Aufbau der Datenverbindung, beziehungsweise des Sprachkanals.

Sprachkodierung Umwandlung der Sprache in digitale Signale.

Steuerung und Kontrolle Überwachung und Kontrolle des Datenstroms.

Verbindungsabbau Abbau des Sprachkanals, bzw. der Datenverbindung.

3.2 Die Ebenen der digitalen Sprachkommunikation

Digitale Sprachkommunikation ist Netzwerkkommunikation mit Echtzeitanforderungen. Aus dem vorgestellten detaillierten Ablauf der digitalen Sprachkommunikation lassen sich die folgenden Ebenen einer Anwendung zur digitalen Sprachkommunikation ableiten. Die Liste wurde um den Punkt Benutzerinteraktion ergänzt,

da es für einen Benutzer notwendig ist mit dem User Equipment (UE) zu interagieren. Als UE wird dabei das Gerät oder die Software bezeichnet mit der der jeweilige Benutzer bei einer Ende-zu-Ende Sprachverbindung die Kommunikation durchführt.

Die vorgestellten Phasen werden getrennt voneinander evaluiert. Dennoch stellt die strikte Trennung in der späteren Implementierung keine zwingende Notwendigkeit dar.

Signalisierung Signalisierungsprotokoll um einem Kommunikationspartner den Verbindungswunsch mitzuteilen. Die Signalisierung ist Voraussetzung für den Aufbau eines (virtuellen) Sprachkanals, also den Aufbau dem Medientransportstroms, dessen Steuerung und für den Abbau der Verbindung.

Medientransport Transportprotokoll, um die digitalisierten Sprachdaten zwischen beiden Kommunikationsendpunkten auszutauschen.

Medientransportkontrolle Steuerungsprotokoll, um den Datenfluss und den Medientransport zu steuern. Mittels der Medientransportkontrolle kann auch die Einhaltung von QoS-Parametern überwacht werden.

Sprachkodierung Umwandlung und Kodierung von analoger Sprache in digitale Daten um den Transport in paketvermittelnden Netzen zu ermöglichen.

Benutzerinteraktion Graphische Benutzeroberfläche oder sonstiges Benutzerinterface, das es dem Benutzer ermöglicht mit der Anwendung zu interagieren und diese zu kontrollieren.

Bei dieser Betrachtung der digitalen Sprachkommunikation wurde noch nicht auf die Sicherheitsaspekte eingegangen. Sie stellt nur das notwendige Vorwissen und die Gliederung dar, in deren Rahmen die Evaluation durchgeführt wird. Sicherheitsaspekte bei der digitalen Sprachkommunikation sind Anforderungen, die an die Anwendung zur digitalen Sprachkommunikation gestellt werden. Auf die Sicherheitsaspekte wird daher im folgenden Kapitel 4.2 eingegangen, in dem die allgemeinen Anforderungen und die sicherheitsbezogenen Anforderungen an digitale Sprachkommunikation explizit dargelegt werden.

Teil II

Evaluation

Wie in Kapitel 3.2 ausgearbeitet, werden für die Umsetzung der digitalen Sprachkommunikation auf dem Android Betriebssystem Möglichkeiten und die Notwendigkeit evaluiert, um Abhörsicherheit und Verschlüsselung in folgenden Bereichen zu realisieren:

- Signalisierung
- Medientransport
- Medientransportkontrolle
- Sprachkodierung

Die Komponente des Benutzerinterfaces wird nicht evaluiert, da Android und das Android SDK eine geeignete Möglichkeit bieten das Benutzerinterface zu realisieren und das Benutzerinterface in dieser Arbeit keinen direkten Einfluss auf die Abhörsicherheit der Sprachkommunikation hat.

In Kapitel 4 wird ein Anforderungskatalog aufgestellt, auf dessen Grundlage in den darauf folgenden Kapiteln die unterschiedlichen Protokolle evaluiert werden. Dies erfolgt kategorisiert nach den Ebenen der digitalen Sprachkommunikation die in Kapitel 3.2 definiert wurden.

4 Anforderungskatalog

Digitale Sprachkommunikation stellt besondere Anforderungen an die Protokolle und Technologien, die für den Transport der Sprachdaten verwendet werden. Ebenso werden durch die Aspekte der Abhörsicherheit und der Verschlüsselung der Sprachkommunikation ganz eigene Anforderungen gestellt, deren Einhaltung durch die evaluierten Technologien gewährleistet werden muss. Diese Anforderungen werden in diesem Kapitel definiert.

Die Evaluation der Möglichkeiten zur praktischen Umsetzung der verschlüsselten Sprachkommunikation und die Bewertung der differenten Optionen, geschieht in den späteren Kapiteln im Kontext der in diesem Kapitel definierten Anforderungen und Bewertungskriterien.

Die Grafische Anforderungsanforderungsanalyse (dargestellt unter Abbildung 4.1) wurde mit Hilfe der Modellierungssprache User Requirements Notation (URN) erstellt. Weiterführende Information zur Verwendung von URN können den Spezifikationen Z.150[41] und Z.151[42] der *International Telecommunication Union* (ITU) entnommen werden. Ein Erfahrungsbericht zur Verwendung von URN befindet sich im Anhang A.

4.1 Anforderungen an die Sprachkommunikation

Aufgrund der Echtzeitanforderungen an digitale Sprachkommunikation, ist die Paketlaufzeit, auch als Ende-zu-Ende Verzögerung (Delay) bezeichnet, das schwerwiegendste Bewertungskriterium. Daneben beeinflussen noch weitere QoS-Parameter die Sprachqualität:
Dazu zählen neben der *Bandbreite* der Ende-zu-Ende-Verbindung auch die Schwankung in der Paketübertragungszeit (*Jitter*) sowie die Paketverlustrate (*Packet Loss Rate*) (Vgl. [36, S. 100]).

Laut dem Standard G.114[43] der ITU, liegt die maximale Obergrenze des Delays für Sprachkommunikation bei der noch ein befriedigendes Benutzererlebnis garantiert werden kann, bei 400 ms. Um hervorzuheben welcher Bereich mit dem Delay gemessen wird, wird dieses Delay auch als *Ein-Wege Delay* oder noch deskriptiver, als *mouth-to-ear Delay* bezeichnet. Damit wird die Zeitdauer bezeichnet, die

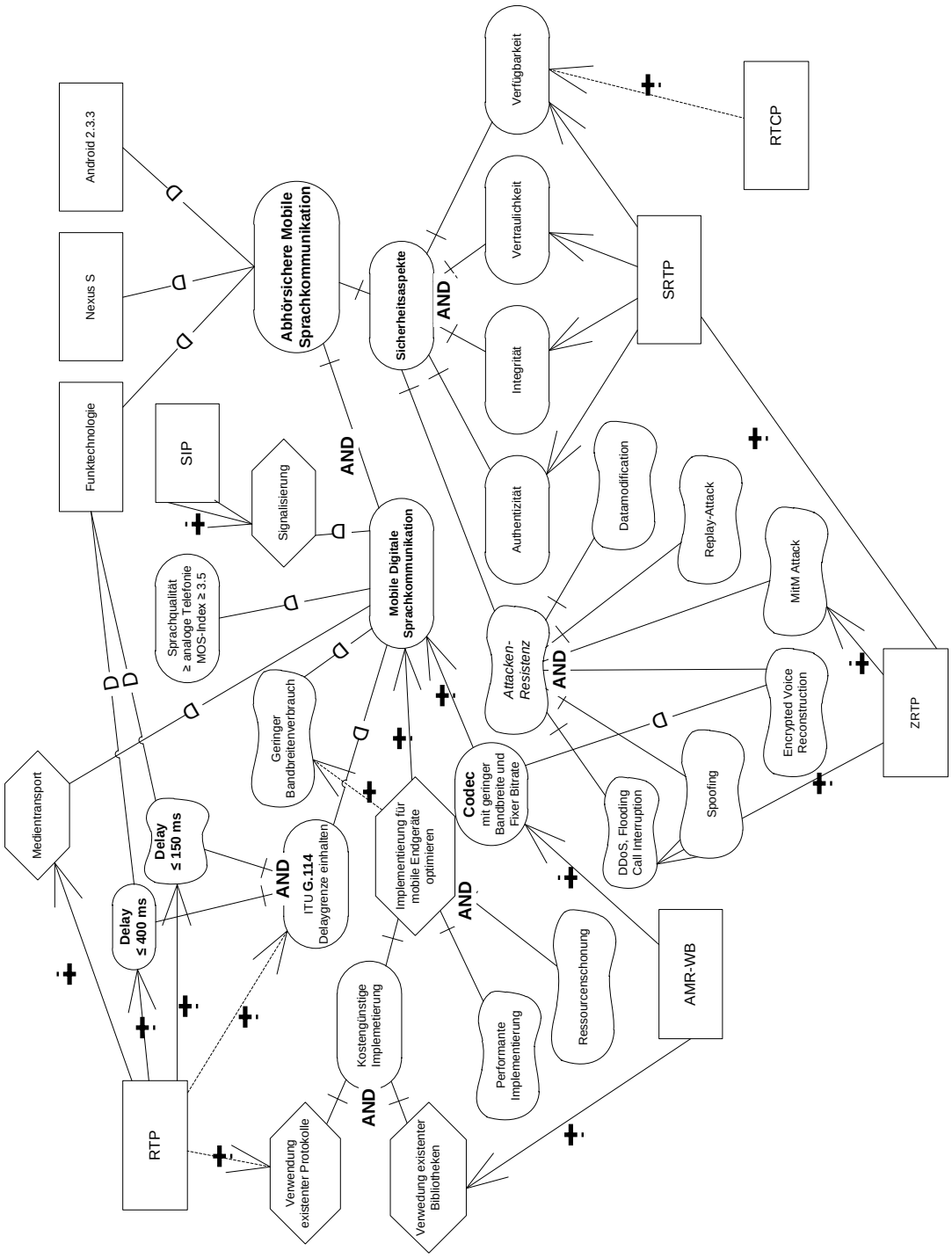

Abbildung 4.1: Anforderungsanalyse mit der User Requirements Notation (Goal-oriented Requirements Language)

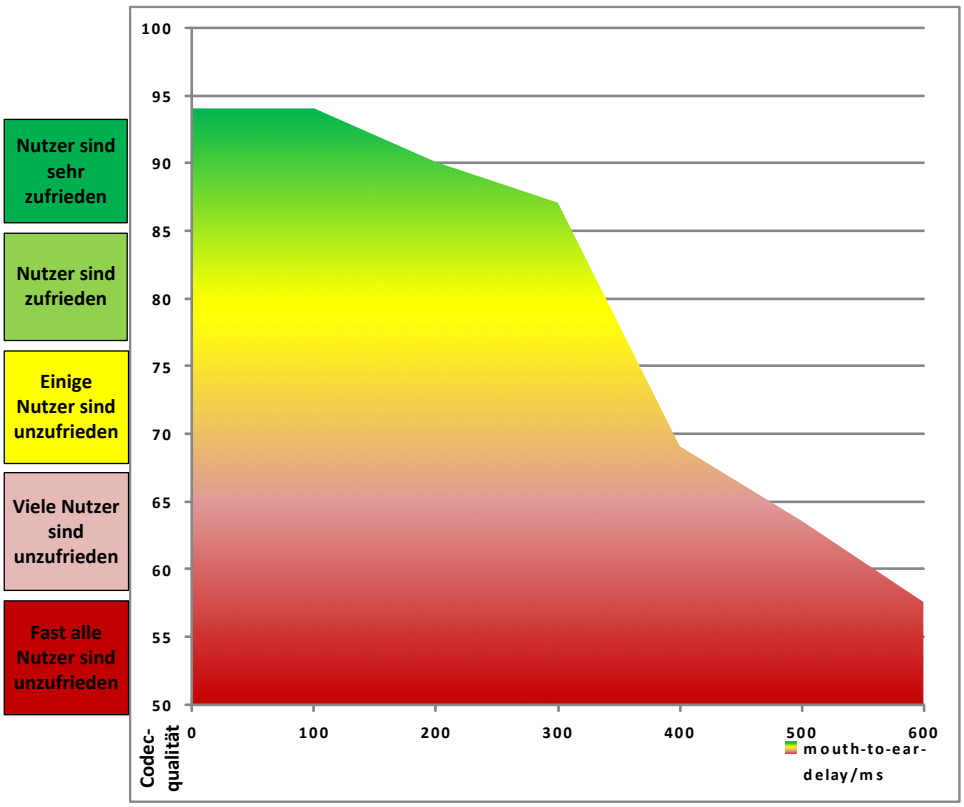

Abbildung 4.2: ITU-T G.114 Zusammenhang zwischen Ein-Wege Delay, Qualität der Sprachkodierung und Benutzerzufriedenheit (Vgl. [43])

ein Sprachsignal benötigt, um zum Ohr des Kommunikationspartners zu gelangen wenn es ausgesprochen wurde. Bei 400 ms Delay ist die Verzögerung bereits deutlich hörbar und es tritt eine Beeinträchtigung der Kommunikation und der Benutzerzufriedenheit in Kraft.
Um eine optimale Benutzerzufriedenheit zu gewährleisten sollte ein Delay von 150 ms nicht überschritten werden. Nichtsdestotrotz ist Sprachkommunikation auch mit einem Delay von 150 ms bis 400 ms möglich. Aus technischer Sicht ist Sprachkommunikation auch bei einem mouth-to-ear Delay von mehr als 400 ms noch möglich. Die Sprachkommunikation erhält dadurch allerdings eher den Charakter eines Funkgespräches.

Der Zusammenhang zwischen Paketverlustrate und dem Ein-Wege-Delay und der daraus resultierenden Benutzerzufriedenheit ist in Abb. 4.2 dargestellt.

Von der ITU-T wurden mehrere Verfahren spezifiziert, um die Qualität von Sprachkommunikation nach der Übertragung zu messen. Ein älteres Verfahren ist der Mean Opinion Score (MOS) der unter [44] standardisiert ist. Der MOS verwendet dabei eine Skala von 1-5 um die Qualität auszudrücken. Ein weiteres von der ITU-T definiertes Verfahren ist das MUltiple Stimuli with Hidden Reference and Anchor

(MUSHRA)[45] das im Gegensatz zu MOS nicht nur für die Bewertung von entfernter Kommunikation entwickelt wurde, sondern zur Bewertung von Audio Qualität im Allgemeinen. MUSHRA kommt zudem auch mit weniger Testsubjekten aus, als das MOS-Verfahren, um eine valide Bewertung der Audioqualität zu ermöglichen.

Abschließend betrachtet findet eine Bewertung der Technologien basierend auf folgenden Parametern statt:

Delay Ein Delay von $\leq$150 ms gilt als Optimum, 150 ms$-$400 ms als noch akzeptables Delay, >400 ms als technisch möglich unter starker bis sehr starker Beeinträchtigung des Spracherlebnisses.

Bandbreite Basierend auf dem verwendeten Codec und der gewünschten Sprachqualität sowie den zusätzlichen Verwaltungsdaten durch das Transportprotokoll, werden unterschiedliche Bandbreiten benötigt. Abhängig von den verwendeten Netztechnologien werden unterschiedliche Bandbreiten zur Verfügung gestellt.

Dies bedeutet für die Evaluation, dass der verwendete Codec und der Protokolloverhead des Medientransportprotokolls die minimal notwendige Bandbreite vorgeben. Als Referenz kann der von der ITU spezifizierte Codec G.711 angesehen werden, der ohne die Verwaltungsdaten des Transportprotokolls eine Bandbreite von 64 kbit benötigt[46].

Jitter Der auftretende Jitter sollte minimiert werden. Dies kann durch spezielle Steuerprotokolle oder durch einen Datenpuffer an der Empfängerseite umgesetzt werden. Der Puffer sollte möglichst klein sein, da andernfalls der Delay erhöht wird. Der auftretende Jitter wird allerdings durch das Transportnetz verursacht, auf das man nur begrenzten Einfluss hat. Maßnahmen zur Minimierung des Jitters sind dadurch nur bis zu einer bestimmten Grenze wirksam.

Paketverlust Überwachung und Reaktion auf die Paketverlustrate oder Übertragungsfehler durch die verwendeten Protokolle. Durch externe Mechanismen aus dem Bereich des Quality of Service kann auch auf den Paketverlust Einfluss genommen werden. Quality of Service Maßnahmen sind allerdings nicht Teil dieser Arbeit. Weitere Erläuterungen zu QoS Parametern können in [47, 48] und [36, S. 99 ff] eingesehen werden.

4.2 Anforderung an die sichere Sprachkommunikation

Um die Evaluation von Technologien für eine sichere Sprachkommunikation durchführen zu können, werden in diesem Kapitel die Anforderungen an sichere Sprachkommunikation definiert. Sicherheitsmechanismen wie Verschlüsselung sollten dabei auf dem gesamten Kommunikationsweg realisiert werden (*Ende-zu-Ende Sicherheit*).

Sichere Sprachkommunikation erfordert laut [49, 50]:

Vertraulichkeit Die Sprachkommunikation muss Ende-zu-Ende verschlüsselt sein und darf nicht abgehört werden können. Die Ende-zu-Ende Verschlüsselung ist notwendig, da dadurch die autorisierten Entschlüsselungsmöglichkeiten technisch auf die beiden Kommunikationspartner begrenzt ist und nicht durch Angriffe auf Infrastruktur innerhalb der Kommunikationskette eine Kommunikation belauscht werden kann.

Integrität der Sprachdaten. Sprachdaten dürfen nicht unbemerkt verändert werden. Wenn Sprachdaten verändert wurden muss dies zweifelsfrei nachweisbar sein.

Verfügbarkeit Die Kommunikation darf nicht durch eine nicht-autorisierte Drittperson gestört oder verhindert werden. Da dies zu gewährleisten in einem dezentralen und öffentlichen Netzwerk wie dem Internet nicht immer möglich ist, werden die Technologien im Hinblick auf eine Minimierung der Angriffsmöglichkeiten hin untersucht und nicht auf deren Eliminierung.

Authentizität Gewährleistung der Authentizität der Sprachdaten und der Kommunikationsteilnehmer, damit keine Drittpersonen unter Vorgabe einer falschen Identität Daten an die Kommunikationsteilnehmer versenden können. Kurz gesagt, als Schutz vor Man-in-the-Middle Angriffen.

Zusätzlich sollte sichere Sprachkommunikation auch einen Schutz vor bereits bekannten Attacken bieten, oder zumindest die Angriffsmöglichkeit einschränken sofern eine Eliminierung des Angriffsvektors nicht möglich sein sollte. Die Technologien werden auf Resistenz, beziehungsweise auf die Abschwächung folgender Attacken und Angriffsszenarios hin evaluiert (Vgl. [51]):

DoS Attacke Die Denial of Service Attacke ist ein Angriff auf einen Server oder einen Peer im allgemeinen, um dessen Verfügbarkeit zu beeinträchtigen dessen Erreichbarkeit vollständig zu unterbinden. DDoS steht für Distributed Denial of Service und ist mit der DoS Attacke identisch, mit dem Unterschied, dass der Angriff von zahlreichen Peers gleichzeitig ausgeht. Spam over Internet Telephony oder kurz SPIT kann zu der gleichen Kategorie gerechnet werden, da Werbeanrufe auch die Verfügbarkeit eines Peers einschränken. Call Interruption, als die Unterbrechung eines Anrufes, fällt als vierter Angriff ebenfalls in die Kategorie der DoS Angriffe.

Replay Attacke Replay Attacken sind Angriffe bei denen aufgezeichnete Netzwerkpakete erneut versendet werden, häufig auch in Kombination mit Veränderungen an den erneut versendeten Paketen.

Spoofing Als Spoofing Attacke werden Angriffe bezeichnet bei denen der Angreifer versucht unter der Vorgabe falscher Tatsachen oder Daten einen Angriff durchzuführen. Spoofing kann durch Sicherung der Authentizität eliminiert

werden.

Eavesdropping Eavesdropping Attacken bezeichnen Angriffe bei denen Angreifer Kommunikation belauschen, um an Informationen zu gelangen.

Data Modification Durch Data Modification Attacken werden vom Angreifer Daten in der Kommunikation zweier Peers verändert, um den Kommunikationspartnern Informationen zu verheimlichen oder falsche Informationen zu vermitteln.

Man-in-the-Middle Als Man-in-the-Middle Angriffe werde Angriffe bezeichnet, bei denen sich der Angreifer in die Kommunikation zweier Peers einklinkt. Dies geschieht häufig beim Verbindungsaufbau, so dass die gesamte Kommunikation zweier Peers über die dritte Person, also den Angreifer abgewickelt wird.

Encrypted Voice Reconstruction Diese Attacke bezeichnet Angriffe, die bei verschlüsselter Sprachkommunikation den Inhalt von Gesprächen reproduzieren können. Möglich wird das aufgrund der variablen Paketgröße beim Medientransport wenn man einen Audiocodec mit variabler Bitrate verwendet.

Sidechannel Attacken bezeichnen alle Angriffe die aufgrund der physischen Implementierung eines kryptographischen Algorithmus durchgeführt werden. Dazu zählen unter anderem Angriffe, die durch das messen des Stromverbrauchs eines Gerätes zur Laufzeit des kryptografischen Algorithmus wirksam sind (*Differential Power Analysis*)[52] oder durch messen der Laufzeit von kryptografischen Algorithmen wirksam sind (*Timing Attack*)[53].

Unter Tabelle 4.1 ist eine Übersicht über die behandelten Attacken zusammengestellt, einschließlich einer Kategorisierung nach den Sicherheitskriterien aus Kapitel 4.2.

Ein zusätzliches kryptographisches Verfahren dessen Anwendung wünschenswert ist, ist *Perfect Forward Secrecy*. Mit Hilfe dieses Verfahrens wird gewährleistet, dass durch die Kompromittierung von kryptographischen Schlüsseln nicht alle Daten entschlüsselt werden können, die damit einmal verschlüsselt wurden. Es können maximal die Kommunikationsdaten der letzten Kommunikationssitzung entschlüsselt werden. Beispielprotokolle, die dieses Verfahren verwenden sind beispielsweise das Secure Shell Authentication Protokoll[55] oder das Protokoll Off-the-Record-Messaging[56].

Zur Benennung der Teilnehmer einer Kommunikation oder eines Schlüsselaustausches werden die Namen Alice und Bob verwendet[57].

Attacke	Vertraulichkeit	Integrität	Authentizität	Verfügbarkeit
DOS, DDoS, Flooding, SPIT, Call Interruption				⊗
Replay Attacke	⊗	⊗	⊗	⊗
Spoofing	⊗	⊗	⊗	⊗
Eavesdropping (Belauschen), Sniffing	⊗		⊗	
Data Modification		⊗		
Man-in-the-Middle Attacke (MitM), Call Hijacking	⊗	⊗	⊗	⊗
Encrypted Voice Reconstruction[54]	⊗			
Schwachstellen bei CA	⊗	⊗	⊗	
Sidechannel Attacken	⊗	⊗	⊗	

Tabelle 4.1: Attacken und ihr Einfluss auf Sicherheitsbereiche

4.3 Allgemeine Anforderungen an Technologien und Implementierung

Neben den in Kapitel 4.1 und Kapitel 4.2 definierten speziellen Anforderungen, existieren zusätzlich noch allgemeine Anforderungen, die ebenfalls bei der Bewertung mit einbezogen werden.
Zu den allgemeinen Anforderungen gehören die Komplexität der Umsetzung, beziehungsweise der Implementierung des Protokolls oder der Technologie. In die gleiche Kategorie fallen die Kosten, die durch eine bestimmte Technologie verursacht werden sowie der zeitliche Aufwand für die Umsetzung des Konzepts.

Bei der Implementierung der Technologien sollte zusätzlich noch besonderes Augenmerk auf geringen Ressourcenverbrauch und Performance gelegt werden, da ein mobiles Endgerät nur über beschränkte Ressourcen verfügt.

Die Anwendung sollte, wenn möglich, mit keiner oder nur geringer unterstützender Infrastruktur auskommen. Dies senkt zum einen die Kosten und den Aufwand für den Einsatz der Anwendung, zum anderen wird dadurch die Anzahl der möglichen Angriffsvektoren minimiert.

Die allgemeinen Anforderungen an die Anwendung zur abhörsicheren Sprachkommunikation umfassen folgende Punkte:

- Komplexität, Kosten und zeitliche Dauer der Umsetzung
- Keine oder nur sehr wenige unterstützende Infrastruktur als Voraussetzung für den Anwendungsbetrieb
- Einfache Integration in existierende Infrastruktur
- Performance auf dem Mobilgerät
- Ressourcenverbrauch

5 Netz-Technologie

Sprachkommunikation stellt besondere Anforderungen an das Transportmedium. Anders als beispielsweise Emails, das FTP-Protokoll oder das HTTP-Protokoll, stellt Sprachkommunikation sowohl an das Transportmedium, an die verwendeten Protokolle, als auch an die Gesamtperformance sowie alle Teile des Kommunikationssystems Echtzeitanforderungen (Dargestellt in Abb. 4.1).
Das Transportmedium hat hierbei kritischen Einfluss auf die Latenz und die maximal verfügbare Bandbreite bei mobiler Datenübertragung und setzt Obergrenzen für diese beiden Parameter.

Die Anforderungen beginnen bereits auf den Ebenen 1 und 2 des OSI Referenzmodells[58, 59]. Der Physical Layer und Data-Link Layer mobiler und drahtloser Übertragungstechnologien sind teilweise starken Beschränkungen in Bezug auf *Bandbreite* und *Kapazität* sowie die *Latenz (Delay)* unterworfen. Zu dieser Auflistung kommt noch hinzu, dass die Netztechnologien unterschiedlich weit verbreitet sind, sowohl in Deutschland, Europa, als auch weltweit betrachtet. Die in Deutschland verfügbaren Netztechnologien sind laut [60]:

CSD - Circuit Switched Data wird von Android – zumindest aktuell – nicht unterstützt[61, 62].

GPRS - General Packet Radio Service (2.5 G) wird von Android unterstützt und ist laut Angaben der Netzbetreiber die verbreitetste Netztechnologie[63, 64, 65, 66].

HSCSD - High Speed Circuit Switched Data wird von Android – zumindest aktuell – nicht unterstützt[61, 62].

EDGE - Enhanced Data Rates for GSM Evolution (2.75 G) wird von Android unterstützt. Die Abdeckung mit EDGE ist bei T-Mobile genauso gut wie mit GPRS, bei den übrigen Netzanbietern ist sie schlechter[65]. EDGE spezifiziert neue Modulationsarten und erhöht dadurch die Geschwindigkeit von GPRS und HSCSD um das dreifache. Durch die Möglichkeit der Kanalbündelung wird dies noch weiter gesteigert (Vgl.: [35]).

E-GPRS - Enhanced GPRS Ausdruck für die verbesserte GPRS Variante. Wird von Android unterstützt.

ECSD - Enhanced CSD Ausdruck für die verbesserte HSCSD Variante. Wird von Android – zumindest aktuell – nicht unterstützt[61, 62].

eEDGE - Evolved EDGE Weiter verbesserte Version von EDGE. Wird von Android unterstützt.

UMTS - Universal Mobile Telecommunications System (3 G) Netzabdeckung ist bei den Mobilfunkanbieter unterschiedlich stark. Android unterstützt UMTS.

HSPA High Speed Packet Access (3.5 G) wird unterschieden in
High Speed Downlink Packet Access (HSDPA) und
High Speed Uplink Packet Access (HSUPA) um klar zu stellen, dass unterschiedliche Datenraten im Up- und Downlink zur Verfügung stehen. VoIP über HSPA ist sehr gut möglich[67] und wird von Android unterstützt.

HSPA+ Evolved High Speed Packet Access Verbesserte Version von HSPA. HSPA+ wird von Android unterstützt.

LTE - Long Term Evolution Wird von Android unterstützt und ist ebenso wie HSPA und HSPA+ bestens für den Transport von Sprachdaten geeignet.

LTE+ - Long Term Evolution Advanced Wird von Android *nicht* unterstützt.

W-LAN / WIFI - Wireless LAN Wird von Android unterstützt. Für VoIP bestens geeignet, allerdings abhängig von der verwendeten Internetzugangstechnologie an die der W-LAN Access Point angeschlossen ist.

WiMAX - Worldwide Interoperability for Microwave Access Wird von Android unterstützt und ist für den zufriedenstellenden Transport von Sprachdaten geeignet.

Vergleicht man diese in Bezug auf Delay und Bandbreite wird deutlich, dass die Layer 1 und 2 Technologien sich teilweise stark voneinander unterscheiden (Tabelle 5.1).

Netztechnologie	GSM		UMTS			LTE	ADSL
	GPRS	EDGE	UMTS	HSPA	HSPA+	LTE	ADSL
Downlink	53,6 kbps	236,8 kbps	384 kbps	14,4 mbps	~42 mbps	150 mbps +	16 mbps
Uplink	26,8 kbps	236,8 kbps	384 kbps	5,8 mbps	~11 mbps	50 mbps +	1 mbps
Delay	500 ms +	300-400 ms	170-200 ms	60-70 ms	25-35 ms	10-20 ms	20-50 ms

Table 5.1: Netztechnologien - Latenz und Bandbreitenvergleich (Quelle: [60], [68] und [69])

Das Mouth-to-ear Delay sollte laut ITU-T G.114 400 ms nicht überschreiten (siehe Kapitel 4.1). Dennoch sind nahezu alle existierenden Netztechnologien für die digitale Sprachkommunikation geeignet. In Bezug zur Bandbreite und der Latenz

bildet nur GPRS eine Ausnahme. Telefonie ist zwar möglich, bietet aber ein unbefriedigendes Benutzererlebnis, da ein Telefonat via GPRS eher den Charakter eines Funkgespräches als eines Telefonats hat (siehe Abbildung 4.2).

GPRS bietet ein nur unbefriedigendes Ergebnis was den Delay betrifft, obwohl aus technischer Sicht auch bei einem Delay von 500ms Sprachkommunikation noch möglich ist.

EDGE bietet ein Delay innerhalb der ITU G.114 Grenzwerte und ermöglicht Kommunikation mit gerade noch akzeptabler Sprachqualität.

UMTS ermöglicht gute bis sehr gute Sprachqualität, da der Grenzwert von 150 ms fast eingehalten wird.

HSPA, HSPA+, LTE sowie der in Planung befindliche Standard LTE Advanced ermöglichen eine sehr gute Sprachkommunikation mit einer optimalen, da sehr geringen Verzögerung von $\leq$70 ms.

W-LAN, WiMax bieten, abhängig von der verwendeten Internetzugangstechnologie einen sehr guten Zugang zum Internet und ein geringes Delay. Geht man davon aus, dass mindestens die Performance eines ADSL Internetzugangs vorliegt, sind ebenfalls Latenzen $< 70ms$ möglich.

Da sich diese Arbeit auf die Konzeption von verschlüsselter Sprachkommunikation auf Basis des Android OS beschränkt, werden CSD, HSCSD und damit auch ECSD nicht näher betrachtet. Diese drei Technologien können unter Android nicht verwendet werden, da das Betriebssystem sie nicht unterstützt werden.

Vergleicht man die Netztechnologie in Bezug zur Bandbreite, sind alle Netztechnologien ausreichend dimensioniert. Die geringste Bandbreite bietet die Netztechnologie GPRS, die aufgrund des zu hohen Delays allerdings nicht zu empfehlen ist. Die geringe Bandbreite von 56 kbit/s ist bei der Verwendung eines (komprimierenden) Codecs mit geringem Bandbreitenbedarf bereits mehr als ausreichend und bietet dabei sogar bessere Sprachqualität als digitale Telefonie über ISDN mit dem G.711[46] Audiocodec (Siehe Kapitel 7).

Im Vergleichspunkt des Delays bieten alle Netz-Technologien außer GPRS ausreichende (150-400 ms) bis optimal ($\leq$150 ms) geeignete Werte. GPRS ist zwar technisch geeignet, bietet aber aufgrund des hohen Delays kein befriedigendes Spracherlebnis nach ITU G.114[43].

Die Verfügbarkeit der Netztechnologien ist stark vom jeweiligen Mobilfunkanbieter abhängig. Zur Zeit der Umfrage (Stand: 07. Juli 2010) liegt die Netzabdeckung vom UMTS und HSPA von Vodafone mittlerweile bei über 70%[70] und T-Mobile beispielsweise, stellt eine einhundertprozentige Versorgung mit EDGE in ihrem gesamten GSM-Netz zur Verfügung[65].

Basierend auf dieser Analyse lässt sich schließen, dass bezüglich der OSI-Layer 1

und 2 Technologien alle Netztechnologien außer GPRS bestens geeignet sind um Sprachdaten auf mobilen Endgeräten zu übertragen. Alle Technologien außer GPRS entsprechen dem Delay und der Bandbreite von EDGE oder sind sogar besser.

6 GSM Ende-zu-Ende Sicherheit

Als Alternative zur digitalen Sprachkommunikation über ein mobiles paketvermittelndes Datennetz, existiert auch die Möglichkeit verschlüsselte Sprache über den GSM Sprachkanal zu transportieren. Dies erscheint auf den ersten Blick als logische und einfache Möglichkeit, abhörsichere Kommunikation zu realisieren. Bei näherer Betrachtung des GSM-Sprachkanals werden allerdings die Herausforderungen dieses Verfahrens deutlich.

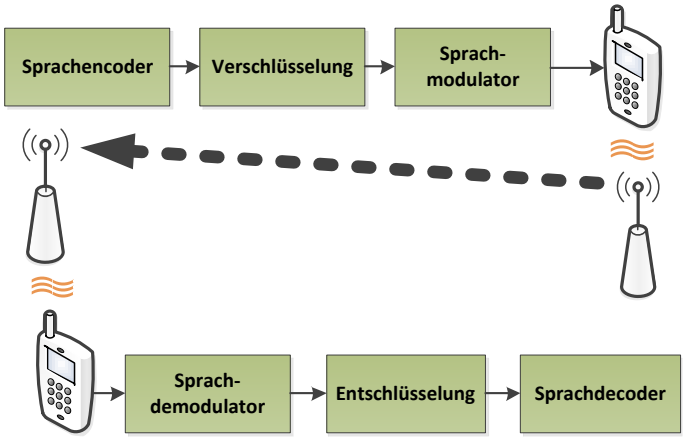

Abbildung 6.1: GSM Ende-zu-Ende Sprachverschlüsselung (Vgl. [71])

Der GSM-Sprachkanal erlaubt nur den Transport von Sprachsignalen die den Charakteristika von Sprache entsprechen. Dies ist bedingt durch die verwendeten Decoding- und Encoding-Chips, sowie durch die Kompression, die nur bei Sprachsignalen korrekt arbeiten kann. Nach dem Verschlüsseln entsprechen die Sprachsignale allerdings nicht mehr den Charakteristika von Sprachsignalen. Dadurch wird eine erneute Modulation der verschlüsselten Sprachdaten auf ein sprachähnliches Trägersignal notwendig um sie über das GSM-Netz zu übertragen.

Die Übertragung von Daten, beziehungsweise von verschlüsselter Sprache wurde durch einen Prototypen belegt. Zudem bietet der GSM Sprachkanal nur ein geringes Delay, das durch das geringe zusätzliche Delay beim Verschlüsseln nicht sonderlich vergrößert wird. Für detailliertere Erläuterungen zum Verfahren und weiterführende Infos können die Arbeiten von [71, 72, 73] herangezogen werden.

Dieses Konzept ist allerdings, neben den hier dargestellten technischen Schwierigkeiten, auch aufgrund der Android Systemarchitektur nicht realisierbar, da auf die Sprachdaten nicht zugegriffen werden kann, bevor sie vom GSM-Telefoniestack verarbeitet werden (siehe auch Erläuterungen in Kapitel 2.2).

7 Codecs

In diesem Kapitel werden die unter Android verfügbaren Audiocodecs vorgestellt und evaluiert bezüglich ihrer Eignung für die Kodierung von Sprachdaten und die Resistenz gegen Angriffe auf die Vertraulichkeit bei abhörsicherer Sprachkommunikation.

Die Auswahl eines Audiocodecs für die Sprachkodierung hat in erster Linie einen direkten Einfluss auf die Sprachqualität und damit auf die Benutzerzufriedenheit. Audiocodecs unterscheiden sich durch unterschiedliche Bitraten (*Datenmenge pro Zeiteinheit*) und die Abtastrate (*Messwerte pro Zeiteinheit*). Durch Kompression oder die Verwendung einer variablen Bitrate wird die Bitrate beeinflusst, ebenso durch das verwendete Kodierungsverfahren.

Weniger offensichtlich ist, dass die Auswahl des Audiocodecs einen Einfluss auf die Abhörsicherheit von verschlüsselter Sprachkommunikation haben kann. Durch die Verwendung eines Audiocodecs mit variabler Bitrate und trotz der Anwendung von AES-Verschlüsselung[74] auf die Sprachdaten, können durchschnittlich 50% und maximal bis zu mehr als 90% von gesprochen Phrasen, wiederhergestellt werden, durch die Anwendung statistischer Verfahren. Diese Sicherheitslücke kann allerdings leicht eliminiert werden indem man einen Codec mit Konstanter Bitrate verwendet oder Padding anwendet und dabei alle Pakete mit Nullen auf eine bestimmte Größe füllt. Padding oder die Verwendung eines Codecs mit fixer Bitrate ist allerdings nicht Teil des SRTP-Standards und wurde daher als Resistenz gegen Encrypted Voice Reconstruction unter Kapitel 4.2 eingeführt (Vgl. [54, S. 6f]).

Die Evaluierung unterschiedlicher Audiocodecs beschränkt sich in dieser Thesis auf die unter Android 2.3.4 verfügbaren Audiocodecs (siehe [75]). Verfügbar bedeutet, dass ein Encoder und Decoder für den Audiocodec im System integriert ist und der Codec den unter Kapitel 4 definierten Anforderungen genügt. Dies ist darin begründet, dass durch die Verwendung der mitgelieferten Codecs, der Implementierungsaufwand für die Anwendung zur sicheren Sprachkommunikation verringert wird und die Kosten der Implementierung verringert werden.

Codec	Bitrate	Abtastrate	MUSHRA Score
AAC LC/LTP	bis zu 160 kbps	8-48 kHz	63 @ 19,85 kbps (Vgl. [76, 78])
AMR-NB	4.75 - 12.2 kbps	8 kHz	46 @ 12,2 kbps[79, 80]
AMR-WB	9 Raten, von 6,6 kbps bis 23,85 kbps	16 kHz	82 @ 19,85 kbps (Vgl. [76, 78, 80])
G.711 *(Referenz)*	64 kbps	8 kHz	35 @ 64 kbps[80]

Tabelle 7.1: Android Audio Codecs (Vergleich)

Die Liste, der von Android unterstützten Audiocodecs für das Transcoding, umfassen:

AAC LC/LTP Advanced Audio Coding - Low Complexity/Long-Term Prediction

AMR Adaptive Multi-Rate Audio Codec

- **AMR-NB** Adaptive Multi-Rate Narrow Band Codec
- **AMR-WB** Adaptive Multi-Rate Wide Band Codec

Der AAC LC/LTP Codec bietet bei niedrigen Bitraten (*bei 19,8 kbps*) eine gute Sprachqualität. Bedingt durch die Tatsache, dass es sich bei AAC LC/LTP um einen Codec handelt der im Gegensatz zu AMR-NB/-WB nicht speziell für Sprache entwickelt wurde, ist die Sprachqualität des AAC Codecs bei niedrigen Bitraten allerdings weniger gut als die des AMR Codecs. Ein weiterer Nachteil des AAC-Codecs ist das hohe De-/Encoding Delay des AAC Codecs, was ihn für Zwei-Wege-Sprachkommunikation weniger attraktiv macht[76, S. 4]. Der AAC Codec erlaubt eine konstante Bitrate und macht verhindert damit die Attacke der *Encrypted Voice Reconstruction*.

Die AMR Kodierung basiert auf dem Algebraic Code Excited Linear Prediction[77] Verfahren welches bei geringer Bitrate eine sehr gute Sprachqualität ermöglicht. Beide AMR Codecvarianten erlauben und benötigen eine Adaptierung an die Transportnetzeigenschaften in Bezug auf die benötigte Bandbreite. Durch eine Verringerung der Bitrate und Vergrößerung der Anteile an Fehlerkorrekturdaten kann auf schlechte Übertragungseigenschaften, wie beispielsweise Paketverlust oder geringe Bandbreite reagiert werden.

Die Unterstützung mehrerer Codecs in der Anwendung ist je nach verwendetem Signalisierungsprotokoll möglich. Der zu verwendende Audiocodec könnte so von zwei Endgeräten anhand einer prioritätsbasierten Präferenzenliste vereinbart werden.

Um eine bestmögliche Sprachqualität zu erreichen wird in der Referenzimplementierung der AMR-WB Codec verwendet. Da es sich bei AMR-WB um einen Codec mit konstanter Bitrate handelt, kann er ohne Padding für abhörsichere Sprachkom-

munikation verwendet werden. Durch die gute Sprachqualität von 82 MUSHRA bei 19,85 kbps ist er zudem der qualitativ beste Codec der hier vorgestellten Audiocodecs. Weiterführende Information zur Sprachkodierung können den Büchern [81, S. 141ff] und [36, S. 145ff] entnommen werden.

8 Medientransport und Medientransportkontrolle

Die Hauptaufgabe einer Anwendung zur Sprachkommunikation ist der Transport von Sprachdaten zwischen den Kommunikationsteilnehmern. In diesem Kapitel werden möglichen Optionen evaluiert, um Sprachdaten zwischen den Kommunikationsteilnehmern bidirektional (*vollduplex*) zu transportieren und dabei den in Kapitel 4 definierten Anforderungen an abhörsichere digitale Sprachkommunikation zu genügen.

8.1 OSI-Layer 4 Transportprotokolle

Die in den folgenden Kapiteln vorgestellten Protokolle, das in Kapitel 8.2 evaluierte RTP eingeschlossen, sind Protokolle, die im OSI-Modell im Sitzungslayer (Layer 5) einzuordnen sind. Gegen den direkten Transport von Sprachdaten in einem Layer 4 Transportprotokoll sprechen bei den bekanntesten Transportprotokollen folgende Argumente:

UDP Das *User Datagram Protocol[82]* ermöglicht keine Überwachung des Datentransports, der Reihenfolge oder der zeitliche Markierung von Paketen.

TCP Das *Transmission Control Protocol[83]* ermöglicht zwar den geordneten Transport von Daten, hat aber aufgrund der TCP-Congestion Control Mechanismen, also dem langsamen Beginn der Datenübertragung und der starke Geschwindigkeitsreduktion bei Paketverlusten, den Nachteil zusätzliches Delay und Jitter beim Datentransport zu verursachen. TCP ist damit nicht für den Transport von Echtzeitsprachdaten geeignet.

DCCP Das *Datagram Congestion Control Protocol[84]* ermöglicht keine Überwachung des Datentransports, der Reihenfolge oder der zeitliche Markierung von Paketen und hat damit die gleichen Nachteile wie UDP.

SCTP Das *Stream Control Transmission Protocol[34]* vereint die Eigenschaften von UDP und TCP und bietet genau wie TCP einen geordneten und verlässlichen Transport von Paketen ist dabei aber datagrammbasiert wie UDP. Aufgrund

der TCP-ähnlichen Flusskontrolle hat SCTP den Nachteil, genau wie TCP, zusätzlichen Delay und Jitter zu verursachen.

8.2 Real-Time Transport Protocol

Das verbreitetste Protokoll für den Transport von Daten mit Echtzeitanforderungen ist das Real-Time Transport Protocol (RTP)[85, 36]. RTP ist speziell für den Transport von Multimediadaten entwickelt worden (aber keineswegs darauf beschränkt) und ist unabhängig von den unteren Transport- und Netzwerkschichten. Um die Netzeigenschaften zu messen, zu überwachen und um auf die Eigenschaften der unteren Netzwerkschichten reagieren zu können, ist das begleitende Medientransportkontrollprotokoll *Real-Time Control Protokoll* (RTCP) spezifiziert und übernimmt diese Aufgaben. Die Charakteristika, die RTP auszeichnen und die es für den Transport von Echtzeitdaten prädestinieren, sind:

Datendurchsatz Aufgrund des geringen Overheads durch den RTP Header (*minimal 12 Bytes*) hat das RTP Protokoll keinen erwähnenswerten negativen Einfluss auf den Datendurchsatz. Durch die Überwachungsfunktionen des RTCP kann zusätzlich auf Ereignisse im Netzwerk reagiert werden und so beispielsweise die Paketgröße verändert werden, um beispielsweise Fragmentierung im Netzwerk entgegenzuwirken.
Durch den Protokolloverhead von 12 Bytes wird auch auf die geringe Bandbreite von mobilen Endgeräten Rücksicht genommen.

Reihenfolge der Pakete wird gewahrt durch Nummerierung der Pakete. Am Ziel können so Paketfolgen die ungeordnet eintreffen, wieder in die richtige Reihenfolge gebracht werden. Dies ist eine Möglichkeit um Jitter entgegenzuwirken.

Isochronität Durch die Vergabe von Zeitstempel beim Versenden von RTP-Paketen können am Kommunikationsendpunkt die Zeitabstände zwischen den Paketen wiederhergestellt werden. Dies ist eine weitere Möglichkeit um Jitter beim Transport der Daten entgegenzuwirken.

Universelles Transportprotokoll Durch die Erweiterbarkeit von RTP durch *Profile* kann das Protokoll für den Transport beliebiger Multimediadaten angepasst werden. Dazu zählen unter anderem Audio-, Video- und andere Daten mit Echtzeitanspruch wie sie beispielsweise bei Netzwerkspielen anfallen.

Transportüberwachung Mit Hilfe des Real Time Control Protokolls kann die Datenübertragung überwacht und kontrolliert werden und so auf Anwendungsebene auf die Eigenschaften und Ereignisse im Transportnetz reagiert werden.

RTP ist *unvollständig* definiert, um es durch Zusatzspezifikation an unterschiedliche Bedürfnisse beim Datentransport anpassen zu können[85, Chap. 1]. RTP ist dadurch

sehr einfach erweiterbar. Eine Notwendigkeit beim Einsatz von RTP sind *daher RTP-Profile*, in denen neben der Spezifikation des Payload, auch Erweiterungen oder Modifikation an der RTP-Spezifikation erlaubt sind. Ein mögliches und häufig verwendetes Profil ist das unter [86] spezifizierte, *RTP Profile for Audio and Video Conferences with Minimal Control* (RTP/AVP). Im RTP Audio Video Profile ist detailliert beschrieben wie Audio und Videodaten mittels RTP transportiert werden, einschließlich der erlaubten Audio und Videocodecs.

Die Funktionsweise des RTP-Protokolls ist an das Funktionsprinzip der normalen Telefonie angelehnt. Mittels eines unterstützenden Signalisierungsprotokolls (Evaluation in Kapitel 9), wird eine *RTP-Session* aufgebaut, da das RTP keine protokollinterne Möglichkeit zum Sitzungsaufbau spezifiziert.
Diese besteht aus zwei separaten *logischen Channels*, einem Media Channel, mittels dem Echtzeitdaten durch das Real Time Protokoll transportiert werden und einem Media Control Channel, der durch das Protokoll RTCP realisiert ist und den Media Channel überwacht (Vgl. [36, S. 149-150]). Das Monitoringprotokoll RTCP arbeitet dabei nach dem Prinzip, dass periodisch Kontrollpakete verschickt werden, deren Laufzeiteigenschaften gemessen werden. Aufgrund der Analyse der RTCP Laufzeiteigenschaften, kann der Media Channel konfiguriert werden um beispielsweise auf Jitter zu reagieren.

Beim Sitzungsaufbau tauschen die beiden Kommunikationsendpunkte die Eigenschaften aus, die die aufzubauende RTP-Session haben soll (siehe Abbildung 8.1). Die Eigenschaften der RTP-Session können abhängig vom Signalisierungsprotokoll, beispielsweise mit Hilfe des Session Description Protokolls (SDP)[87] ausgehandelt werden.

Ist die RTP-Session aufgebaut, findet der Transport von Multimediadaten gekapselt innerhalb von RTP-Paketen statt, die wie in Abbildung 8.2 dargestellt, aufgebaut sind.
Erwähnenswerte Felder im RTP-Header, sind neben der bereits angesprochenen Sequenznummer und dem Zeitstempel, die Felder Synchronization Source Identifier (SSRC) und Contributing Source Identifier (CSRC). Das Feld SSRC identifiziert die Quelle, aus der das Payload des RTP-Paketes stammt. Pakete aus unterschiedlichen Quellen müssen eine unterschiedlichen SSRC-Wert haben, damit der Empfänger diese für die Wiedergabe unterscheiden kann. Das CSRC-Feld ist optional und dient einem ähnlichen Zweck. Das CSRC-Feld wird von einer Zwischeninstanz (beispielsweise einem Medienmixer) beim Transport eines RTP-Paketes zum Empfänger gesetzt, falls dieser den Payload des RTP-Paketes neu zusammengesetzt hat. Im CSRC-Feld steht dann eine Liste mehrerer Multimediaquellen (mehrere SSRCs) aus denen die Payload des RTP-Paketes zusammengesetzt ist.

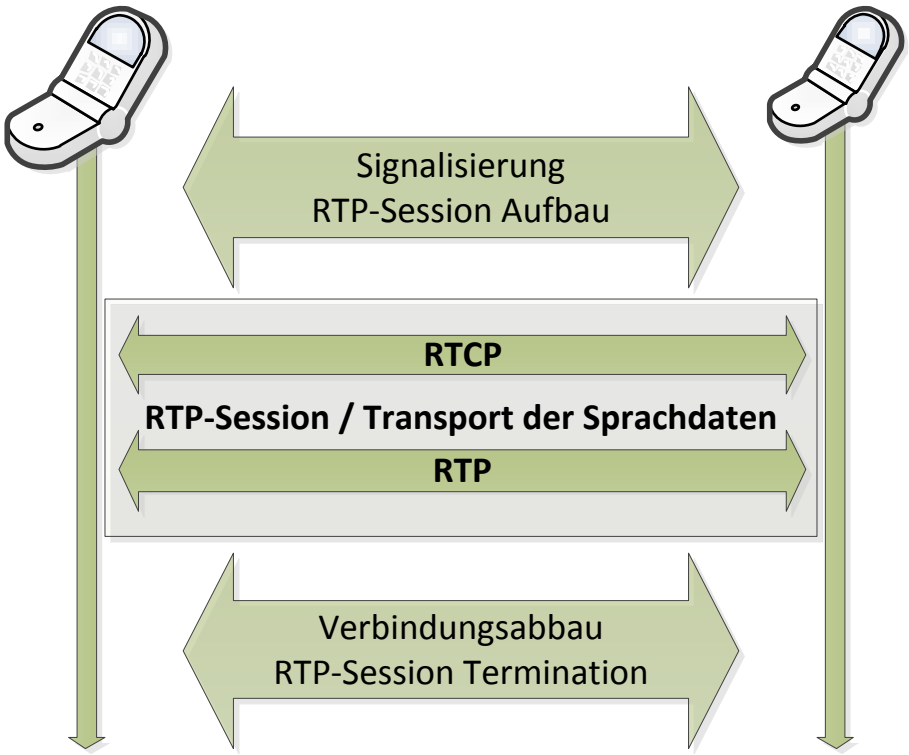

Abbildung 8.1: RTP-Session

RTP ist optimal geeignet aus Sicht der Anforderungen an digitale Sprachkommunikation, da durch die relative Simplizität des Protokolls, keine große Erhöhung von Delay und Bandbreite verursacht werden und da aufgrund des RTCP, auf Jitter oder verlorene Pakete reagiert werden kann. Das Protokoll erfüllt aber nicht die Anforderungen an abhörsichere Sprachkommunikation, da RTP weder die Vertraulichkeit wahrt, noch die Datenintegrität sicherstellt. Daher ist ein zusätzlicher Mechanismus notwendig, um die Anforderungen an abhörsichere Sprachkommunikation umzusetzen.

RTP kann Audiodaten die mit dem AMR-WB Codec kodiert sind transportieren, da AMR-WB als Payload Format für RTP spezifiziert ist[89].

Um die Anforderungen an die sichere Sprachkommunikation (siehe Kapitel 4.2) erfüllen zu können, wird für die Absicherung des Real-Time Protokolls die Erweiterung *Secure Real-Time Protocol*[90] (SRTP) in Form eines RTP-Profiles benutzt.
SRTP spezifiziert die Verschlüsselung des Payloads von RTP-Paketen. Dadurch werden RTP-Pakete gegen Eavesdropping gesichert (Vertraulichkeit), der Absender wird authentifiziert, die Datenintegrität wird gewahrt und SRTP bietet Schutz vor Replay-Attacken (siehe Abbildung 8.3). Nach dem gleichen Prinzip werden auch RTCP-Nachrichten abgesichert, die dann als *Secure Real-Time Control Protocol* (SRTCP) Nachrichten bezeichnet werden.

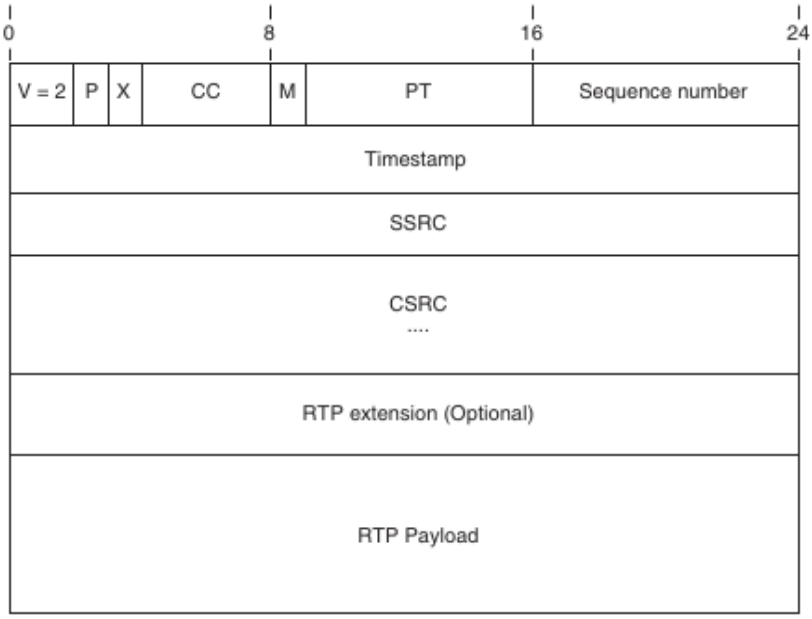

Abbildung 8.2: RTP Paket (Quelle: [88])

Zur Verschlüsselung spezifiziert SRTP die Verwendung des Advanced Encryption Standard (AES) Algorithmus, entweder im Counter Mode (AES-CTR) oder im F8-Mode (AES-F8), der eine Variante des Output Feedback Mode (AES-OFB) darstellt und für UMTS optimiert wurde. Eine ausführliche Erläuterung zu AES und den differenten Operationsmodi findet man in[88, S. 197ff]. Des Weiteren wird im Kapitel zu den kryptographischen Grundlagen, unter Anhang C, in einem Überblick darauf eingegangen.

Um Verschlüsselung und Authentifizierung zu realisieren benutzt SRTP geheime Schlüssel. SRTP spezifiziert allerdings keine Möglichkeiten zum Schlüsselaustausch oder zum Festlegen der Sicherheitsparameter für die Schlüsselgenerierung zwischen den Kommunikationsendpunkten. Aus diesem Grund benötigt man zusätzlich zu SRTP ein Key Management Protokoll, das SRTP um die fehlenden Key Management Funktionen ergänzt. Mögliche Key Management Protokolle werden in Kapitel 10 separat evaluiert.

Laut [91] hat die Verwendung des Diffie-Hellmann Algorithmus zum Schlüsselaustausch und Verschlüsselung der RTP-Pakete mittels AES nur eine geringe Vergrößerung des Delays zur Folge, die vernachlässigbar ist. Diese sehr geringe Vergrößerung des Delays lässt sich durch die Verwendung von speziellen Kryptographiebeschleunigern noch weiter senken.

RTP-Nachrichten können in einem beliebigen Layer 4 Transportprotokoll transportiert werden. Die bekanntesten Transportprotokolle für den Transport von RTP-Nachrichten und Mediendaten im allgemeinen wurden bereits in Kapitel 8.1 vorge-

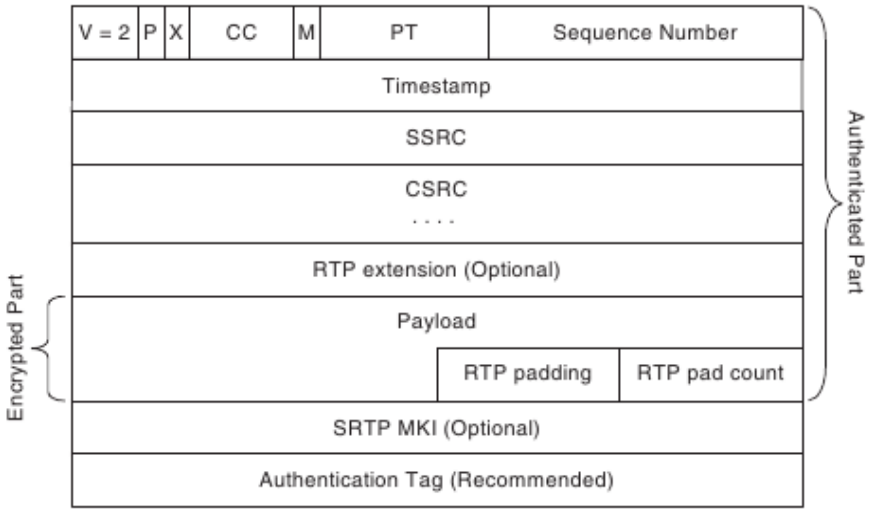

Abbildung 8.3: SRTP Paket (Quelle: [88])

stellt. Die Argumente die gegen den Transport von RTP in TCP oder SCTP Paketen sprechen, kommen weiterhin zum Tragen.
Die Argumente gegen UDP oder DCCP werden durch die Verwendung von RTP mit dem RTCP aufgehoben, so dass beide Protokolle für den Transport von RTP-Nachrichten in Frage kommen.

In dieser Thesis wird das UDP Protokoll als Transportprotokoll für RTP verwendet, da es im Internetprotokollstack unter Android integriert ist. DCCP ist ebenfalls optimal geeignet und bietet im Gegensatz zu UDP noch zusätzliche Congestion Control Mechanismen. Für die Verwendung von DCCP unter Android müsste eine externe DCCP-Implementierung verwendet werden, worauf aus Kosten- und Zeitgründen verzichtet wird.

8.3 Alternative Streamingprotokolle

Neben dem Real-Time Transport Protocol existieren noch andere Streamingprotokolle die aufgrund ihrer Eigenschaften allerdings nicht für den bidirektionalen Transport von Sprachdaten geeignet sind.

Dazu gehören das *Real-Time Streaming Protocol*[92] (RTSP) und das proprietäre Protokoll *Microsoft Media Server*[93] (MMS), die nur für den Abruf von gespeicherten Echtzeitmedien geeignet sind. Ausführliche Informationen zu RTSP kann man unter [81] einsehen.

Das *Real-Time Messaging Protocol* (RTMP) von Adobe wäre für den bidirektionalen Transport von Sprachdaten geeignet, da seine Eigenschaften ähnlich denen von RTP sind. Aufgrund der unzureichenden und lückenhaften Dokumentation scheidet es

allerdings aus der Liste der möglichen Medientransportprotokolle aus. In der RTMP-Dokumentation sind zudem keinerlei Sicherheitsmechanismen spezifiziert, so dass eine Umsetzung der Anforderungen an sichere Sprachkommunikation in den unteren Protokollschichten, beispielsweise durch IPSec[94] realisiert werden müssten[95].

8.4 Externe Absicherung der Transportprotokolle

Man kann das Medientransportprotokoll auch durch externe Sicherungsmaßnahmen in den darunterliegen Protokollschichten (OSI-Modell) absichern. Gegen die externe Absicherung des Medientransports sprechen allerdings einige Argumente die hier kurz vorgestellt werden.

DTLS Das Protokoll Datagram Transport Layer Security[96] verschlüsselt das gesamte Datenpaket das durch DTLS transportiert wird. Dadurch können QoS-Maßnahmen die VoIP-Traffic bevorzugt behandeln nicht mehr angewandt werden.

TLS Das Protokoll Transport Layer Security[97] ist auf TCP als Transportprotokoll angewiesen und kommt durch die schlechten Eigenschaften von TCP als Transportprotokoll für Echtzeitsprachdaten nicht in Frage. Zudem verschlüsselt TLS, ebenso wie DTLS, das gesamte Datenpaket, wodurch QoS-Maßnahmen die VoIP-Traffic bevorzugt behandeln nicht mehr angewandt werden können.

IPsec Die Sicherheitsprotokollsammlung Internet Protocol Security (IPsec)[94] ist im OSI Referenzmodell noch eine Ebene tiefer angesiedelt wie (D)TLS und kann deswegen TCP und UDP Pakete verschlüsseln. Eines der Probleme ist bei IPsec, genau wie bei TLS und DTLS, dass QoS-Maßnahmen die VoIP-Traffic bevorzugt behandeln nicht, mehr angewandt werden können.

8.5 Fazit

Für den sicheren Transport von Sprachdaten wird in der vorliegenden Arbeit das Real-Time Protokoll mit dem Erweiterungsprofil Secure Real-time Transport Protokoll eingesetzt. RTP ist zur Zeit für den Transport von Echtzeitmultimediadaten konkurrenzlos und optimal geeignet, da es als einziges Medientransportprotokoll, alle Anforderungen an die abhörsichere mobile Sprachübertragung erfüllt und dazu ein erprobter und frei verfügbarer Standard ist.
Als Transportprotokoll für RTP Pakete wird UDP, wie in Kapitel 8.2 eingesetzt.

Es wird SRTP zur Absicherung der RTP-Pakete benutzt, da SRTP, statt einer externen Absicherung keinen Einfluss auf die Anwendung von Quality of Service auf dem Transportweg hat. Im Gegensatz zu (D)TLS und IPSec. SRTP verschlüsselt ausschließlich den Payload eines RTP Paketes und erlaubt Router die Paketanalyse um spezifische Weiterleitungsregeln für VoIP-Traffic anzuwenden.

Da die Absicherung durch SRTP im Anwendungslayer des OSI Referenzmodells durchgeführt wird, wird keine Unterstützung auf Seiten der Router benötigt.

9 Signalisierung

RTP Medienverbindungen müssen signalisiert werden, da das Real-Time Protokoll keine protokollinternen Möglichkeiten bietet den Kommunikationswunsch zu signalisieren. In diesem Kapitel werden einige Technologien evaluiert, die als Signalisierungsprotokoll für RTP-Sitzungen unter Android verwendet werden können.

9.1 Session Initiation Protocol

SIP steht für *Session Initiation Protocol* ,ist ein weit verbreitetes und, durch den Einsatz im IP Multimedia Subsystem (IMS), sehr erprobtes Signalisierungsprotokoll. Die SIP-Nachrichten, die vom Protokoll ausgetauscht werden, sind textbasiert und damit menschenlesbar. SIP kann wahlweise über UDP, TCP und SCTP transportiert werden. Die Funktionalitäten von SIP sind sehr umfangreich und nicht auf die bloße Anrufsignalisierung beschränkt. Zu den Funktionen von SIP zählen unter anderem (Vgl. [81, 98]):

Namensübersetzung und Benutzerlokalisierung Funktion als Benutzerdatenbank und Möglichkeiten der Benutzerlokalisierung anhand eines SIP Uniform Resource Identifiers (SIP-URIs).

Capability Exchange Austausch der Endpunkt-Fähigkeiten durch Session Description Protokollnachrichten, die mit Hilfe des SIP Protokolls transportiert werden. Das Session Description Protocol beschreibt detailliert die Fähigkeiten und Anforderungen, die an die Sitzung die aufgebaut werden soll gestellt werden, einschließlich aller Codec und Verbindungsparameter.

User Availability and Presence Überwachen und Statussignalisierung von angemeldeten Benutzern.

Session Setup and Management Starten von Mediensitzungen, in dem konkreten Fall dieser Thesis, das Starten von RTP-Sitzungen.

SIP Peers werden als User Agent (UA), bzw. User Agent Client (UAC) bezeichnet, wenn es sich um Clients handelt, oder als User Agent Server (UAS), wenn es sich um eine SIP-Proxy oder allgemein um eine SIP-Server Entität handelt.

Mit Hilfe des Session Initiation Protokolls kann zwischen zwei Peers direkt signalisiert werden, ohne dass ein SIP-Server als Registrar oder Proxy verwendet wird. SIP

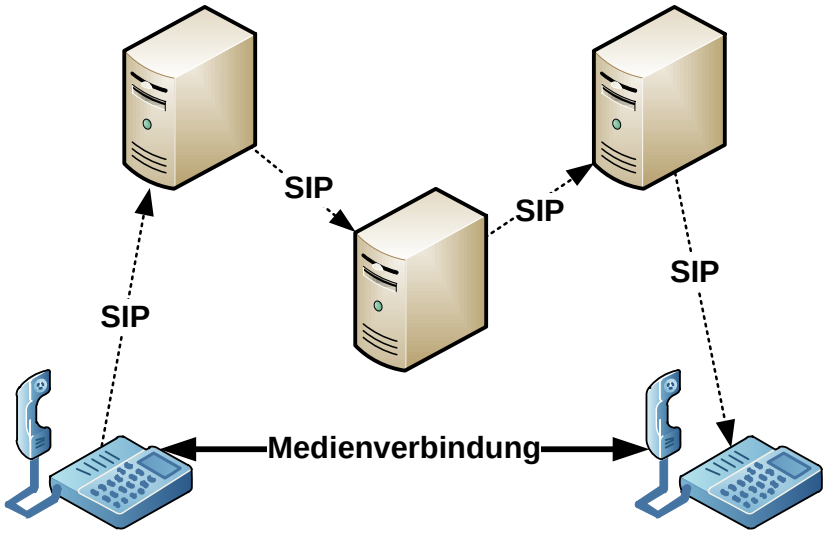

Abbildung 9.1: SIP-Trapezoid

wird aber für gewöhnlich in einem Client-Server-Szenario benutzt. Die Signalisierung zwischen dem User Agent Client wird dabei über einen oder mehrere SIP-Proxies weitergeleitet (siehe Abbildung 9.1). Damit ein UAC im Netz gefunden werden kann, melden sich diese bei einem SIP-Registrar (auch als Location Server bezeichnet) an, der meist, aber nicht zwangsläufig ein SIP-Proxy mit einem Registrar-Service ist, statt einer separaten Entität.

Der Ablauf einer SIP-Signalisierung inklusive dem Aufbau einer Mediensitzung und dem anschließenden Abbau ist in Abbildung 9.2 vorgestellt. Zur Adressierung verwendet SIP einen Uniform Resource Locator (URL).

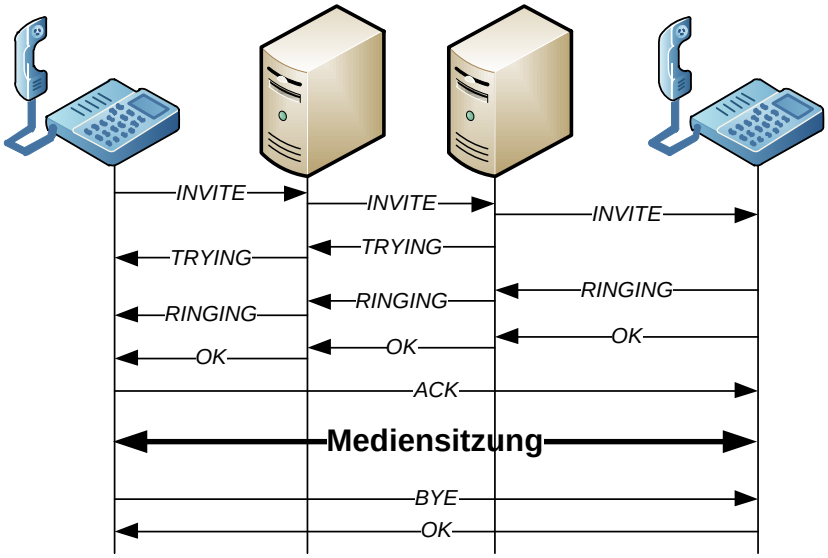

Abbildung 9.2: SIP Signalisierung (Protokollablauf) (Vgl. [99])

Listing 9.1: SDP Nachricht

```
v=0
o=sweisgerber 6345123463 567889123452 IN IP4 88.198.204.165
s=Audio Call
i=Small Talk
u=http://dystopianfuture.org
e=sebastian@dystopianfuture.de (Sebastian Weisgerber)
c=IN IP4 134.96.208.112
t=5614397496 5614404696
a=recvonly
m=audio 49171 RTP/AVP 0
a=rtpmap:99 h263-1998/90000
```

Das SIP-Protokoll spezifiziert zwei unterschiedliche Nachrichtenkategorien. Zum einen *SIP-Requests*, die ohne vorhergehende Kommunikation zwischen zwei Kommunikationspartnern ausgetauscht werden kann, zum anderen *SIP-Responses*, die als Antwort auf eine SIP-Request Nachricht verschickt werden.
Es werden in diesem Kapitel nicht alle Nachrichtentypen vorgestellt, da dies den Umfang dieser Arbeit sprengen würde. Erwähnenswert sind allerdings *SIP-Register* (Request) Nachrichten, mit denen sich ein User Agent am SIP-Registrar an- und abmeldet und damit seine Verfügbarkeit für Anrufe signalisiert. Der zweite wichtige Nachrichtentypus sind *SIP-Invite* (Request) Nachrichten, mit denen einem potentiellen Kommunikationspartner ein Anrufwunsch mitgeteilt wird.

Im SIP-Invite, werden Nachrichten des Session Description Protokolls [87] ausgetauscht um zu signalisieren welche Mediensitzung aufgebaut werden soll. In der SDP-Nachricht werden alle notwendigen Parameter angegeben, die für den Aufbau der Medienverbindung notwendig sind, wie IP-Adresse, verwendetes RTP-Profil oder der gewünschte Audiocodec. SDP ist unter [87] spezifiziert. Ein Beispiel einer SDP-Nachricht ist im Quellcodeauszug unter 9.1 dargestellt.

SIP ähnelt in seiner Funktionsweise dem HTTP-Protokoll[100] und spezifiziert ähnliche Sicherheitsmechanismen. So wird für die Authentifizierung, die mittlerweile aufgrund zahlreicher Sicherheitslücken und schwacher Sicherheit als veraltet eingestufte HTTP-Digest Authentifizierung[101] angeboten, diese kann allerdings durch zusätzliche Mechanismen abgesichert werden. Um den Sicherheitsanforderungen unter Abschnitt 4.2 zu genügen bietet SIP folgende zusätzliche Möglichkeiten:

SIP Digest Authentication over TLS (SIPS) Gewöhnliche HTTP-Digest-Authentication, gesichert durch das Transport Layer Security Protokoll (TLS). SIPS realisiert keine Ende-zu-Ende Verschlüsselung sondern, bedingt durch die SIP-Proxy Architektur, nur eine Verschlüsselung der Verbindung zwischen benachbarten Hops. SIPS benötigt aufgrund der Einschrän-

kungen durch TLS, das nur verbindungsorientierte Protokolle unterstützt, das TCP-Protokoll und funktioniert nicht über UDP.
TLS basiert auf Zertifikaten und benötigt eine Public Key Infrastructure (PKI) und kann damit zwischen zwei einander unbekannten Systemen eingesetzt werden um damit Vertraulichkeit, Integrität und Autorisierung zwischen den Kommunikationsendpunkten zu realisieren (Vgl. [50, S. 108]). Weiterführende Informationen zu TLS können [97] entnommen werden.
Ein negativer Punkt bei der Verwendung von SIP über TLS ist, dass der Verbindungsaufbau durch die Verwendung von TCP und TLS verlängert wird [102].

IPSec Statt über TLS, kann die SIP Digest-Authentication auch mittels IPSec auf dem Transport- und Netzwerklayer abgesichert werden. Im Gegensatz zu TLS können über einen IPSec-Tunnel auch UDP-Pakete verschickt werden. Es ist ebenfalls möglich den IPSec Tunnelaufbau mittels SIP zu signalisieren.
IPSec kann sowohl wie bei SIPS Hop-by-Hop eingesetzt werden, es kann aber auch eine direkte Tunnelverbindung zwischen den Kommunikationsendpunkten aufgebaut werden. Eine einheitliche Schlüsselverwaltung ist nicht spezifiziert, kann aber beispielsweise durch das Protokoll Internet Key Exchange(IKE)[103] realisiert werden. IPSec ermöglicht Integrität und Vertraulichkeit der SIP-Nachrichten (Vgl. [104]).
SIP mit UDP über IPSec verlängert die Zeitdauer für den Verbindungsaufbau nur unmerklich (Siehe: [102]).

S/MIME bietet Ende-zu-Ende Verschlüsselung der SIP-Nachrichten, abzüglich einiger Header-Informationen, die für das Routen der SIP-Nachrichten erforderlich sind, wie beispielsweise die Zielhost- und Quellhostadresse. Die Endpunkte der Signalisierung sind damit nicht vertraulich. Die Ende-zu-Ende Verschlüsselung wird durch die Verwendung von Zertifikaten realisiert, die durch Signierung der SIP-Nachrichten eine Identifizierung der Kommunikationsteilnehmer und die Integrität der Daten sicherstellen. Um eine vollständige SIP-Nachricht zu verschlüsseln kann diese mittels S/MIME gesichert und in einer äußeren SIP-Nachricht verpackt werden (*SIP-Tunneling*).
S/MIME ermöglicht damit Vertraulichkeit und Integrität von SIP-Nachrichten. Das Kriterium der Verfügbarkeit wird durch S/MIME nicht gewährleistet, ebenso bietet S/MIME keinen Schutz vor Replay-Attacken.
Die Zertifikate die S/MIME für die Verschlüsselung und Signierung benutzt müssen verteilt werden, beispielsweise durch eine Public Key Infrastruktur (PKI). Es ist aber auch möglich die Public Keys durch SIP-Nachrichten auszutauschen. Eine einheitliche vorgeschriebene Schlüsselverwaltung ist nicht spezifiziert. Eine weltweit zugängliche und vereinheitliche PKI für die S/MIME

Zertifikatsverwaltung und dem Zertifikatsaustausch wäre sinnvoll, fehlt aber zur Zeit völlig.

Das Problem der Zertifikatsverteilung ist aber keines das sich ausschließlich auf S/MIME zur Absicherung des SIP-Protokolls bezieht, sondern ein generelles Problem, der zertifikatsbasierten Verschlüsselung und Authentifizierung.

Ein weiterer Nachteil, der durch den Einsatz von S/MIME verursacht wird, ist die zusätzliche Zeitdauer, die der Verbindungsaufbau bei der Anrufsignalisierung benötigt (Vgl. [51, S. 7]).

S/MIME kann x.509 Zertifikate verwenden, benötigt aber für deren Austausch ein separates Protokoll zur Schlüsselverwaltung oder manuell verteilte Zertifikate.

SIP Identity Der SIP Identity Mechanismus[105] stellt eine Möglichkeit zur Verfügung SIP-Requests Ende-zu-Ende, mit Hilfe von kryptographischen Verfahren, zu authentifizieren. SIP-Responses sind davon ausgenommen. Zur Authentifizierung verbindet sich der SIP-UAC von Alice mit dem SIP-Proxy (bestenfalls gesichert via TLS), der den SIP Identity Service bereitstellt. Der SIP-Proxy authentifiziert den Alice SIP-UAC und berechnet einen Hash über den SIP-Header und signiert diesen mit dem Domain-Zertifikat. Zusätzlich fügt Alices Proxy in den SIP-Header Informationen ein, wo man das Zertifikat mit dem signiert wurde, abrufen kann. Der SIP Identity Service muss hierfür allen SIP-Proxies auf der gesamten Signalisierungsstrecke zwischen den beiden UACs vertrauen.

Die signierte SIP-Nachricht wird danach an den SIP-Proxy der Domain von Bob weitergeleitet, der die Authentizität prüft und die Identität von Alice verifiziert. Bobs UAC kann den Identitätscheck ebenfalls (nochmals) durchführen. Nach dem gleichen Prinzip kann Alice die Identität von Bob verifizieren. Die UACs müssen hierbei den SIP-Proxies vertrauen, basierend auf den Prinzipien von TLS und Certificate Authorities (CA). Durch dieses Verfahren kann auf eine PKI die von den UACs verwaltet wird verzichtet werden und Alice und Bob müssen sich nicht um die Zertifikatsverwaltung kümmern[105].

SIP Connected Identity Diese Ergänzung des SIP und SIP Identity Protokolls erweitert den SIP Identity Mechanismus um die Möglichkeit, während eines Gespräches einen Identitätswechsel von Alice oder Bob zu signalisieren, wie er beispielsweise bei einer Anrufweiterleitung oder dem weiterleitenden eines Anrufers zu einem anderen Kommunikationspartner durchgeführt werden muss. Die Authentifizierung erfolgt dabei nach den Mechanismen die SIP Identity spezifiziert[106].

Die vorgestellten Verfahren zur Absicherung des Session Initiation Protokolls erhöhen allerdings nicht die Verfügbarkeit oder die Resistenz gegenüber (D)DoS Atta-

cken.

9.2 H.323-SIG

Ein weiteres Signalisierungsprotokoll ist die H.323-SIG Protokollfamilie[39]. Im Gegensatz zu SIP verwendet H.323 ein binäres Nachrichtenencoding, was sich in einer geringeren Größe der Signalisierungsnachrichten, im Vergleich zu SIP- oder Jingle-Nachrichten, bemerkbar macht. H.323 verwendet das Unterprotokoll H.225.0[107] zur Anrufsignalisierung, zur Teilnehmerregistrierung und zur Rechte- und Statusüberwachung (Registration, Admission and Status - RAS).

Durch ein weiteres Unterprotokoll (H.245.0[108]) werden Medienverbindung auf- und abgebaut, sowie die Endpunktfähigkeiten ausgetauscht. Es können damit RTP und SRTP Verbindungen aufgebaut werden.

Der Aufbau einer RTP Medienverbindung mit Hilfe des H.323 Protokolls wird in Abbildung 9.3 dargestellt. Zuerst wird mit Hilfe des H.225 Signalisierungsprotokolls der Anrufwunsch mitgeteilt. Danach werden durch das H.245 Protokoll die Parameter und Fähigkeiten der Medienverbindung vereinbart und danach basierend auf diesen Parametern die RTP und RTCP Verbindungen aufgebaut.

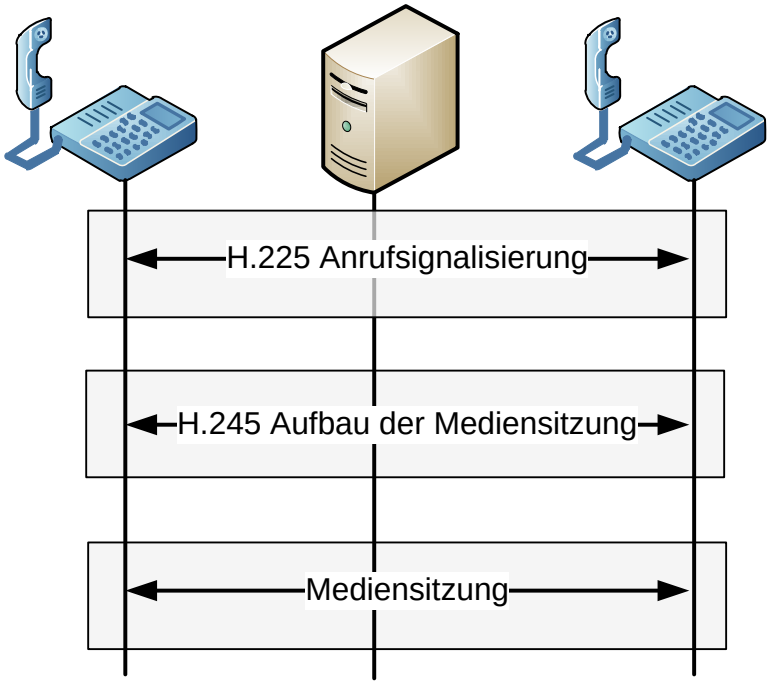

Abbildung 9.3: H.323 Protokollsuite und Allgemeiner Protokollablauf

Die Protokollspezifikationen von H.225 und H.245 beinhalten keinerlei Sicherheitsmechanismen. Um H.225 und H.245 abzusichern, wurde das Sicherheitsframework

H.235[109] spezifiziert, das für die beiden Protokolle die Autorisierung und Datenintegrität realisiert (Vgl. [50, S. 101]).

Das eben erwähnte Sicherheitsframework H.235 spezifiziert drei Sicherheitsprofile zur Absicherung der Signalisierung:

Baseline Security profile H.235.1 (*H.235 Annex D*) verwendet Passwörter, die manuell ausgetauscht werden müssen (*Shared Secrets*) ,zur Symmetrischen Verschlüsselung und ist damit aufwendig zu warten und skaliert schlecht bei steigender Anzahl an Peers. Ein Vorteil des Baseline Security Profiles ist die geringe Rechenzeit, die für die Verschlüsselung und Authentifizierung benötigt werden[109].

Signature Security Profile H.235.2 (*H.235 Annex E*) verwendet Zertifikate zur Verschlüsselung und verschlüsselt damit jede einzelne Signalisierungsnachricht Hop-by-Hop. Dadurch wird viel Rechenzeit und Bandbreite für die Verschlüsselung benötigt. Das Signature Security Profile skaliert aufgrund dieser Eigenschaft bei steigender Anzahl an Peers schlecht[109].

Hybrid Security Profile H.235.3 (*H.235 Annex F*) ist ein effizientes Sicherheitsprofil, das auch bei großer Anzahl an Peers sehr gut skaliert. Es können symmetrische Schlüssel nach dem Diffie-Hellman Verfahren ausgetauscht werden und es werden Zertifikate (in der Regel x.509 Zertifikate) und eine PKI für Asymmetrische Verschlüsselung verwendet. Das Hybrid Security Profile benötigt weniger Rechenzeit, weil es Symmetrische Schlüssel für die Verschlüsselung nutzt, wie das Baseline Security Profile. Die symmetrischen Schlüssel werden durch ein authentifiziertes (durch asymmetrische Kryptographie geschütztes) Diffie-Hellman Verfahren ausgetauscht, anstatt Shared Secrets manuell auszutauschen[110].

Aufgrund der Eigenschaften, der drei vorgestellten Security Profile, ist nur das Hybrid Security Profile von Interesse für die Implementierung unter Android. Durch das Hybrid Security Profile werden Nachrichtenauthentifizierung und Datenintegrität gewährleistet. Übermittelte Nachrichten sind nicht abstreitbar (non-repudiation). Ein Nachteil des Hybrid Security Profile ist, dass es auf eine PKI angewiesen ist.

9.3 XMPP und Jingle

XMPP (Jabber[111]) ist ein Instant-Messaging und Presence Protokoll, das sich durch eine minimale Architektur auszeichnet und dessen Funktionalität durch Erweiterungen ergänzt werden kann. Die XMPP-Erweiterung Jingle[112], ergänzt das XMPP-Protokoll um die Fähigkeit, Mediensitzungen aufzubauen (beispielsweise RTP und SRTP-Verbindungen). Der Medientransport wird außerhalb des XMPP

Protokolls abgewickelt (outband) und gesteuert und kontrolliert durch XMPP-Nachrichten (inband). XMPP ist ein verbindungsorientiertes Protokoll und verwendet TCP als Transportprotokoll. Für die Aushandlung der Verbindungsparameter und Eigenschaften der Mediensitzung, können durch XMPP/Jingle SDP Nachrichten ausgetauscht werden.

Bei XMPP und Jingle handelt es sich um ein XML-basiertes Protokoll, wodurch bedingt durch die Klartextkommunikation mehr Bandbreite als bei binärem Nachrichtenencoding benötigt wird. Da es sich bei Jingle um eine Erweiterung der Kernspezifikation von XMPP handelt, verwendet Jingle die gleichen Sicherheitsmechanismen wie XMPP.

Zur Verschlüsselung der Kommunikation wird TLS[97] verwendet und genügt damit den Anforderung an Integrität und Vertraulichkeit, allerdings werden diese nur zwischen den Instanzen in der XMPP-Kommunikationskette realisiert (*Hop-by-Hop Security*). Die Autorisierung erfolgt durch den Simple Authentication and Security Layer (SASL)[113]. Die Verwendung von TLS und SASL sind Teil der offiziellen XMPP-Spezifikation [111, Kapitel 5.3.4].

XMPP bietet zusätzlich zu den TLS und SASL Mechanismen die Möglichkeit eine Ende-zu-Ende Verschlüsselung und Nachrichtensignierung auf Basis des S/MIME Standards zu realisieren. Die Erweiterung E2E Signing and Object Encryption[114] spezifiziert die Verwendung von S/MIME durch XMPP. Mit deren Hilfe lassen sich Statusinformation, Instant-Messaging Nachrichten, sowie beliebige XMPP-Nachrichten verschlüsseln und signieren.

Eine weitere Möglichkeit Ende-zu-Ende Verschlüsselung zwischen zwei XMPP/Jingle-Clients zu realisieren ist die XMPP-Erweiterung Jingle-XTLS[115]. XTLS bietet die Möglichkeit, TCP und UDP-Nachrichten zu verschlüsseln anstatt ausschließlich TCP-Nachrichten wie TLS. Für die Realisierung der vertraulichen Ende-zu-Ende Kommunikation können beliebige XMPP-Nachrichten durch Jingle XML Kanäle getunnelt werden. Da es sich bei Jingle-XTLS um einen unfertigen Entwurf handelt der noch nicht anerkannt wurde, ist es zur Zeit nicht empfohlen diese Erweiterung zu implementieren.

9.4 MGCP, Megaco und H.248

Die Media Gateway Control Protokoll Architektur und Anforderungen [116] definieren ein Protokoll, das durch H.248.1[117] und Megaco[118] umgesetzt wird. H.248.1 (ITU-T) und Megaco (IETF) sind identische Protokolle und wurden von den beiden Organisationen unter unterschiedlichem Namen veröffentlicht. Die beiden Protokollstandards sind eine Überarbeitung des Media Gateway Control Protokolls (MG-

CP)[119] und lösen es ab.

Durch Megaco ist eine Architektur standardisiert, mit dem Media Gateway Controller (MGC) gegenseitig Signalisierungsnachrichten austauschen können. Media Gateway Controller sind Instanzen die Media Gateways (MG) kontrollieren. Media Gateways sind Instanzen zwischen einem VoIP-Kommunikationsnetz und einem klassischen Kommunikationsnetz, wie beispielsweise ISDN oder dem analogen Telefonnetz.

H.248.1/Megaco sind nicht für die Signalisierung zwischen UE und Server geeignet sondern einzig auf die Signalisierung zwischen Media Gateway Controllern beschränkt und scheiden daher für den Einsatz als Signalisierungsprotokoll für die sichere Sprachkommunikation unter Android aus. Zudem müssen Sicherheitsmechanismen extern durch IPSec, TLS oder ähnliche Mechanismen, realisiert werden.

9.5 Skinny Client Control Protocol

Das Skinny Client Control Protocol (SCCP) ist ein proprietäres Signalisierungsprotokoll und Terminalsteuerungsprotokoll, das von der Firma Cisco entwickelt wurde. Da es keine offizielle und vollständige Dokumentation gibt und bedingt durch die Tatsache, dass die Spezifikation von Cisco jederzeit geändert werden könnte, steht es außer Frage, das Protokoll SCCP in dieser Arbeit zu verwenden. Es ist hier nur der Vollständigkeit halber aufgeführt. SCCP nutzt TCP-Port 2000 für die Kommunikation und bietet keinerlei protokollinterne Sicherheitsmechanismen zur Umsetzung der Anforderungen an gesicherte Sprachkommunikation (Vgl. [120]).

9.6 Inter-Asterisk eXchange Version 2

Das Protokoll Inter-Asterisk eXchange Version 2 (IAX2)[40] ist ein kombiniertes Signalisierungs- und Medientransportprotokoll, das von der Open-Source-Community und Digium entwickelt wird. Digium ist das Hauptunternehmen hinter der Private Branch Exchange Software (PBX) Asterisk. Bei einer Private Branch Exchange Software handelt es sich um eine einer digitale Softwaretelefonanlage.

Was es von anderen vorgestellten Protokollen unterscheidet, ist dass Signalisierungsdaten und Sprachdaten nicht getrennt voneinander übertragen werden, sondern dass alles über einen gemeinsamen UDP-Kanal transportiert wird. Durch den gemeinsamen Transport von Sprach- und Signalisierungsdaten über einen logischen Kanal, können IAX2-Verbindung einfacher administriert werden. Ein weiterer Aspekt, der IAX2 beispielsweise von SIP unterscheidet, ist die Minimierung des Protokollover-

heads durch binäres Kodieren des Protokollheaders, wie es auch von H.323 standardisiert ist. Dies ermöglicht eine effizientere Bandbreitennutzung.

Die Sicherheitsmechanismen, die durch IAX2 definiert sind, sind unzureichend. Zwar wird die Vertraulichkeit und Integrität der Sprachkommunikation gewahrt durch AES-Verschlüsselung der Sprachdaten und die Autorisierung erfolgt durch ein Hashbasiertes Challenge-Response-Verfahren. Die Anrufdetails und Verbindungsdaten werden aber stets unverschlüsselt übertragen. Kontrollkommandos werden zudem bei IAX2 nicht gegen Veränderung geschützt, wodurch ein Angreifer die Möglichkeit erhält, Sprachverbindungen zu beenden oder zu transferieren.

Abschließend betrachtet ist das IAX2-Protokoll aufgrund der Sicherheitsmängel und bedingt durch die Tatsache, dass es nicht standardisiert ist, ungeeignet, abhörsichere Sprachkommunikation zu gewährleisten.

9.7 Fazit

Als Signalisierungsprotokoll wird in der Anwendung zur sicheren Sprachkommunikation das Session Initiation Protokoll eingesetzt. Die einzigen wirklich konkurrenzfähigen Protokolle zu SIP sind H.323 und XMPP mit Jingle. Die übrigen vorgestellten Protokolle sind entweder aus technischen Gründen wie MGCP (nur für Multimedia Gateways) oder aufgrund ihres proprietären Charakters wie IAXv2 und SCCP ungeeignet für die Signalisierung unter Android.

SIP wird eingesetzt da es im Gegensatz zu XMPP/Jingle das erprobtere Protokoll ist. Jingle ist weiterhin von der XMPP Standards Foundation als Entwurf gekennzeichnet.
H.323-SIG wird nicht verwendet da es die Komplexität der Protokollsammlung unattraktiv für den Einsatz als Signalisierungsprotokoll macht. SIP besticht in der Protokollarchitektur durch Simplizität. Vergleicht man SIP und H.323 in den Punkten der technischen Eignung und der Erfüllung der Sicherheitsanforderungen, stehen sich die beiden Protokolle in nichts nach.

SIP ermöglicht die Ende-zu-Ende Absicherung durch S/MIME und die Hop-by-Hop Absicherung durch SIPS (SIP mit TLS). In der Referenzimplementierung wird auf eine Absicherung des Signalisierungsprotokoll verzichtet, da aufgrund des ausgewählten Key Management Protokolls keine Absicherung des Signalisierungsprotokolls notwendig ist (siehe 10.3).

10 Key Management

Die Schlüssel die zur Verschlüsselung der SRTP Pakete verwendet werden, müssen zwischen den Kommunikationspartnern ausgetauscht werden. Da das Secure Real-Time Protokoll keine Möglichkeiten dazu spezifiziert, werden in diesem Kapitel Key Management Protokolle evaluiert. Zunächst werden allgemeine Anforderungen an das Key Management erläutert und anschließend verschiedene Key Management Protokolle untersucht.
In Kapitel 10.9 wird auf die Verwendung von Crypto-Smartcards zur Verschlüsselung und Zertifikatsspeicherung eingegangen.

10.1 Key Management Protokolle

Durch die Verwendung einer weiteren Protokollschicht, der Key Management Ebene, wird neben der Signalisierungsebene und der Medientransportebene eine dritte Schicht im Protokollstack eingeführt. Eine Aufgabe des Key Management Protokolls ist es diese drei Ebenen aneinander zu binden (Binding). Mit dem Binden von Signalisierungs- und Medientransportebene aneinander wird vermieden, dass eine signalisierte Medienverbindung von Dritten übernommen werden kann. Zur Authentifizierung wird meist nach dem Aufbau von Signalisierungs- und Medienkanal überprüft, dass beide Kanäle von dem selben Peer stammen. Man unterscheidet dabei zwischen *Early Binding* und *Late Binding*, abhängig davon zu welchem Zeitpunkt das Binding geschieht.
Beim Early Binding werden im Signalisierungsprotokoll Information über die spätere Medienverbindung ausgetauscht um diese später zu identifizieren und verifizieren.
Beim Late Binding findet die Authentifizierung des Kommunikationspartner und das Binding erst nach dem Aufbau der Medienverbindung statt.

In Kapitel 8 wurde das Protokoll (S)RTP als Medientransportprotokoll ausgewählt, daher werden in diesem Kapitel ausschließlich Key Management Protokolle für SRTP evaluiert. Key Management Protokolle für SRTP lassen sich in zwei Kategorien unterteilen. Zum einem existieren Key Management Protokolle die die Schlüssel über das Signalisierungsprotokoll austauschen (z.B. SDES), zum anderen realisieren einige Key Management Protokolle die Schlüsselverwaltung über das Medientransport-

protokoll RTP (z.B. ZRTP).

Eine weitere Aufgabe des Key Management Protokolls ist es Man-in-the-Middle Attacken wirksam zu verhindern, indem beispielsweise basierend auf kryptographischen Verfahren ein Short Authentication String (SAS) von beiden Kommunikationspartner vorgelesen wird, der im Falle eines Man-in-the-Middle nicht bei beiden identisch ist (z.B. bei ZRTP).

Die Bewertung der Key Management Protokolle geschieht in diesem Kapitel unter Berücksichtigung der unter Kapitel 4 definierten Anforderungen, sowie der in diesem Kapitel vorgestellten Kriterien[121].

10.2 TESLA-SRTP

TESLA steht für *Timed Efficient Stream Loss-Tolerant Authentication[122]* und ist ein Protokoll, um die Authentizität und Datenintegrität von SRTP Paketen zu gewährleisten. TESLA ergänzt SRTP um die Möglichkeit in Gruppen (beispielsweise Multicastgruppen) die Datenintegrität und Datenzuordnung zu einem bestimmten Sender sicherzustellen (*Data Origin Authentication*).
TESLA realisiert dies anstatt mit asymmetrischer Kryptografie, mit symmetrischen Kryptografie-Verfahren um die Data Origin Authentication weniger rechenintensiv zu realisieren. Dazu wird durch das TESLA Protokoll zeitverzögert der symmetrische Schlüssel mit dem kurze Zeit zuvor das versendete Datenpaket authentifiziert wurde an die Empfänger der Daten verschickt. Dadurch dass der Schlüssel kurz nach dem Datenpaket eintrifft, hat ein Angreifer keine Möglichkeit das Paket oder den Schlüssel zu fälschen, weil dadurch sonst das Timing der eintreffenden Pakete verändert werden würde. TESLA benötigt daher eine lose Uhrensynchronisation der Kommunikationsteilnehmer im Bereich von wenigen Millisekunden Genauigkeit um diese Funktion zu realisieren.

TESLA-SRTP benötigt neben einem externen Key Management Protokoll für die Verteilung der Schlüssel, noch weitere Protokolle für das Bootstrapping einer TESLA-Sitzung und die damit verbundene Zeitsynchronisation zwischen den Kommunikationsteilnehmern. TESLA spezifiziert keine Mechanismen zur Schlüsselverwaltung, sondern bietet nur ein kryptografisches Verfahren um Rechenzeit bei der Verschlüsselung von SRTP Paketen einzusparen. Da dies bei mobilen Endgeräten zum Tragen kommt wird TESLA hier dennoch erwähnt.

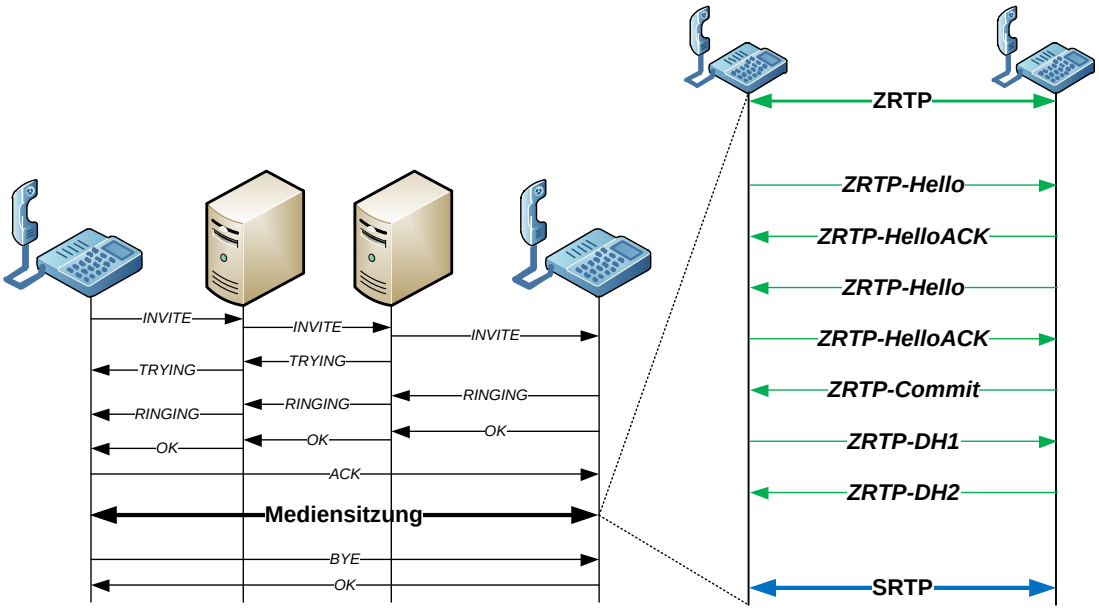

Abbildung 10.1: ZRTP Protokoll Handshake (Vgl. [124])

10.3 ZRTP

ZRTP[123] (Media Path Key Agreement for Unicast Secure RTP) ist ein Protokoll zum sicheren Schlüsselaustausch über das Medientransportprotokoll RTP, um die für SRTP notwendigen kryptografischen Parameter und Schlüsselinformationen auszutauschen.

Der ZRTP Protokollablauf ist durch den Austausch von sehr wenigen ZRTP Nachrichten abgewickelt. Es sind minimal zehn ZRTP-Nachrichten notwendig bis die verschlüsselte Kommunikation über SRTP aktiviert werden kann. Durch die geringe Anzahl an Nachrichten wird zudem die benötigte Bandbreite im Medientransportkanal nur minimal erhöht, was bei der mobilen Datenkommunikation von Vorteil ist. Der ZRTP-Protokollablauf ist in Abbildung 10.1 dargestellt. Ein mitgeschnittener ZRTP-Handshake eines Netzwerkprotokollanalysators ist in Anhang D abgebildet. ZRTP-Hello werden inband im gleichen logischen Netzwerkkanal wie RTP-Pakete übertragen. Durch sie wird die Fähigkeit eines Peers zum Verstehen des ZRTP-Protokolls signalisiert, einschließlich der unterstützten Verschlüsselungsverfahren. Sobald die Kommunikationspartner sich mit einer HelloACK Nachricht die gegenseitige ZRTP-Unterstützung bestätigt haben, beginnt der Diffie-Hellman Schlüsselaustausch.

ZRTP standardisiert Verfahren, die keinerlei Sicherung der Integrität und der Vertraulichkeit auf der Ebene der Signalisierung benötigen. ZRTP ist durch den Verzicht auf Zertifikate nicht auf eine PKI oder eine andere externe Zertifikatsverwaltung angewiesen, obwohl diese bei Bedarf optional verwendet werden können. Falls kein

Short Authentication String benutzt wird, muss beim ersten Verbindungsaufbau zwischen zwei Kommunikationspartnern ein integritätsgeschützer Signalisierungskanal verwendet werden. ZRTP ist unabhängig von der Signalisierungsebene und kann in Kombination mit einem beliebigen Signalisierungsprotokoll eingesetzt werden.

Beim SAS handelt es sich um eine Sicherheitsfunktion, durch die sich zwei Kommunikationspartner gegenseitig authentifizieren können und Man-in-the-Middle Attacken erkannt werden können. Die Funktionsweise des SAS ist simpel und effektiv. Basierend auf dem öffentlichen Diffie-Hellman Parameter des Initiators und der ZRTP-Hello Nachricht des Antwortenden wird ein kurzer String berechnet, den sich die beiden Kommunikationspartner gegenseitig über die Audioverbindung vorlesen. Sollte ein MitM-Angriff vorliegen, wären die beiden SAS nicht identisch.
Angriffe auf die SAS-Methode sind nur von theoretischer Natur und setzen einen Stimmimitator oder aufgezeichnete Sprache eines der Teilnehmer voraus. Zusätzlich ist nicht bekannt mit welchem Timing und auf welche Art die Kommunikationspartner den SAS vorlesen. Die Verwendung des SAS-Mechanismus ist zudem auch nur bei der initialen Kontaktaufnahme zweier Kommunikationspartner notwendig. Sobald die Identität einmal mittels des SAS verifiziert wurde, können die Identität und die Abwesenheit eines MitM mit Hilfe des Key Continuity Features (wie bei SSH[55]) gewährleistet werden (baby-duck Modell[125] oder resurrecting-duckling Modell[126]). Dafür werden vom jeweils letzten Anruf, Teile des Schlüssel gesichert um bei der Schlüsselberechnung des darauf folgenden Gesprächs mit verrechnet zu werden.

Für jede Sitzung wird bei ZRTP ein eigener Sitzungsschlüssel berechnet und am Ende der Sitzung wieder gelöscht. Sollte ein Schlüssel kompromittiert werden, kann damit nicht die aufgezeichnete vorhergehende Sitzung entschlüsselt werden und insbesondere nicht alle jemals mit einem bestimmten Schlüssel abgewickelten Gespräch (*Perfect Forward Secrecy*).

ZRTP ist damit eine sehr gute Möglichkeit Authentifizierung und die Schlüsselvereinbarung ohne die Notwendigkeit einer PKI umzusetzen. Die Abwesenheit praktisch umsetzbarer Angriffe und die geringe Anfälligkeit gegen DoS-Attacken machen sie zudem den übrigen Key Management Protokollen im Bereich der Ende-zu-Ende Absicherung von zwei Kommunikationsteilnehmern überlegen[127].

Laut [128] hat die Verwendung von ZRTP eine vernachlässigbar geringen Einfluss auf die QoS Parameter Jitter und Delay, da nur sehr wenige ZRTP Pakete ausgetauscht werden müssen, bis die Schlüssel für die Verschlüsselung mit SRTP ausgetauscht sind.

Dadurch hat ZRTP keinen negativen Einfluss auf die Anforderungen an Digitale Sprachkommunikation (Kapitel 4.1). Durch die Möglichkeiten, den Diffie-Hellman

Schlüsselaustausch nach dem ECDH-Verfahren durchzuführen, lässt sich ZRTP auch für mobile Endgeräte effizient implementieren.

10.4 MIKEY

Das Multimedia Internet KEYing[129] (MIKEY) Key Management Protokoll operiert in der Signalisierungsebene. MIKEY kann zur Schlüsselverwaltung für *one-to-one*, *one-to-many* und *many-to-many* Kommunikation eingesetzt werden
Zu den Designmerkmalen von MIKEY zählen dabei die Ende-zu-Ende Sicherheit (im Gegensatz zur Hop-by-Hop Sicherheit) und die Einfachheit des Protokolls. MIKEY zeichnet sich durch geringen Bandbreitenverbrauch und geringen Rechenaufwand aus. Das Key Management Protokoll MIKEY wurde so entwickelt, dass es in anderen Protokollen wie RTSP oder SDP getunnelt zu transportieren ist und unabhängig von den Sicherheitsmechanismen der unteren Protokollschichten ist.

MIKEY unterstützt mehrere Schlüsselaustausch-Modi, zum einen das bekannte Diffie-Hellmann Schlüsselaustausch Verfahren (MIKEY-DH, MIKEY-DHMAC), zum anderen Schlüsseltransportverfahren, bei denen die Schlüssel direkt ausgetauscht werden (MIKEY-PSK, MIKEY-RSA, MIKEY-RSA-R).

Das MIKEY-Protokoll generiert einen einzigartigen Sitzungsschlüssel pro Sitzung, unter anderem anhand des Signaling Source Feldes der RTP-Pakete und speichert diese in einer Signaling Source Map. Der MIKEY Protokollhandshake kann in einer Runde abgewickelt werden. Dabei werden zwischen den Kommunikationsteilnehmern die Signaling Source Map und weitere kryptografische Parameter, wie Algorithmen und Schlüssellänge, ausgetauscht. Das Binding der Signalisierungs- und Key Management Ebene erfolgt mit Hilfe des SIP Uniform Resource Identifiers. Aufgrund des Early Bindings werden DoS-Attacken vermieden, die auf dem (abgebrochenen) Sitzungsaufbau durch unautorisierte Teilnehmer basieren. Die bedeutet dass Kommunikationsteilnehmer authentifiziert werden bevor die rechenintensive Schlüsselberechnumg durchgeführt wird (Vgl. [127]).

Gegenseitige Authentifizierung ermöglicht MIKEY entweder durch Pre-Shared Keys (PSK) oder durch digitale Zertifikate. Dies ist auch einer der offensichtlichen Nachteile von MIKEY, da zwei Kommunikationspartner nicht ohne Administrationsaufwand *authentifiziert* miteinander kommunizieren können. Bei der Verwendung von PSKs kann zudem keine Perfect Forward Secrecy mehr gewährleistet werden, die man bei den Diffie-Hellmann Schlüsselaustauschverfahren hat. Der Vorteil der PFS bringt auf mobilen Endgeräten die Nachteile mit sich, dass der Bandbreitenverbrauch durch den Schlüsseltransport und die Rechenzeit zur Schlüsselberechnung steigt (im Vergleich mit PSKs).

Aufgrund des fehlenden Zufallsfaktors bei der Schlüsselberechnung, kann MIKEY nicht als kryptografisch sicher bewiesen werden, da die Verwendung eines Zufallswertes essentiell für den Beweis der kryptographischen Sicherheit ist (Vgl. [104, S.10]).

10.5 DTLS-SRTP

DTLS-SRTP ist ein Framework, um basierend auf DTLS das Key Management für das SRTP Protokoll durchzuführen und wurde 2010 von der IETF standardisiert[130].
Eine der herausragendsten Eigenschaften von DTLS-SRTP ist, dass zwei nicht in Beziehung stehende Kommunikationspartner den Schlüsselaustausch vollziehen können, ohne dass dafür alle bei der Signalisierung beteiligten Elemente geschützt werden müssten oder eine PKI vonnöten wäre. Mit Hilfe von selbst signierten und selbst generierten Zertifikaten, die über eine separate DTLS-Verbindung (Key Management Ebene) ausgetauscht werden, wird dies bei DTLS-SRTP realisiert. Mit Hilfe des in der Signalisierungsebene übertragenen Fingerprints wird das Zertifikat, das mittels Datagrammen (UDP oder DCCP) übertragen wurde, verifiziert. Mit Hilfe von Sicherungsmaßnahmen in der Signalisierungsebene wie SIPS (Hop-by-Hop Security), der SIP-Identity Protokollerweiterung oder S/MIME muss der Fingerprint in der SDP-Nachricht vor unbemerkter Modifikation geschützt werden (*Integritätssicherung*). Vertraulichkeit wird nicht benötigt. Zur Schlüsselvereinbarung kommt bei DTLS-SRTP das Diffie-Hellman Verfahren zum Einsatz.

Mittels des Diffie-Hellman Schlüsselaustauschmechanismus realisiert DTLS-SRTP Perfect Forward Secrecy. Das Diffie-Hellman Verfahren kann auch mittels elliptischer Kurven durchgeführt werden, wodurch es auf mobilen Endgeräten effizient implementiert werden kann. DTLS-SRTP bindet die Medientransportebene spät an die Signalisierungsebene (Late Binding), da im Vorfeld keine Session Source Identifier der RTP-Sitzung über das Signalisierungsprotokoll ausgetauscht werden können. Um den Austausch der Schlüssel der Peers zu ermöglichen, um dadurch beispielsweise Schlüssel für die Gruppenkommunikation unter allen Teilnehmern auszutauschen, wurde DTLS-SRTP durch ein zusätzliches RFC aktualisiert[131] .

Durch die Tatsache, dass die Fingerprints der Zertifikate auf der Signalisierungsebene ausgetauscht werden, wird unbedingt ein Externer Authentifizierungsmechanismus für die Kommunikationspartner benötigt (Vgl. [127]).

Laut [127] ist DTLS-SRTP anfällig gegen zahlreiche DoS-Attacken und schneidet bei der Sicherheitsanforderung *Verfügbarkeit* damit schlechter ab, als beispielsweise ZRTP (Kapitel 10.3) oder MIKEY (Kapitel 10.4). Insbesondere DTLS-SRTP Clients sind anfälliger für DoS-Attacken. Eine Möglichkeit ist beispielsweise das Late

Listing 10.1: SDES Tag in SDP Nachrichten

```
a=crypto:<tag> <crypto-suite> <key-params> [<session-params>]
```

Binding in der Form wie es bei DTLS-SRTP zum Einsatz kommt. Dadurch müssen nicht-authentifizierte SRTP-Nachrichten entschlüsselt werden, bevor sie verworfen werden können.

10.6 GDOI-SRTP

The *Group Domain of Interpretation* - SRTP (GDOI-SRTP)[132] ist ein Key Management Protokoll mit dem Fokus auf Schlüsselverwaltung für Gruppen, wie zum Beispiel Multicastgruppen. GDOI-SRTP tauscht, genau wie ZRTP oder DTLS-SRTP die Schlüssel über das Medientransportprotokoll RTP aus. Es adaptiert hierfür Funktionen des *Internet Security Association and Key Management Protokolls* (ISAKMP)[133], das bei IKE[134] und IPsec[94] Verwendung findet. Aufgrund der Tatsache dass GDOI-SRTP nur als Internet-Draft der IETF (RFC) vorliegt und seit Juni 2008 abgelaufen ist, kann es zur Zeit nicht als Key Management Protokoll empfohlen werden. Zudem basiert GDOI-SRTP auf ISAKMP, das mittlerweile ebenfalls veraltet ist und in IKEv2[103] aufgenommen und abgelöst wurde.

10.7 IKE

IKE steht für Internet Key Exchange[134] und ist das Key Management Protokoll in dem Protokollpaket von IPsec. Für das Schlüsselmanagement von SRTP ist es ungeeignet, da kein Bezug zu SRTP-spezifischen Parametern genommen werde kann, wie beispielsweise dem Signaling Source Identifiern der RTP Pakete.

10.8 SDES

Security Descriptions for Media Streams[135], oder kurz SDES ist ein Key Management Protokoll, das den Schlüsselaustausch mit Hilfe von SDP-Nachrichten in der Signalisierungsebene vollzieht. Dazu führt SDES einen neuen Tag in SDP-Nachrichten ein:

SDES benötigt den Schutz der Integrität und Vertraulichkeit in der Signalisierungsebene, da das SDES Protokoll ohne externe Absicherung die Schlüssel ungesichert und nicht authentifiziert zwischen den Kommunikationsendpunkten austauscht. SDES-Nachrichten müssen deshalb durch ein geeignetes Verfahren wie TLS

oder S/MIME geschützt werden. Die Ende-zu-Ende Vertraulichkeit beim Schlüsselaustausch mit SDES ist ausschließlich durch S/MIME zu erreichen (Vgl. [127]).
Die Verwendung von S/MIME für verschlüsselten Ende-zu-Ende Transport von Schlüsselmaterial führt jedoch dazu, dass ein separater Schutz vor Replay Attacken in der Anwendung integriert werden muss, wie beispielsweise eine lose Uhrensynchronisation (Vgl. [104]).

SDES benutzt einen Late Binding Mechanismus, um die Signalisierungsebene, die Medientransportebene sowie den Kryptografischen Kontext der Key Management Ebene miteinander zu verknüpfen. Die Verknüpfung wird anhand der Signaling Source Werte der SRTP-Pakete identifiziert. Ein erneutes Binding der Ebenen wird auf der Medientransportebene durchgeführt. Dadurch wird der Angriffsvektor für eine DoS-Attacke im Vergleich zu einem vollständigen Binding in der Signalisierungsebene vergrößert, weil ankommende SRTP-Nachrichten mit unbekannten SSRC-Werten immer entschlüsselt werden müssen, bevor sie verworfen werden können.

10.9 Der Einsatz von Crypto-Smartcards unter Android

Crypto-Smartcards wie die *certgate SmartCard microSD*[136] von Certgate[137] können unter Android eingesetzt werden, um kryptographische Berechnungen zu unterstützen, beschleunigen und die Sicherheit zu erhöhen[138].

Die *certgate SmartCard microSD* hat eine Vielzahl von kryptographischen Algorithmen in Hardware auf der microSD Karte implementiert und kann dadurch diese kryptographischen Berechnungen schneller und sicherer ausführen als es in Software auf dem Android OS möglich ist [139].

Der verbaute Chipsatz auf der microSD Karte ist der P5CC072[140] von NXP[141] (Philips) der unter anderem folgende Berechnungen durchführen kann:

PKI co-processor zur Beschleunigung der RSA-Berechnung[142] und Berechnung von Elliptischen Kurven (ECC)[143].

DES-3 co-processor zur Beschleunigung der Dual- und Triple-DES[144, S.309] Schlüsselberechnung.

AES co-processor zur Beschleunigung der AES[74] Berechnung (wie er bei SRTP und ZRTP Verwendung findet).

Zufallszahlengenerierung nach BSI-AIS31[145] Standard.

Die Sicherheit der kryptographischen Berechnungen wird dadurch erhöht, dass der NXP P5CC072 Schutz vor Sidechannel-Attacken, wie der Differential Power Analysis[52] oder vor Timing-Attacken[53] bietet und dafür auch vom Bundesamt für

Sicherheit in der Informationstechnik zertifiziert wurde unter [146].

Auf der Crypto-Smartcard können Private Schlüssel gespeichert werden, wodurch Schlüssel nicht ausgelesen oder abgehört werden können, da sie nur innerhalb der Smartcard verwendet und gespeichert werden[136].

Abschließend betrachtet sind Crypto-Smartcards eine Möglichkeit die Sicherheit von kryptographischen Verfahren und Algorithmen unter Android zu erhöhen. Das in dieser Thesis keine Crypto-Smartcards zum Einsatz kommen liegt darin begründet, dass keine Crypto-Smartcard zu Evaluationszwecken beschafft werden konnte.

10.10 Fazit

Für das Key Management des Secure Real-Time Protokolls wird ZRTP eingesetzt. ZRTP bietet ohne auf eine PKI angewiesen zu sein, Schutz vor Man-in-the-Middle Angriffen, Perfect Forward Secrecy und ist nicht auf die Absicherung des Signalisierungsprotokoll angewiesen, da die Authentifizierung und der Schlüsselaustausch vollständig im selben Kommunikationskanal wie das Medientransportprotokoll RTP abgewickelt wird.

ZRTP kann ein beliebiges Signalisierungsprotokoll verwenden. Durch die Verwendung des Sicherheitsfeatures Short Authentication String und der Key Continuity Mechanismen werden Man-in-the-Middle Angriffe wirksam verhindert. Dadurch dass der Schlüsselaustausch nach dem Diffie-Hellman Verfahren durchgeführt wird, ist ein Schlüsselaustausch ohne den vorherigen Kontakt zweier Kommunikationspartner möglich.
ZRTP ist das einzige der vorgestellten Key Management Protokolle, das alle hier aufgelisteten Sicherheitsfeatures abdeckt. Der Vergleich der in diesem Kapitel vorgestellten Key Management Protokolle wird in Tabelle 10.1 nochmals übersichtlich dargestellt.

Protokoll	**Protokollschicht**	**Binding, DoS-Verstärkung**	**Perfect Forward Secrecy**	**Gegenseitige Authentifizierung / MitM Schutz**	**Signalisierungsprotokoll**
SDES	Signalisierungsebene	Late Binding, DoS-Verstärkung	nein	Durch andere Protokolle	SDP-basierende
MIKEY	Signalisierungsebene	Early Binding	nein (PSK & PKI), ja im DH-Modus	Zertifikate (PKI)	SIP, H.323
ZRTP	Medientransportebene	Late Binding, keine DoS-Verstärkung	ja	SAS bei erstem Kontakt, baby-duck Key Continuity danach	SIP, Jingle, H.323, u.a. (beliebiges Signalisierungsprotokoll)
DTLS-SRTP	Medientransportebene	Late Binding, DoS-Verstärkung	ja (DH-Modus), nein (bei gespeicherten Zertifikaten)	Abhängig von der Signalisierung (nur durch PKI), SAS (nur durch TLS-Erweiterung)	SIP
GDOI-SRTP	Medientransportebene	%	%	SIP-Identity	SDP-basierende

Tabelle 10.1: Key Management Protokolle im Vergleich (Vgl. [88, erweitert])

11 Übersicht über die Evaluation

In der vorliegenden Evaluation der Protokolle und Technologien zur sicheren Signalisierung, dem Medientransport und der Schlüsselverwaltung wird eine Vielzahl von Protokollen und Technologien betrachtet und bewertet.

In Abbildung 11.1 ist das Zusammenspiel der einzelnen Protokoll in der Anwendung zur sicheren Sprachkommunikation dargestellt. Das Session Initiation Protokoll wird zur Signalisierung verwendet, der verschlüsselte Medientransport wird durch das Secure Real-Time Protokoll realisiert und das für SRTP notwendige Key Management wird durch ZRTP übernommen.

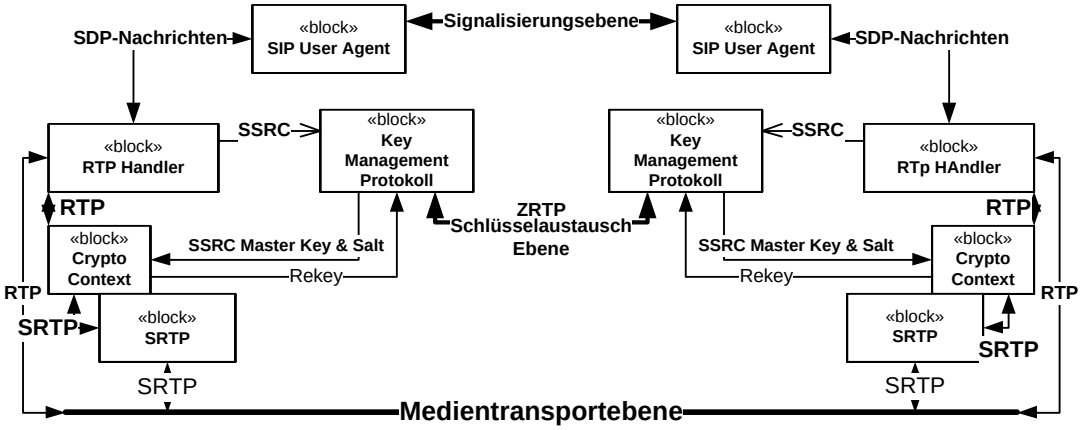

Abbildung 11.1: Abhörsicheres VoIP (Überblick) (Vgl. [88, modifiziert])

Die Technologien der drei Ebenen Medientransport, Signalisierung und Key Management wurden einander gegenübergestellt und bewertet. Mit Hilfe eines URN Diagramms wird dies in diesem Kapitel nochmals übersichtlich in Abbildung 11.2 dargestellt. Die Eignung der einzelnen Technologien für die Realisierung einer der drei Ebenen, wird mit der bekannten Symbolik der URN Modellierungssprache graphisch bewertet.

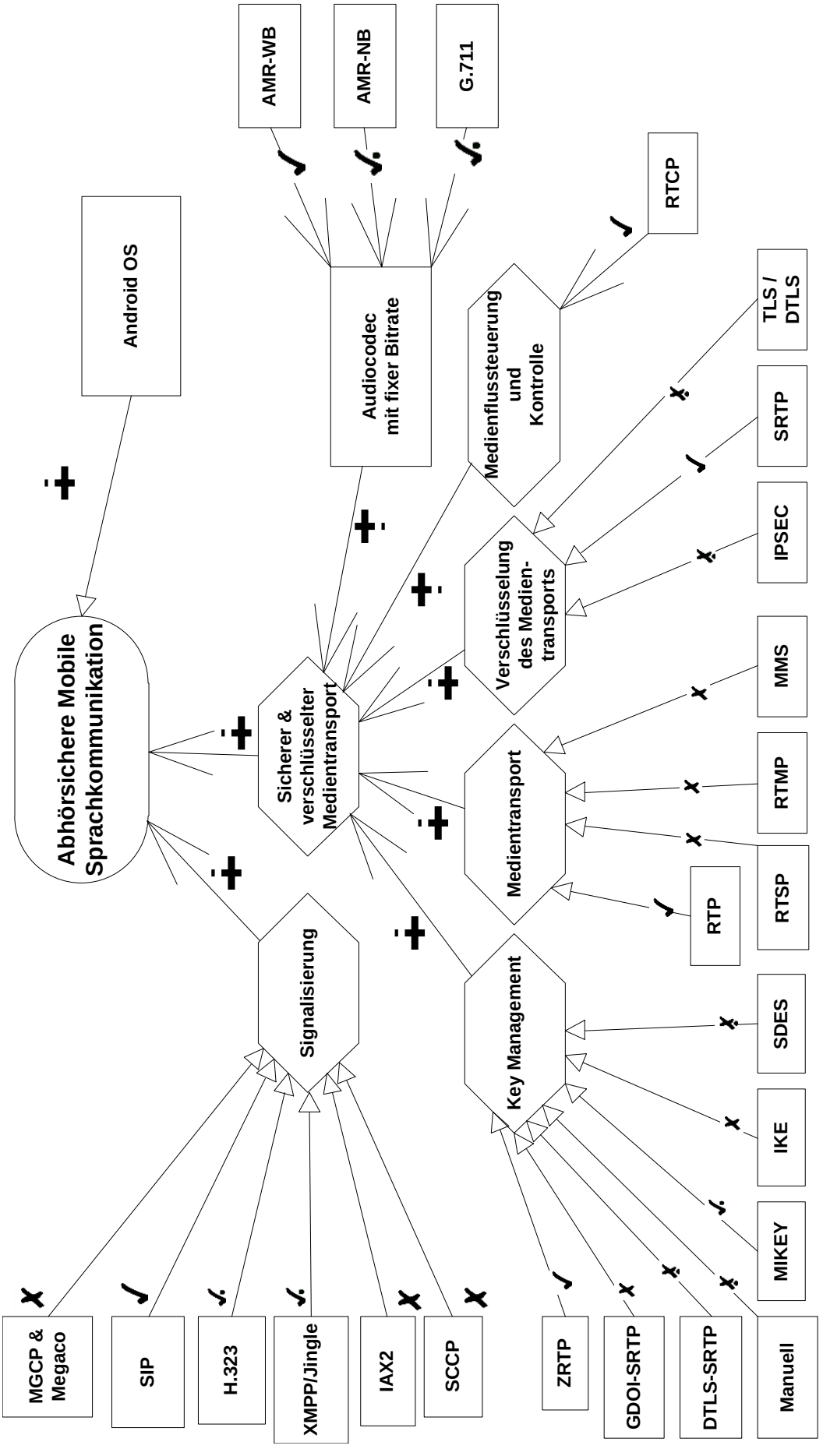

Abbildung 11.2: Anforderungsanalyse alle Protokolle

Teil III

Implementierung

12 Konzept

In diesem Kapitel wird das Konzept dargelegt, auf dessen Grundlage die Implementierung der Anwendung zur verschlüsselten Sprachkommunikation unter Android realisiert wird. Es wird dabei auf die Auswahl der Bibliotheken eingegangen mit deren Hilfe das Konzept umgesetzt werden soll, es wird die Architektur erläutert die implementiert werden soll und es wird auf die Besonderheiten eingegangen die man bei der Umsetzung unter Android beachten muss. Dies geschieht unter Beachtung des Anforderungskataloges der unter Kapitel 4 definiert wurde.

Um auf die Referenzimplementierung im Text leichter Bezug nehmen zu können, wird die mobile Anwendung zur verschlüsselten Sprachkommunikation auf Basis des Android Betriebssystems auch mit dem Projektnamen *MemhoPhone* referenziert, der Kurzform für *MemoryHolePhone*. Als Memoryhole wird ein Konzept aus dem Roman 1984[147] von George Orwell bezeichnet, mit dem Dokumente, Fotografien und alle Medien der Zeitgeschichte im Allgemeinen vernichtet werden können, als ob sie nie existiert hätten:

> A memory hole is any mechanism for the alteration or disappearance of inconvenient or embarrassing documents, photographs, transcripts, or other records, such as from a web site or other archive, particularly as part of an attempt to give the impression that something never happened [148].

Die Diagramme im Implementierungsteil wurden mit Hilfe der Modellierungssprache SysML erstellt, die in Anhang B vorgestellt wird.

12.1 Auswahl der Bibliotheken

Um die ausgewählten Protokolle und Technologien in einer mobilen Anwendung zur verschlüsselten Sprachkommunikation auf Basis des Android Betriebssystems verwenden zu können, muss zunächst für möglichst alle Protokolle eine existierende Implementierung als Bibliothek gefunden werden. Dies beschleunigt den Implementierungsprozess, verringert den Implementierungsaufwand und führt zu geringeren Kosten für die Anwendungsentwicklung. Dies bedeutet für die Auswahl der Bibliotheken auch, dass sie möglichst unter eine BSD-Lizenz zur Verfügung stehen, die

eine spätere kommerzielle Verwertung der Anwendung, ohne eine Veröffentlichung des Quellcodes, zu ermöglichen.

12.1.1 Audiocodec

Als Audiocodec kommt AMR-WB zum Einsatz, wie unter Kapitel 7 begründet wurde. Mittels der Android SDK Java-API ist das Live-Transcoding im Speicher mit diesem Audiocodec nicht möglich. Man muss zum Transcoding den Flashspeicher des Mobiltelefons verwenden. Unter Verwendung des im Android NDK mitgelieferten Audioverarbeitungsframeworks OpenSL ES ist ein Transcoding der Audiodaten von und nach AMR-WB im Speicher möglich und wird von dessen API, die in C geschrieben ist, unterstützt.

Die Verwendung von Android SDK und NDK Komponenten ist gegenüber der Verwendung von externen Bibliotheken vorzuziehen, da dadurch die Chance von Kompatibilitätsproblemen verringert wird. Daher habe ich mich für den Einsatz von OpenSL ES mittels des Android NDK entschieden. Insbesondere da dadurch die Lizenzierungsgebühren für die Verwendung des AMR Codecs durch das Android OS abgedeckt sind.

Diese Entscheidung hat zur Folge, dass die RTP und SRTP Bibliothek wenn möglich, ebenfalls in C oder C++ implementiert sind um einen Datenaustausch zwischen NDK und SDK Schicht zu vermeiden. Andernfalls würden zusätzliche Latenzen bei der Audioverarbeitung auftreten, die es zu vermeiden gilt (dies wurde in Kapitel 4.1 dargelegt).

12.1.2 Medientransportprotokoll RTP und SRTP

Bei den möglichen (S)RTP Programmbibliotheken fällt aus der unter Tabelle 12.1 dargestellten Auswahl an evaluierten RTP-Bibliotheken, die *ccrtp* Bibliothek weg. Sie bietet zwar SRTP Support und eine einfache ZRTP Integration mittels einer Zusatzbibliothek, sie ist allerdings unter der GPL lizenziert, was eine kommerzielle Vermarktung des Produktes erschwert.

Die Bibliotheken *jlibrtp* und *jrtplib* sind weniger gut geeignet für die Aufgabe des sicheren Medientransports, da eine Verschlüsselung und Integration des SRTP-Profils in diese Bibliotheken aufwendig nachgerüstet werden müsste.

Die Bibliothek *ortp* bietet mittels der zusätzlichen Bibliothek *libsrtp* volle Unterstützung des RTP Protokolls, sowie des SRTP Profils. Wenn *ortp* mit SRTP Unterstützung und verlinkter *libsrtp* kompiliert wurde, kann die sichere Kommunikation unter Android damit umgesetzt werden. Ein kleiner Nachteil der *ortp* Bibliothek ist,

dass man sie mit Hilfe des Android NDKs verwenden muss, da sie in C implementiert ist. Dieser Umstand ist für den optimalen Transport der Audiodaten, nur auf Android NDK Ebene, wiederum einen Vorteil. Die Lizenzierung unter LGPL ermöglicht zudem eine problemlose kommerzielle Vermarktung eines späteren Produktes. In der Bibliothek libsrtp sind die kryptographischen Algorithmen zur Verschlüsselung mit dem Algorithmus AES im Cipher Block Chaining Mode und im Integer Counter Mode bereits integriert (weitere Erläuterungen zu AES, Blockchiffren und Operationsmodi befinden sich im Anhang C).

Eine fünfte Möglichkeit um den sicheren Medientransport unter Android zu realisieren ist die Bibliothek *FMJ* (*Freedom for Media in Java*). Sie bietet SRTP Unterstützung und eine leichte Integration des ZRTP Protokolls. Da sie in Java implementiert ist, kann man sie unter Android auch ohne größeren Aufwand verwenden. Da der *AMR-WB* Audiocodec durch das Android NDK benutzt werden soll, hat die Verwendung von *FMJ* allerdings den entscheidenden Nachteil dass ein latenzinduzierender Datenaustausch zwischen Android SDK und NDK Ebene stattfinden würde.

Ich habe mich für *ortp* in Verbindung mit *libsrtp* entschieden, da sie in Kombination mit OpenSL ES einen optimierten Sprachdatenfluss bietet, ohne der Notwendigkeit des Datenaustausches zwischen der C und Java Schicht.

Bibliothek	Sprache	Lizenz	Anmerkung	Quelle
jlibrtp	Java	LGPL	SRTP & ZRTP Integration umständlich nachzurüsten	http://sourceforge.net/projects/jlibrtp
jrtplib	C++	MIT	SRTP & ZRTP Integration umständlich nachzurüsten	http://research.edm.uhasselt.be/~jori/page/index.php?n=CS.Jrtplib
ortp	C	LGPL	Einfache libsrtp Integration; einfache ZRTP Integration, STUN	http://linphone.org/eng/documentation/dev/ortp.html
ccrtp	C++	GPL	RTP, SRTP integriert; einfache ZRTP Integration	http://gnu.org/software/ccrtp
FMJ	Java	LGPL	RTP, SRTP integriert; einfache ZRTP Integration	http://fmj-sf.net
libsrtp	C	BSD	SRTP Bibliothek; Einfache ZRTP Integration; benötigt zusätzlich RTP Bibliothek	http://srtp.sourceforge.net/srtp.html

Tabelle 12.1: RTP Bibliotheken (Vergleich)

12.1.3 Signalisierungsprotokoll SIP

Aufgrund der Einschränkung die durch die gewünschte Lizenz gegeben ist, bleiben von der möglichen Auswahl an Bibliotheken, die in Tabelle 12.2 dargestellt sind, nur drei mögliche Bibliotheken zum weiteren Vergleich übrig. Dies wären *J-SIP*, *Sofia-SIP*, sowie der *SIP-Stack*, der unter Android 2.3.4 integriert ist.

Bei der Verwendung des SIP-Stacks von Android 2.3.4 ist eine externe Absicherung der SIP-Kommunikation über TLS nicht möglich. Dies ist zwar nicht zwingend notwendig um mittels SRTP und ZRTP sicher zu kommunizieren, wäre aber als Option wünschenswert. Bei dem unter Android 2.3.4 mitgeliefertem SIP-Stack, handelt es sich um eine modifizierte Version des *J-SIP* Stacks, der aber aufgrund der Abgeschlossenheit und der Modifikationen, wie der zusätzlichen Abstraktionsschicht durch die Android API und der nicht-zugänglichen internen Funktionen nicht in Frage kommt.

Bibliothek	**Sprache**	**Lizenz**	**Anmerkung**	**Quelle**
J-SIP	Java	Public Domain	TLS	http://jsip.java.net/
PJSip	C & Java JNI Wrapper	GPL		http://www.pjsip.org/pjsip.org http://sourceforge.net/projects/pjsip-jni
Sofia-SIP	C	LGPL	NAT-T (STUN); TLS	http://sofia-sip.sourceforge.net/
MjSip	Java	GPL		http://www.mjsip.org
Android SIP (SDK)	Java	%	TLS nicht möglich	Android SDK (API Level 10)[149]

Tabelle 12.2: SIP Bibliotheken (Vergleich)

J-SIP ist in Java programmiert und ist eine Entwicklung des *National Institute of Standards and Technology* (NIST), was für die Qualität der Bibliothek spricht. Zudem ist die Dokumentation und die Menge an Beispielanwendungen sehr zahlreich. Die Entwicklung von *J-SIP* wird zur Zeit weiterhin kommerziell vorangetrieben durch mobicents[150].

Sofia-SIP ist eine Entwicklung die auf dem SIP-Stack des *Nokia Research Centers* basiert, wurde allerdings von einer Open-Source Community weiterentwickelt. *Sofia-SIP* unterstützt alle Features des SIP RFC 3261 und lässt sich durch TLS absichern.

Vergleicht man *J-SIP* mit *Sofia-SIP*, sprechen die *freiere* Lizenz (Public Domain), sowie die einfachere Verwendung unter Android (Java) für den J-SIP Stack. Auf-

Bibliothek	Sprache	Lizenz	Anmerkung	Quelle
ZRTP4J	Java	GPL 3.0 mit besonderer Ausnahme-regelung	Einfache FMJ Integration	http://www.gnutelephony.org/index.php/GNU_ZRTP4J
ZORG	Java & C++	AGPL 3.0	einfache ccrtp Integration	http://www.zrtp.org
libzrtp SDK	C	AGPL & kommerziell	offizielle Implementierung	http://zfoneproject.com/prod_sdk.html
libzrtpcpp	C++	GPL 3.0	Einfache ccrtp Integration	http://www.gnutelephony.org/index.php/GNU_ZRTP

Tabelle 12.3: ZRTP Bibliotheken (Vergleich)

grund dieser beider Fakten und der guten Dokumentation habe ich mich für die Verwendung des J-SIP Stacks entschieden.

12.1.4 Key Management Protokoll ZRTP

Bei der Auswahl der Bibliothek für die ZRTP Implementierung ist man ebenfalls stark eingeschränkt durch die Vorgabe der Softwarelizenz. Es existiert nur eine Bibliothek, die eine spätere kommerzielle Verwendung des Produktes (ohne Offenlegung des Quellcodes) ermöglicht: *ZRTP4J*. Diese ist zwar unter GPL 3.0 veröffentlicht, erlaubt aber durch eine Zusatzklausel eine Verwendung in unfreier Software wenn man die Bibliothek in unabhängigen Modulen verlinkt.
ZRTP4J ist in Java implementiert, was zur Folge hat, dass eingehende ZRTP Pakete von der Android NDK Ebene zur SDK Ebene weitergereicht werden müssen. Dies ist bei ZRTP Paketen im Gegensatz zu RTP Paketen allerdings nicht weiter problematisch, da eine geringe zusätzliche Latenz bei der Verarbeitung von ZRTP Paketen nicht weiter ins Gewicht fällt.

Aus technischer Sicht betrachtet sind *ZORG*, *libzrtpcpp* und *libzrtp* SDK ebenfalls geeignet um unter Android den Schlüsselaustausch mittels des ZRTP Protokolls zu realisieren, scheiden aber aufgrund der Lizenzierung unter AGPL, beziehungsweise GPL, aus. Eine Übersicht über ZRTP-Bibliotheken ist in Tabelle 12.3 zusammengestellt.

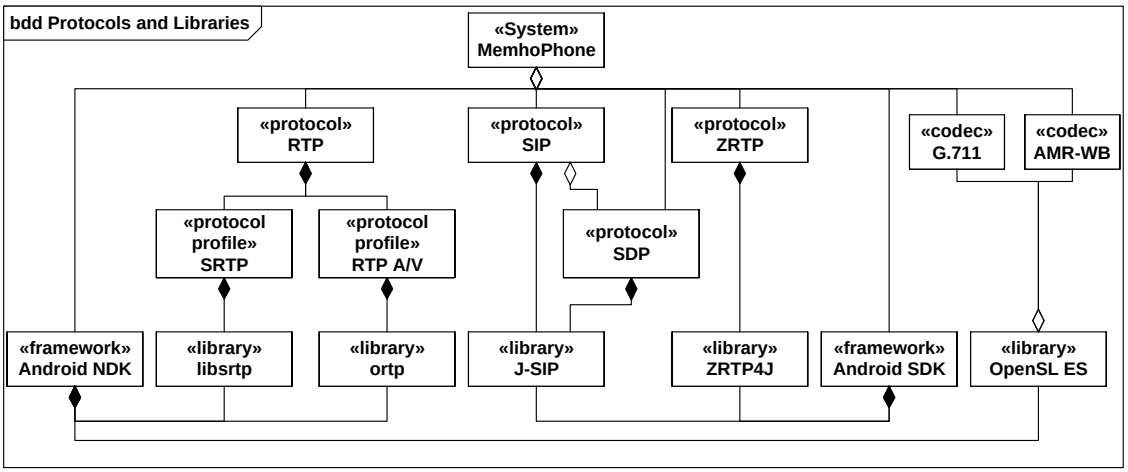

Abbildung 12.1: Protokolle und Bibliotheken

12.1.5 Übersicht über die verwendeten Bibliotheken

Die Referenzimplementierung MemhoPhone setzt sich aus zahlreichen in Android integrierten Bibliotheken und zusätzlichen externen Bibliotheken zusammen. Die Bibliotheken werden dabei für die Realisierung der Protokollfunktionalität von einem oder mehreren Protokollen benutzt. Aus welchen Protokollen sich die Implementierung MemhoPhone zusammensetzt und welche Bibliotheken für die Realisierung ausgewählt wurden, wird in Abbildung 12.1 dargestellt.

12.2 Datenflussmodell

Bedingt durch die Auswahl der Bibliotheken für das Audiotranscoding und den (S)RTP Protokollstack wird die Verarbeitung der Audiodaten auf Ebene des Android NDKs in einer Audioverarbeitungspipeline durchgeführt. Die Implementierung der Audioverarbeitungspipeline erfolgt vollständig in der Programmiersprache C.

Die Verarbeitung der Daten in der Audioverarbeitungspipeline soll dabei schrittweise erfolgen. Die Audiodaten werden zunächst über die Audiohardware in den Arbeitsspeicher geladen. Die Audiodaten liegen nach der Aufnahme in einem Audio RAW Format vor und sind unkomprimiert und würden für den Transport über das Netzwerk zu viel Bandbreite benötigen. Daher werden sie in einen Audiocodec umgewandelt der weniger Bandbreite benötigt. Danach werden die encodierten Audiodaten in RTP Pakete verpackt und anschließend verschlüsselt. Der letzte Schritt ist, die verschlüsseltem SRTP Pakete durch die Netzwerkhardware zu verschicken. Die Audioverarbeitungspipeline ist in Abbildung 12.2 schematisch dargestellt.

Der Empfang von verschlüsselten SRTP Paketen geschieht analog dazu in umgekehrter Reihenfolge. Der Datenaustausch zwischen den einzelnen Stufen der Audiopipeline erfolgt gepuffert und wird in separaten Threads erfolgen um die Verarbeitung

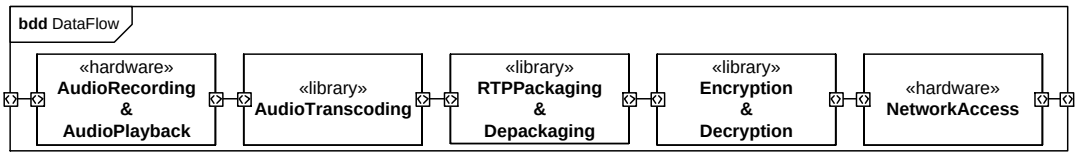

Abbildung 12.2: Datenfluss im NDK

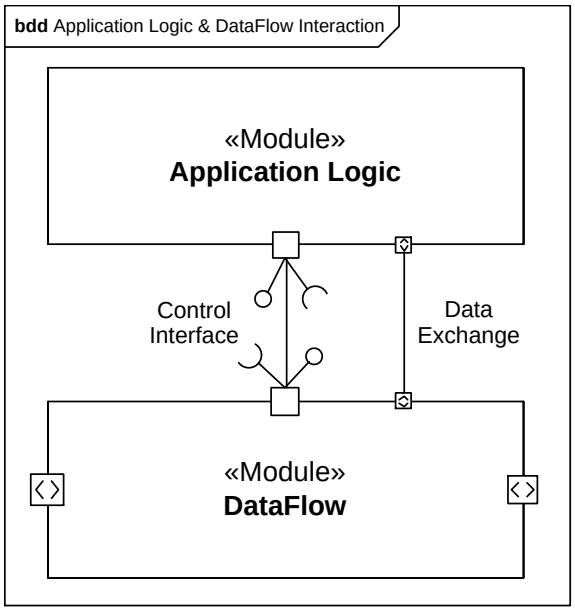

Abbildung 12.3: Anwendungslogik Java

parallelisierbar zu implementieren. Allerdings wird nicht jede Verarbeitunsgstufe in einem separaten Thread ausgeführt. Die Parallelisierung beschränkt sich auf das Audiotranscoding und die Funktionalität des RTP-Stacks.

Die Anwendungslogik und Integration in das Android OS wird in Java realisiert, da die Implementierung dieses Anwendungsmoduls in Java effizienter und besser zu realisieren ist. Der Datenaustausch zwischen Audiopipeline im Modul DataFlow und dem Modul Anwendungslogik beschränkt sich auf ein Minimum und beinhaltet keinerlei Audiodaten die die Sprachkommunikation betreffen. Der Zusammenhang zwischen Anwendungslogik und der Audioverarbeitungspipeline ist in Abbildung 12.3 dargestellt.

12.3 Anwendung als SystemService

Android Anwendung werden pausiert wenn sie nicht im Vordergrund ausgeführt werden, das heißt wenn sie nicht auf dem Bildschirm des Mobilgeräts aktiv sind. Dadurch wird der Anwendung keine Rechenzeit mehr zugewiesen und der Programmzustand muss gesichert werden. Dies hat zur Folge, dass man bei der Implementierung als gewöhnliche Android Anwendung, vor eingehenden Anrufen die Anwendung manu-

ell starten müsste und man während der Laufzeit der Anwendung nichts weiter mit dem Mobilgerät machen könnte.

Um dies zu umgehen wird der SIP-Protokollstack als Android System Service in einem separaten Thread implementiert. Ein Service unter Android ist mit einem Unix Daemon, also einem Hintergrundprozess vergleichbar, der auf eingehende SIP Anrufsignalisierung lauscht. Zusätzlich ist der SIP Service für das periodische Registrieren am SIP-Registrar zuständig ist. Das periodische Registrieren ist notwendig, da die maximale Registrierungsdauer nach einer einzelnen SIP-Register Nachricht nur 3600 Sekunden beträgt und die Registrierung bei längeren Registrierungszeiträumen daher periodisch erneuert werden muss.

12.4 Java Native Interface und NDK

Um die C Bibliotheken mit dem NDK verwenden zu können müssen Wrapperklassen oder Wrapperfunktionen geschrieben werden. Zum einen in Java, zum anderen deren Gegenstück in C. Dies ist bedingt durch die Funktionsweise des *Java Native Interfaces*. Für weiterführende Informationen zur Verwendung des Java Native Interfaces verweise ich an dieser Stelle an die offizielle Dokumentation[151].

Um den Implementierungsaufwand der Wrapperklassen gering zu halten, wird deren Anzahl auf zwei beschränkt, was einen optimalen Kompromiss aus Modularität und Implementierungsaufwand darstellt. Die beiden zu erstellenden Wrapperklassen sind die Klasse für die Verwendung der Audiobibliothek und Audiohardware, sowie eine Klasse für die Kontrolle der RTP-Funktionalität einschließlich der integrierten SRTP-Bibliothek. Dadurch wird es möglich die Audiobibliothek und die RTP-Bibliothek auch in anderen verwandten Projekten zu verwenden. Zudem bleibt dadurch die Übersichtlichkeit der Anwendungsarchitektur gewahrt.

12.5 Anwendungskonfiguration

Die Einstellung der Anwendungsparameter müssen vom Benutzer vorgenommen werden können. Dazu zählen die Konfiguration der Medientransport- und SIP Ports, der SIP Serveradresse und dem SIP Benutzernamen, sowie weitere relevante Anwendungsparameter. Die benutzerdefinierten Einstellungen müssen für den Entwickler in allen Bereichen der Anwendung zur Verfügung stehen. Die Architektur der Einstellungsverwaltung muss entsprechend gewählt werden.

Die Konfiguration der benutzerdefinierten Anwendungsparameter sollte nur zentral an einer Stelle erfolgen, damit nicht zu jeder Zeit und von jedem Anwendungsmodul Parameter geändert werden können.

Alle Anwendungskomponenten die die benutzerkonfigurierten Einstellungen verwenden, müssen nach deren Änderung darüber informiert werden, damit die entsprechende Anwendungskomponenten die neue Konfiguration laden können.

12.6 Graphische Benutzeroberfläche und Systemintegration

Durch die Offenheit des Systems (Kapitel 2.4), hat man unter Android die Möglichkeit seine eigene Anwendung nahtlos in das System zu integrieren. Die Anwendung wird daher nicht als klassische Android Anwendung implementiert bei der man eine Vollbildanwendung bedient, sondern in die originale Telefonieanwendung integriert. Dies wurde in Kapitel 12.3 bereits thematisiert und wird in diesem Kapitel weiter ausgeführt.

Erst im Falle eines eingehenden Anrufes, muss wie bei einem gewöhnlichen Anruf aus dem Mobilfunknetz der Benutzer umgehen akustisch und visuell durch einen Anrufbildschirm über den eingehenden Anruf informiert werden. Die akustische Benachrichtigung soll sich dabei nach den Audioprofileinstellungen des Mobiltelefons richten. Damit der Benutzer visuell über einen eingehenden Anruf informiert wird, soll eine Activity, also eine Grafische Bedienoberfläche in Android, im Vollbild gestartet werden

Auf der Activity des Anrufbildschirm muss der ZRTP Short Authentication String angezeigt werden damit sich die Anrufer gegenseitig authentifizieren können. Zusätzlich muss auf dem Anrufbildschirm auch dargestellt werden wie der Status des Verschlüsselungsmechanismus ist, da man sowohl verschlüsselt als auch unverschlüsselt über die Anwendung telefonieren kann. Zusätzlich wird die SIP-URI oder der Name des Anrufers angezeigt sofern er im Telefonbuch eingetragen ist.

13 Implementierung

Die Umsetzung des Konzeptes aus dem vorangehenden Kapitel (Kapitel 12) wird hier erläutert. Es werden zunächst die Entwicklungsumgebung und die verwendete Hardware vorgestellt, bevor in Kapitel 13.3 der Aufbau der Anwendung MemhoPhone dargelegt wird. Die darauf folgenden Kapitel präsentieren einzelne Module der Referenzimplementierung, bevor in Kapitel 13.9 die Gesamtarchitektur dargestellt wird. Besondere Aspekte bei der Entwicklung des Prototypen werden in Kapitel 13.10 separat erläutert, bevor auf die Verwendung eines Software Private Branch Exchanges in Kapitel 13.11 eingegangen wird.

Die Referenzimplementierung (*MemhoPhone*) wird für das Android Release 2.3.4 (Android SDK API Level 10) entwickelt. Die Diagramme wurden mit Hilfe der Modellierungssprache SysML erstellt, die in Anhang B vorgestellt wird.

13.1 Entwicklungsumgebung

Bei der Implementierung des Prototypen für die verschlüsselte Sprachkommunikation wurde eine Vielzahl an Frameworks und Entwicklungswerkzeugen eingesetzt. Entwickelt wurde unter Debian 7.0 Codename Wheezy[152], das aktuell noch die Testing Distribution der Debian Distribution darstellt. Als Entwicklungsumgebung kommt Eclipse in Version 3.7 zum Einsatz, inklusive des ADT-Plugins[153] in Version 12.0, welches die Android SDK Tools[154] in Eclipse integriert. Die Android SDK Tools, darunter Debugger, Tracer und einem Management Programm für den Android Emulator kommen ebenfalls in Version 12.0 zum Einsatz. Das Android NDK, mit dem die externen Bibliotheken, die in C geschrieben sind verwendet werden, wird in Revision 6 eingesetzt[155].

Die Referenzanwendung wird für Android 2.2 (API Level 8) kompiliert, inklusive aller Module und externen Bibliotheken und auf Android 2.3.4 auf dem Nexus S getestet und eingesetzt. Einzelne Komponenten der Anwendung wurde auch im Emulator (Android Virtual Device) mit Android 2.2 und Android 2.3.3 getestet.

Der gesamte Quellcode, einschließlich der externen Bibliotheken wird mittels des git Versionskontrollsystems[156] verwaltet und ermöglicht es, durch die dezentrale Versionierungsarchitektur, das Quellcode Repository auf mehreren Entwicklungs-

Abbildung 13.1: Google Nexus S (Quelle: [157])

computern und externen Laufwerken zu synchronisieren, um die Redundanz und Mobilität der Softwareentwicklung zu ermöglichen, ohne dabei auf eine Internetverbindung angewiesen zu sein.

13.2 Google Nexus S

Die Referenzimplementierung und Evaluation wurde auf Basis des Smartphones Nexus S (*Codename Crespo*), das durch eine Gemeinschaftsproduktion von Google und Samsung entwickelt wurde, durchgeführt. Das aktuelle Google-Handy Nexus S (dargestellt in 13.1) wird vom Hersteller Samsung auch als *Samsung GT-I9020T* geführt und vertrieben. Das Google Nexus S genügt bezüglich der Konnektivität den unter Kapitel 4.1 spezifizierten Anforderungen völlig.

Konnektivität

- Quad-band GSM: 850, 900, 1800, 1900
- Tri-band HSPA: 900, 2100, 1700
- HSPA type: HSDPA (7.2Mbps) HSUPA (5.76Mbps)
- WiFi 802.11 n/b/g
- Bluetooth 2.1+EDR
- Near Field Communication (NFC)

Prozessor

- 1GHz Cortex A8 (Hummingbird) Prozessor
- Samsung 16GB iNAND Flashspeicher

Betriebssystem

- Android 2.3.4 (*Codename Gingerbread*)

13.3 Allgemeiner Anwendungsaufbau

Die Implementierung und die Verwendung der einzelnen Protokolle, die in der Anwendung zur sicheren Sprachkommunikation Verwendung finden, sind in protokollspezifische Klassen gekapselt. Diese Klassen tragen den Suffix "*Manager*" und weisen sich dadurch als alleinige Kontrollinstanz für die Verwendung eines spezifischen Protokolls aus.
Neben den Managerklassen für die Implementierung der Protokollkomponenten, existieren noch weitere Managerklassen für die Verwaltung der benutzerdefinierten Einstellungen (*SettingsManager*), sowie für die Verwaltung von Anrufdaten (*CallManager*).

Für die Interaktion zwischen Java (SDK) und C-Ebene (NDK) der Anwendung wurden zwei JNI-Wrapperklassen erstellt: ORTPWrapper für die JNI-Calls zur RTP-Bibliothek *ortp* und *libsrtp* und OpenSLWrapper für die JNI-Calls zur Audio Transcoding Bibliothek *OpenSL ES.*

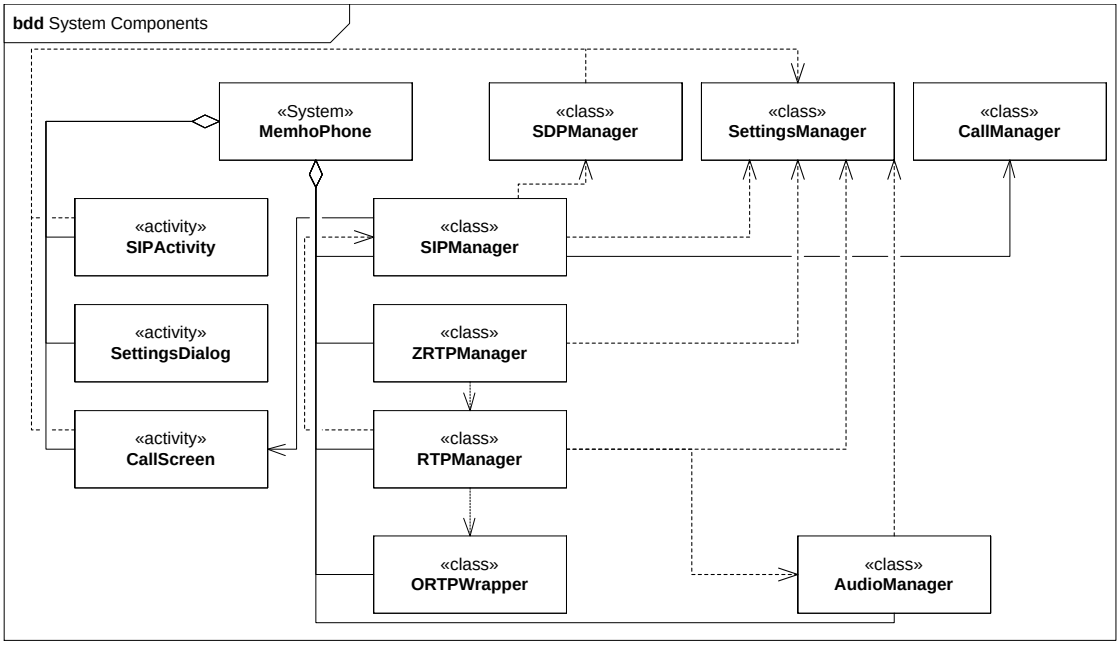

Abbildung 13.2: Systemkomponenten und Abhängigkeiten

In Abbildung 13.2 ist dargestellt aus welchen Hauptkomponenten sich die Anwendung zur sicheren Sprachkommunikation zusammensetzt. Ebenso sind die Abhängigkeiten der Komponenten untereinander dargestellt.

Besonders hervorzuheben ist die Eigenschaft des SettingsManagers, der die benutzerdefinierten Einstellungen der Anwendung zentral zugänglich macht. Um einen einfachen Zugriff auf die Klasse zu ermöglichen und um die multiple Instantiierung zu verhindern ist sie als Singleton implementiert. Der CallManager ist ebenfalls als

Singleton implementiert aus den gleichen zuvor genannten Gründen.
Eine Besonderheit des SettingsManagers ist, dass er ausschließlich von der Klasse SIPActivity aus beschrieben wird. Die SIPActivity ist die Hauptkomponente der Anwendung MemhoPhone und starte die übrigen Managerklassen. Alle übrigen Klassen greifen nur lesend auf ihn zu. Durch diese Eigenschaft wird verhindert, dass benutzerdefinierte Konfiguration an unterschiedlichen Stellen oder unbemerkt geändert werden. Nach der Änderung der benutzerdefinierten Einstellungen, werden ausgehend von der SIPActivity alle Managerklassen benachrichtigt, damit die Bibliotheken und Variablen, die in Abhängigkeit von den benutzerdefinierten Einstellungen stehen, neu initialisiert werden.

Die Benutzeridentifikationssignatur (ZID) des ZRTP-Protokollstacks wird in einem abgeschotteten Speicherbereich des Android OS gespeichert, zu dem nur die Anwendung MemhoPhone Zugang hat.

Zur Interaktion zwischen den Klassen werden Callbackinterfaces benutzt, die nur die wirklich benötigten Funktionen enthalten mit denen die Klassen interagieren. Dadurch sind das Anwendungs- und Interfacedesign voneinander unabhängig, was die Modularität der Anwendung und die Wartbarkeit erhöht.

Um den Quellcode und die Klassen strukturiert zu organisieren, sind sie in Pakete unterteilt, deren Wurzelpaket, unter dem alle restlichen Pakete und Klassen organisiert sind, das Paket `de.consistec` ist. In Abbildung 13.3 ist die Organisation der Klassen und Pakete grafisch dargestellt, beschränkt sich bei der Darstellung aber auf die wichtigsten verwendeten Pakete und Klassen um die Übersichtlichkeit zu wahren.

Besondere Erwähnung verdienen an dieser Stelle die Klassen im Paket `de.consistec.manager`, die die wichtigsten Klassen zur Protokoll- und Ablaufsteuerung darstellen. Das Paket `de.consistec.wrapper,` das alle JNI Wrapperklassen beinhaltet, sowie das Paket `de.consistec.gui` in dem alle Klassen zur Realisierung der grafischen Benutzeroberfläche enthalten sind, sind ebenfalls von zentraler Bedeutung.

13.4 Audiopipeline

Eine der Hauptaufgabe der Anwendung zur sicheren Sprachkommunikation, ist die effiziente Verarbeitung von Audiodaten. Das in Kapitel 12.2 vorgestellt Konzept konnte wie dargestellt in der Implementierung umgesetzt werden. Die Verarbeitung der Audiodaten erfolgt nach dem bekannten Entwurfsmuster einer Pipeline und wird daher auch als Audiopipeline bezeichnet.

Der Hardwarezugriff für die Akquirierung und Wiedergabe der Audiodaten, das Audiotranscoding, das Packen und Entpacken der Audiodaten in RTP-Pakete, das

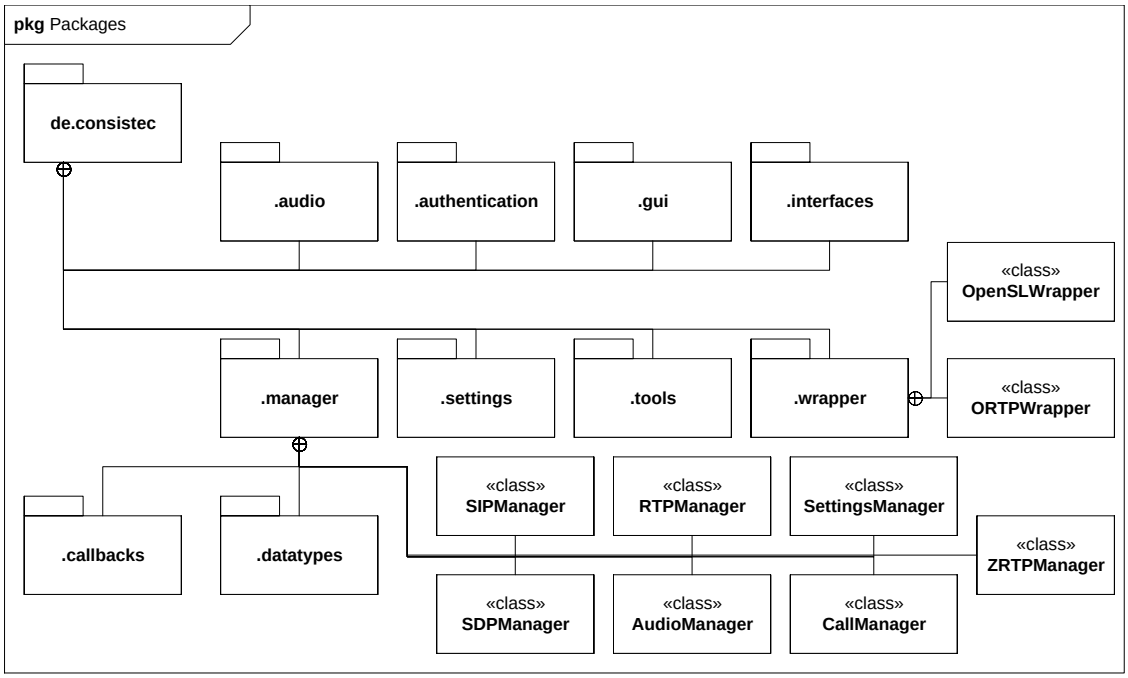

Abbildung 13.3: MemhoPhone Paketstruktur

Ver- und Entschlüsseln der RTP-Pakete und der Zugriff auf die Netzwerkhardware erfolgt vollständig in der Programmiersprache C und wird mit Hilfe des Android NDK realisiert. Dadurch wird die Problematik der latenzinduzierenden JNI Funktionsaufrufe vermieden. Die vollständige Audioverarbeitungskette ist in Abbildung 13.4 dargestellt und wird in diesem Kapitel explizit erläutert.

Die Steuerung und Kontrolle über die Module Audiostack (mit der OpenSL ES Bibliothek) und dem (S)RTP-Stack (mit den ortp und libsrtp Bibliotheken) erfolgt über zwei JNI Wrapperklassen, mit deren Hilfe man aus der Java Ebene Funktionen der Audiostacks und des (S)RTP-Stacks aufruft. Die Kontrollinterfaces sind in Abbildung 13.4 durch die entsprechenden Interface Symbole dargestellt. Die Aufbau der Wrapperklassen ist an dieser Stelle nicht entscheidend und wird im folgende Kapitel 13.5 separat beschrieben.

Durch die Verwendung einer ZRTP-Protokollimplementierung in der Programmiersprache Java (*ZRTP4J*), wird es notwendig ZRTP-Pakete für die Abwicklung des ZRTP-Protokolls in die Android SDK Ebene durchzureichen. Um das zu ermöglichen, müssen ZRTP-Pakete von der RTP-Implementierung erkannt werden, um sie von den RTP Paketen zu unterscheiden. Dies ist notwendig da ZRTP-Pakete im RTP Protokoll inband, über den gleichen Unix-Socket übertragen werden.

Eigentlich bietet die ZRTP4J Protokollimplementierung die Funktionalität der ZRTP-Paketerkennung. Durch Übergabe von RTP Paketen an die folgende Funktion (Listing 13.1) prüft der ZRTP Stack ob es sich um ein valides ZRTP Paket handelt.

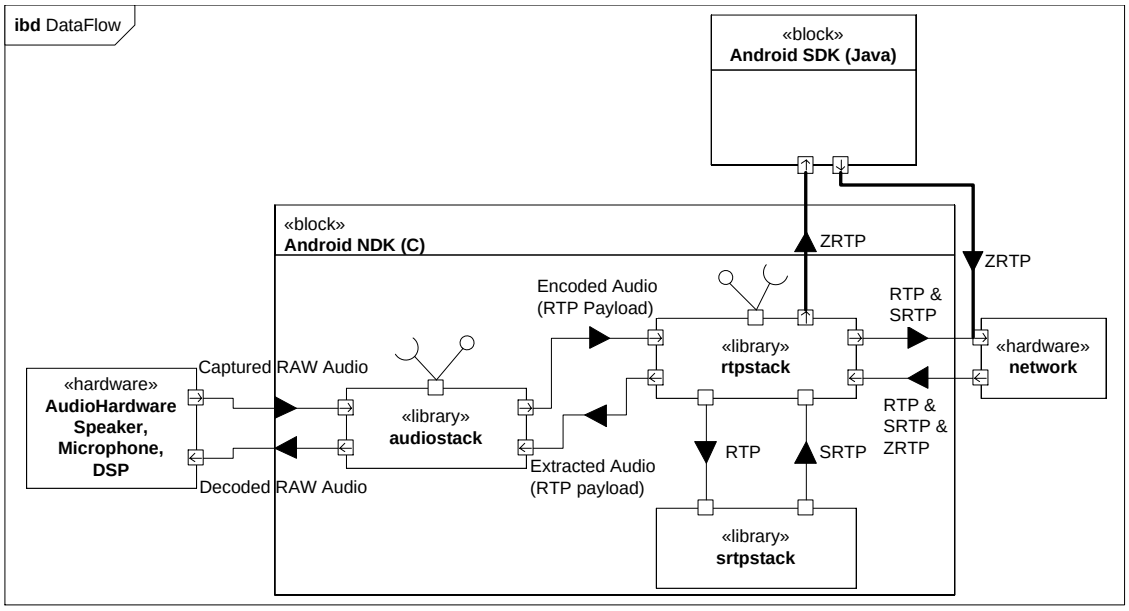

Abbildung 13.4: Datenfluss (Implementierung)

Listing 13.1: ZRTP4J Funktion zur Prüfung von RTP Paketen

```
processZrtpMessage(byte[] extHeader, int ssrc)
```

Falls dies zutrifft, wird das Paket entsprechend dem ZRTP Protokollablauf weiterverarbeitet und >>0<< zurückgegeben. Sofern es sich um ein gewöhnliches RTP-Paket handelt gibt die Funktion >>1<< zurück und man kann das RTP wie gewohnt weiterverarbeiten. Die ZRTP-Erkennung auf Basis der ZRTP4J Bibliothek würde allerdings bedeuten, dass die zusätzliche Latenz durch den JNI Funktionsaufruf auch auf RTP Pakete Auswirkung hat und nicht nur – wie geplant – auf ZRTP Pakete.

Deshalb wurde die ZRTP-Paketerkennung nachträglich im RTP-Stack integriert, so dass gezielt einzelne ZRTP-Pakete an den ZRTP4J-Stack weitergereicht werden können ohne jedes RTP-Paket übergeben zu müssen. Dadurch bleibt die Verarbeitung der Audiodaten auf Ebene des Android NDK davon unberührt und es wird keine zusätzliche Latenz verursacht.

13.5 NDK Wrapperklassen

Um C Funktionen von Java aus aufzurufen und um vice versa auch Java Funktionen aus dem C Quellcode aufzurufen wurden zwei Wrapperklassen implementiert, die die Funktionsaufrufe der einen Programmiersprache in die Funktionsaufrufe der jeweils anderen Sprache übersetzen.

Die vollständige Audiopipeline ist in C implementiert und wird durch den RTPManager und den AudioManager gesteuert (Erläutert in Kapitel 13.4).

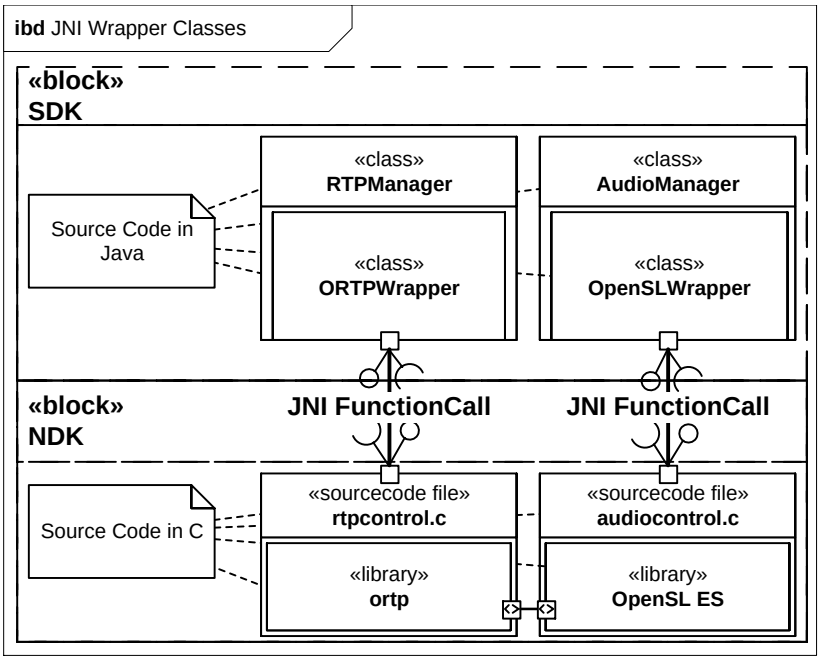

Abbildung 13.5: Java Native Interface Wrapperklassen

Der RTPManager, sowie der AudioManager verwenden ihre jeweilige bibliotheksspezifische Wrapperklasse ORTPWrapper und OpenSLWrapper um die Funktionen der C-Bibliotheken aufzurufen. Die Audiopipeline wird dadurch auf abstrakter Ebene von der SDK-Ebene aus mit einigen wenigen API Funktionsaufrufen gesteuert.

Die Audioverarbeitungspipeline ist in den Blöcken rtpcontrol.c und audiocontrol.c implementiert. Zusätzlich ist in den beiden Blöcken die Implementierungen der nativen Funktionen der beiden Wrapperklassen realisiert. Der Aufbau der NDK Wrapperklassen ist in Abbildung 13.5 dargestellt.

Es existieren Funktionen zur schrittweisen Konfiguration und Initialisierung der Bibliotheken, zum Starten der RTP- und Audioübertragung, sowie zur Deinitialisierung der Bibliotheken. Der ORTPWrapper hat zusätzliche Funktionen um ZRTP Pakete zwischen C- und Javaebene auszutauschen, sowie um die SRTP Verschlüsselung zu aktivieren, nachdem mittels ZRTP die Schlüssel vereinbart wurden.

13.6 Threadingarchitektur

Die Anwendung zur sicheren Sprachkommunikation unter Android macht es aufgrund der multiplen und simultan operierenden Netzwerkprotokollstacks erforderlich, dass die einzelnen Bibliotheken in einem eigenen oder sogar mehreren Threads operieren, damit eine parallele Abarbeitung der Protokolle möglich ist. Da es sich beim ARM Cortex A8 um einen Singlecore Prozessor handelt ist die Abarbeitung allerdings nur quasi-parallel nach dem *Completely Fair Scheduler* (CFS) Algorith-

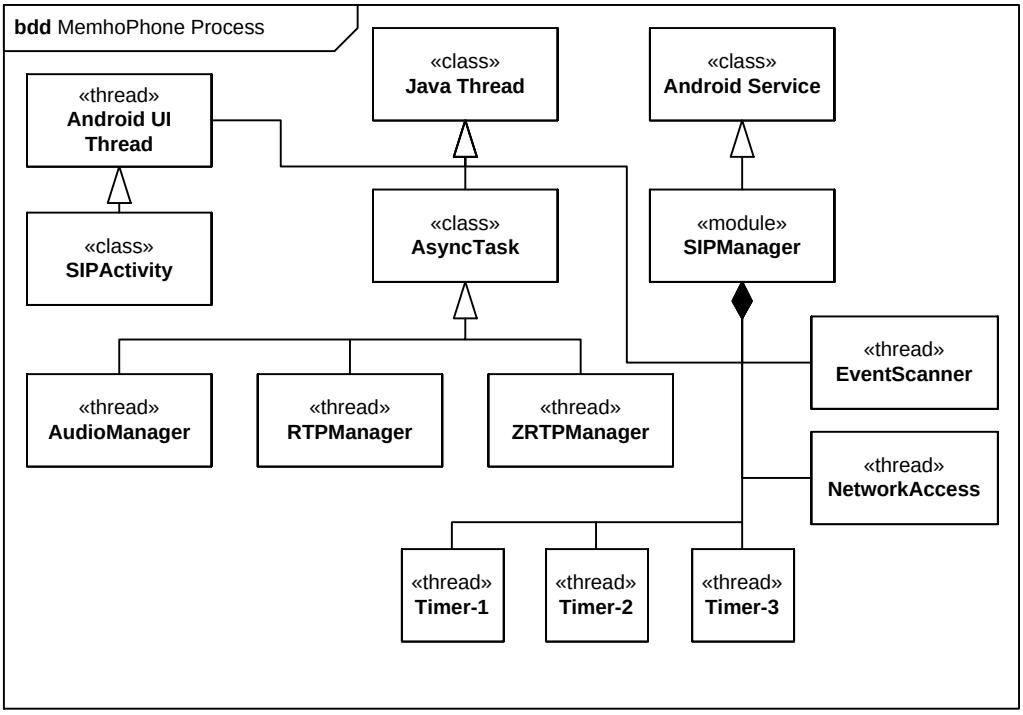

Abbildung 13.6: Threads

mus[158] des Linux Kernels. Weitere Information zum CFS-Algorithmus können unter [159] nachgelesen werden.

Die Benutzeroberfläche der Anwendung läuft bedingt durch die Android Systemarchitektur ebenfalls in einem separaten Thread, um die Benutzeroberfläche nie durch länger andauernde Funktionsaufrufe zu blockieren und so das Benutzererlebnis zu beeinträchtigen. Damit der Thread in dem die GUI Elemente dargestellt werden von anderen Threads und Klassen manipuliert werden kann, wird der *Anwendungskontext* bei der Initialisierung der Klassen die auf die GUI zugreifen müssen mitgegeben.

Die Anwendungsmodule AudioManager, RTPManager und ZRTPManager sind als Android *AsyncTask* realisiert. Die Android Javaklasse *AsyncTask* ist eine vereinfachte Möglichkeit um Java Threads unter Android zu realisieren und um die Grafische Benutzeroberfläche von einem separaten Thread aus zu manipulieren. Die Abhängigkeiten der Threadklassen untereinander sind in Abbildung 13.6 veranschaulicht.

Der native C Quellcode der RTP, SRTP und Audiobibliothek wird innerhalb der Java Threads des AudioManagers und des RTPManagers in der Dalvik VM des Android Betriebssystems ausgeführt.

Der Datenaustausch zwischen den Threads, also auch zwischen den einzelnen Modulen der Audioverarbeitungspipeline geschieht gepuffert (siehe Abbildung 13.7), was bedeutet, dass nach der Übergabe einzelner Pakete zwischen den Threads nicht auf deren Abarbeitung gewartet werden muss. Es wird nur die reine Zeit benötigt um einen Pointer oder Referenz in die eingehende Verarbeitungsqueue eines Moduls zu

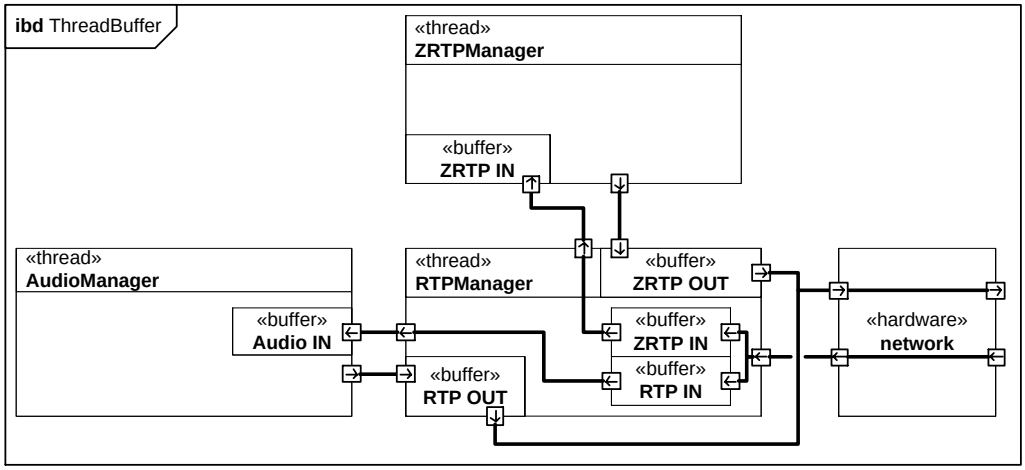

Abbildung 13.7: I/O Buffer Audioverarbeitung bei Threads

schreiben, der dann vom dem entsprechenden Modul weiterverarbeitet wird, sobald dem Thread vom Betriebssystem wieder Rechenzeit zugewiesen wird.

Das Lesen und Schreiben in die Threadbuffer ist durch Lockmechanismen gesichert um atomar Elemente in den Threadbuffer zu schreiben und lesen zu können. Für die Javabuffer wird dies durch eine `ConcurrentLinkedQueue` aus dem *Java Concurrency Framework realisiert.*
Für die Threadbuffer, die in C implementiert sind, wird die Queue durch einen zusätzlich ergänzten Lockmechanismus für das Schreiben und Lesen in die Queueimplementierung `SIMPLEQ` der Linux `Bibliothek <sys/queue.h>` realisiert.

13.7 Android Integration und GUI

Über ein Symbol im Notification Bereich des Android OS wird dem Benutzer signalisiert, ob der SIP Service aktiv ist und ob auf eingehende Anrufe gelauscht wird. Der Notification Bereich befindet sich am oberen Rand des Android Bildschirms und ist mit dem Konzept der *Status Area* unter Windows ab Version '95 aufwärts vergleichbar (Abbildung 13.8). Aktiviert man die Vollbildansicht des Notification Bereiches, wird die Detailansicht aktiviert. In der Detailansicht werden Statusmeldungen der Anwendung dargestellt und durch berühren der Statusmeldung, wird die Telefonieanwendung mit aktiviertem Modus zur sicheren Sprachkommunikation gestartet (Abbildung 13.8 rechts).

Im Feld, das die gewählte Nummer anzeigt, wird dargestellt ob man gerade über das Mobilfunknetz wählt oder ob man über die Anwendung zur verschlüsselten Sprachkommunikation telefoniert. Durch einen einfachen Knopfdruck kann der Benutzer umschalten über welche Kommunikationsanwendung gewählt werden soll (Abbildung 13.9).

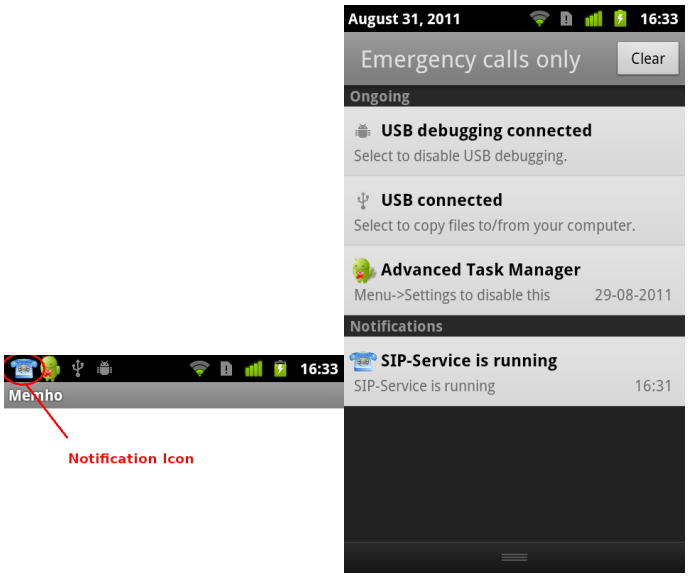

Abbildung 13.8: SIP Service und Notification

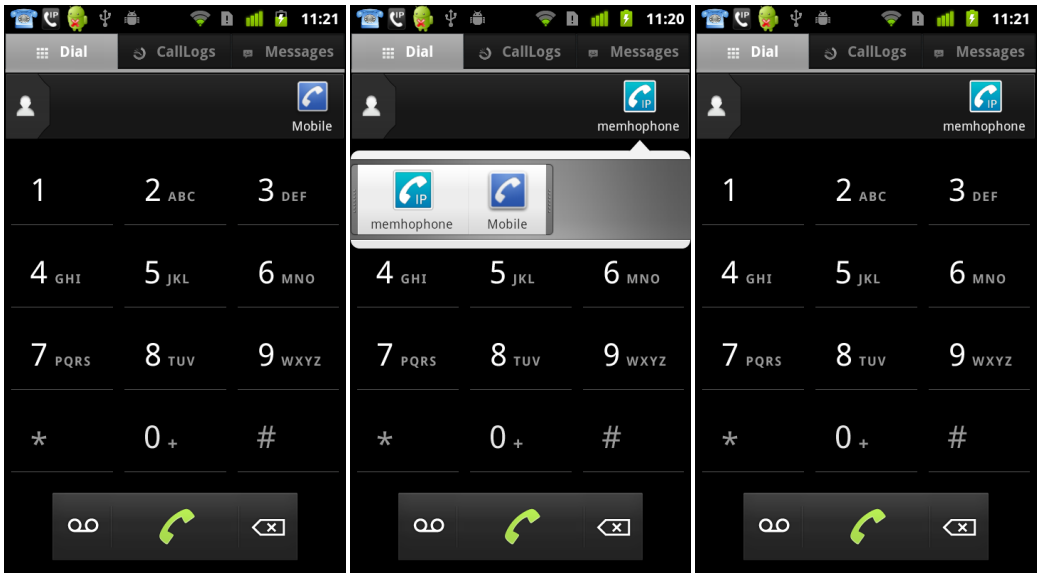

Abbildung 13.9: Integration in Android Telefonieanwendung

Ist der Modus zur sicheren Sprachkommunikation aktiv, kann man über das Kontextmenü das unter Abbildung 13.10 dargestellte Einstellungsmenü starten, in dem der Benutzer wichtige Anwendungsparameter konfigurieren kann, wie beispielsweise die SIP-Serverdaten, Ports für RTP und SIP-Verbindungen oder das Zeitintervall in dem sich periodisch am SIP Server neu registriert werden soll.

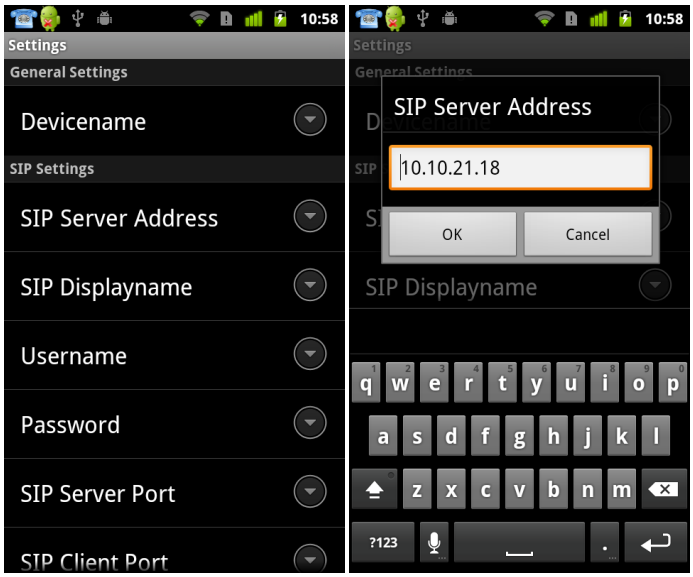

Abbildung 13.10: Einstellungsdialog und Detailansicht

Startet man einen ausgehenden Anruf oder geht ein Anruf ein, wird der Anrufbildschirm in den Vordergrund gebracht. Auf dem Anrufbildschirm in Abbildung 13.11 wird ein eingehender Anruf dargestellt. Auf ihm ist der Short Authentication String zu sehen. Durch das Schloss wird dargestellt ob der Anruf verschlüsselt ist und es ist der Name und das Foto des eingehenden Anrufes sichtbar. Über zwei Buttons kann der Anruf angenommen oder angelehnt werden.

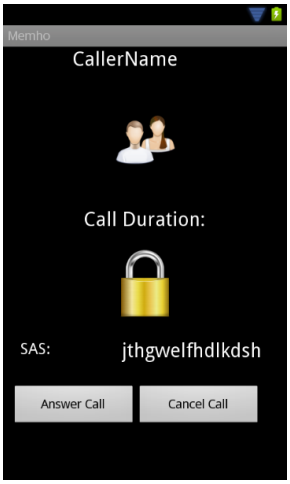

Abbildung 13.11: Anrufbildschirm

Damit der Thread in dem das Benutzerinterface abläuft manipuliert werden kann und Benutzerbenachrichtungen wie beispielsweise Alerts angezeigt werden können, bietet eine Android Activity eine spezielle Funktion, von der aus andere Threads und Klassen diese Meldungen und Manipulationen an der GUI durchführen können. Diese Funktion ist in Listing 13.2 dargestellt der Android Activity aufgerufen. Dies

Listing 13.2: Manipulieren der GUI von nicht UI-Threads

```
runOnUiThread( new Runnable() {
    public void run() {
        // Manipulate GUI here
    }
} );
```

wird durch die Implementierung des IGuiCallbacks Interfaces von der SIPActivity Klasse realisiert.

13.8 Programmablaufplan

In diesem Kapitel wird der Programmablauf in seiner Gesamtheit dargestellt. Die Erläuterungen wurden in Abbildung 13.12 visuell aufbereitet und zusammenfassend dargestellt. Die beiden Bereiche der RTP und ZRTP Übertragung und der SRTP Übertragung sind in Abbildung 13.13 separat und detaillierter abgebildet.

Der Anwendungsstart erfolgt koordiniert nach einer festen Abfolge. Zunächst startet die SIPActivity von der ausgehend, die grafische Benutzeroberfläche initialisiert wird und die gespeicherten Anwendungseinstellungen geladen werden mit denen der SettingsManager initialisiert wird. Nach der Initialisierung der Activity wird der SIPManager gestartet, der den Android Systemservice implementiert. Nach dem Start des Service verbindet sich die SIPActivity mit dem SIP-Service und initialisiert davon ausgehende die übrigen Anwendungskomponenten.

Nachdem die SIPActivity mit dem SIP-Service verbunden ist, wird nacheinander der SIP-Stack, der RTP-Stack und der Audio-Stack soweit initialisiert, das eingehende und ausgehende Verbindungen möglich sind. Nachdem die VoIP Protokollstacks fertig initialisiert sind, meldet sich die Anwendung, basierend auf den benutzerdefinierten VoIP-Einstellungen am SIP-Registrar an und startet den Thread der periodisch die Registrierung am SIP-Registrar erneuert. Ab diesem Zeitpunkt lauscht der SIP-Service auf Anrufsignalisierungen durch andere Peers oder auf den Verbindungswunsch durch den Benutzer des UE.

Bei einem ausgehenden oder ankommenden Anruf werden die RTP-Session und die Audio-Session für die Übertragung der Sprachdaten, basierend auf dem Verbindungsparametern der durch das SIP Protokoll übertragenen SDP Nachrichten, aufgebaut. Der ZRTPManager wird erst initialisiert und erst aktiviert wenn eine Sprachverbindung (RTP-Session) aufgebaut ist. Dies liegt darin begründet, dass ZRTP-Pakete über den Port der RTP-Session versendet werden und der Port erst nach der abgeschlossenen Initialisierung der RTP-Session geöffnet wird. Sind Audio-Session, RTP-Session und ZRTPManager vollständig initialisiert, beginnt die Über-

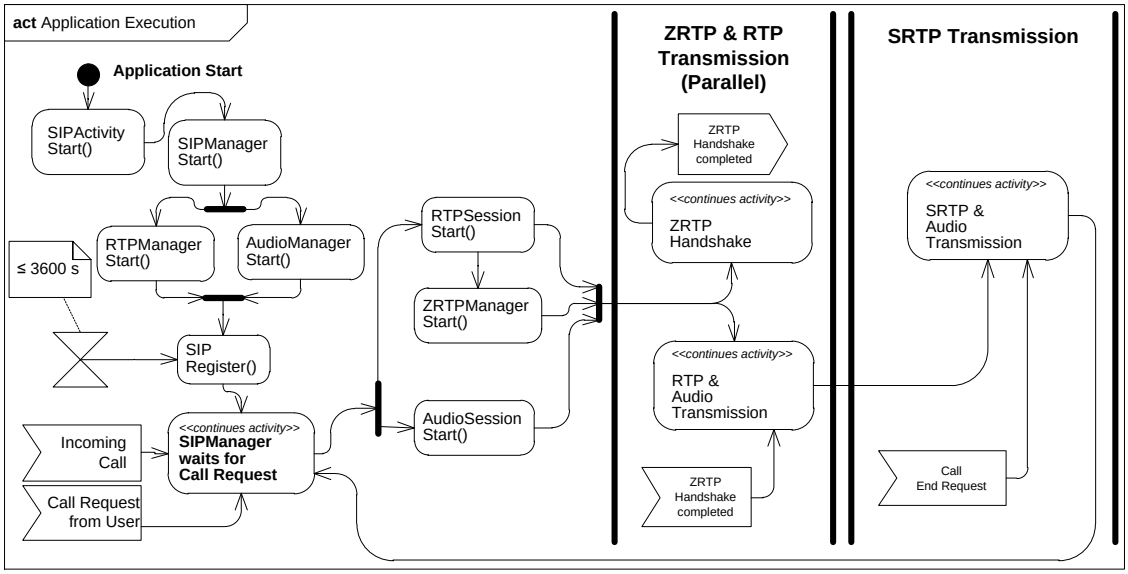

Abbildung 13.12: Programmablaufplan

tragung und der Empfang von Audiodaten in RTP-Paketen ebenso wie die Übertragung und der Empfang von ZRTP-Paketen (Abbildung 13.13).

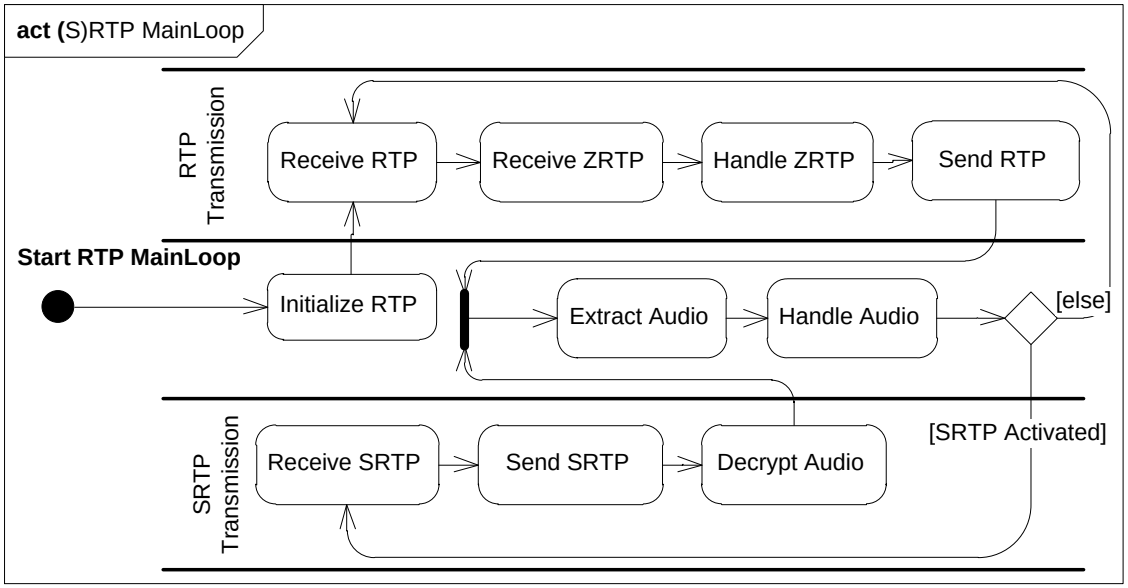

Abbildung 13.13: (S)RTP Mainloop

Zu Beginn werden die Audiodaten noch unverschlüsselt übertragen. Sobald der Schlüssel für die SRTP Verschlüsselung mittels des ZRTP-Handshakes ausgetauscht wurde (siehe Kapitel 10.3), wird die Verschlüsselung aktiviert. Ab diesem Zeitpunkt wird der ZRTPManager deaktiviert und es findet kein ZRTP-Handshake oder die ZRTP Signalisierung mehr statt. Audiodaten werden mittels SRTP nur noch verschlüsselt übertragen.

Wird der Anruf durch den Benutzer des UE oder den Kommunikationspartner beendet, werden Audio- und RTP-Session wieder beendet und die Anwendung kehrt

in den Zustand zurück in dem der SIPManager (*Android System Service*) auf eingehende und ausgehende Verbindungen wartet.

13.9 Gesamtarchitektur

Der Aufbau der Anwendung MemhoPhone wurde bisher nur in einzelnen Teilen dargestellt um spezifische Implementierungsdetails herauszuarbeiten.

In diesem Kapitel und insbesondere in Abbildung 13.14 wird dargestellt wie die einzelnen Module der Anwendung zusammenarbeiten. Dies geschieht in Hinblick auf Kontrollinterfaces, ebenso wie in Hinblick auf den Datenaustausch zwischen den einzelnen Module. Die Kontrollinterfaces dienen der Anwendungssteuerung und sorgen für den koordinierten Ablauf der Anwendungslogik wie in den Kapiteln 13.8 und 13.8 dargestellt.

Die Ports für den Datenaustausch spezifizieren, welche Daten zwischen den Modulen ausgetauscht werden können. Die einzelnen kleineren Blöcken sind durch die größeren Blöcke *Android SDK (Java)* und *Android NDK (C)* gegliedert, um zu visualisieren in welcher Programmiersprache sie realisiert wurden, da dies ja wie bereits in den Kapiteln 12.2 und 13.4 von entscheidender Bedeutung bei der Verarbeitung der Sprachdaten ist.

Ankommende ZRTP-Pakete werden gefiltert und mit Hilfe des ORTPWrappers (JNI) an den ZRTPManager weitergereicht. Der ZRTPManager verarbeitet die ZRTP-Pakete anhand des ZRTP-Protokollablaufs. Ausgehende ZRTP-Pakete werden direkt mit Hilfe des ORTPWrapper über den selben Port wie ausgehende RTP Pakete versendet, ohne von der RTP-Bibliothek verarbeitet zu werden.

Die Anrufeinstellung die vom Benutzer in der SIPActivity festgelegt wurden, werden dem Android Service SIPManager übergeben. Dieser generiert aus den Anrufeinstellungen des Benutzers eine SDP-Nachricht die mit Hilfe des SIP-Stacks zum SIP-Registrar versandt wird.

13.10 Erfahrungswerte der praktischen Umsetzung

Während der Implementierung der Referenzanwendung wurden zahlreiche Probleme festgestellt und gelöst, von denen die erwähnenswerten in diesem Kapitel explizit vorgestellt werden. Diese sind zum einen durch die Android Architektur begründet, zum anderen aber auch durch die Architektur von Java oder sind beeinflusst von Drittanwendungen wie der VoIP Serversoftware.

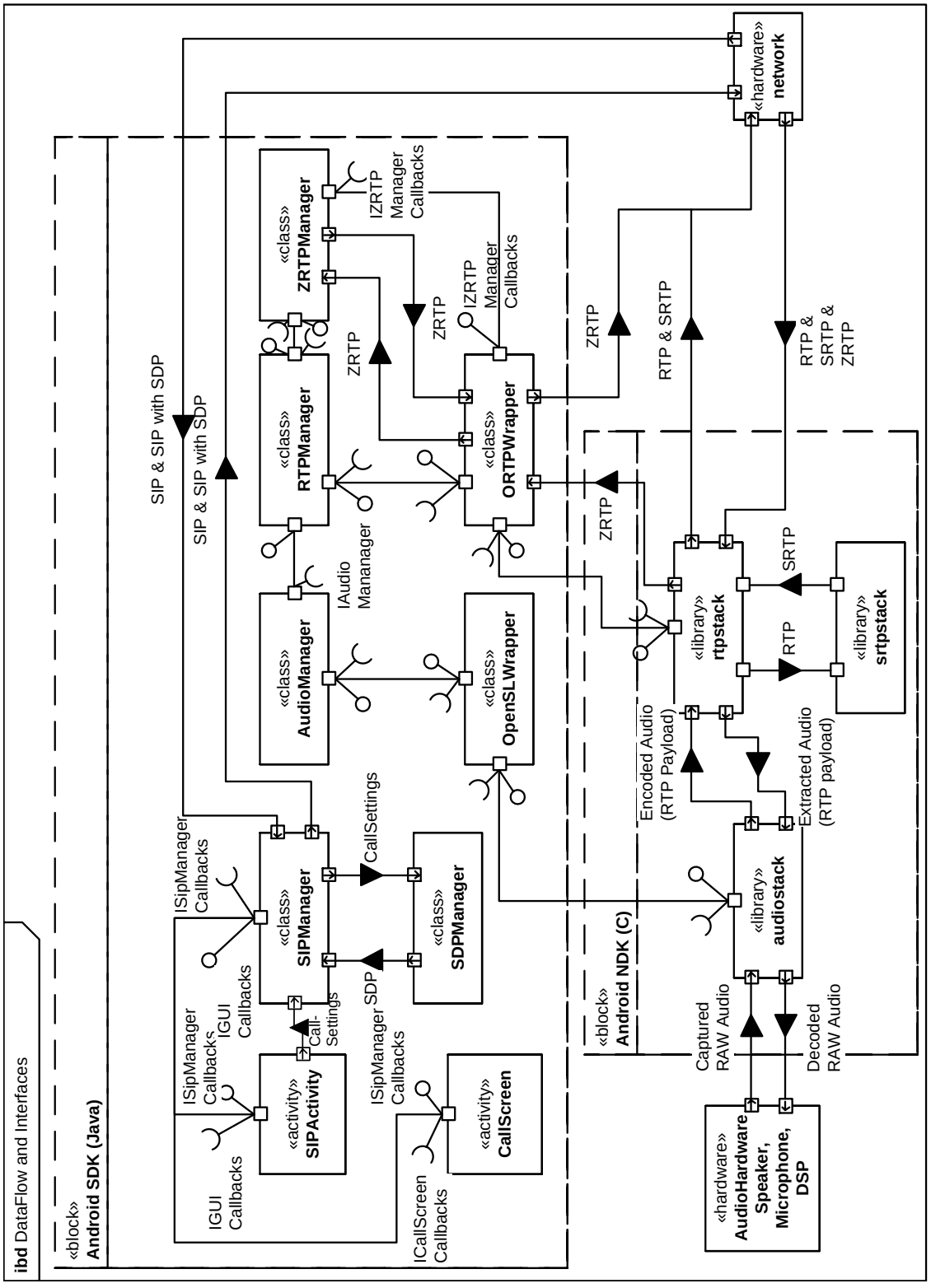

Abbildung 13.14: Datenfluss und Schnittstellen (Gesamtarchitektur)

13.10.1 Latenz

Ein Schwerpunkt bei der Implementierung lag darauf, die Latenz in der Audioverarbeitungspipeline zu minimieren um den Anforderungen an Digitale Sprachkommunikation gerecht zu werden. Diese Anforderung wurde im Konzept bei der Auswahl der Bibliotheken (Kapitel 12.1), sowie im Datenflussmodell in Kapitel 12.2 bereits beachtet. Bei der Implementierung und Laufzeitminimierung half die Analyse der Anwendungs- und Funktionslaufzeiten mit dem Android Tracingtool das im Android SDK mitgeliefert wird.

13.10.2 SIP

Bei der Verwendung des SIP Stacks kam es während der Entwicklungszeit zu einigen zeitaufwendigen Problemen, die an dieser Stelle nicht unerwähnt bleiben sollten, da sie zudem auch Einfluss auf konkrete Designentscheidungen hatten.

Packagename Collision Mit dem Android Release 2.3 wurde das Feature der VoIP Telefonie in die Android Plattform integriert[160]. Bei der Implementierung der SIP Funktionalität in die MemhoPhone Anwendung wurde festgestellt, dass die SIP Implementierung der Android Plattform ebenfalls durch den J-SIP Stack realisiert wurde, allerdings in einer modifizierten Version.
Dies hat zur Folge dass die J-SIP spezifischen Klassen bei dem Deployment der Anwendung auf dem Gerät nicht installiert werden, da der Namespace der Javapakete kollidiert. Die Pakete sind identisch und werden ignoriert, da sie bereits vorhanden sind. Da die in Android integrierte und modifizierte Version von J-SIP nicht API kompatibel ist, mit der des externen J-SIP Stacks, kam es zu Exceptions und Fehlfunktionen der SIP Implementierung, die durch eine Refaktorisierung des externen SIP Stacks gelöst wurden.
Die Wurzelpakete des externen J-SIP Stacks wurden umbenannt und in die Pakete von `de.consistec` umgezogen. Danach konnte der J-SIP Stack problemlos unter Android 2.3.3 eingesetzt werden.

DatagramSocket Bug Ein zeitaufwendiger Fehler, der augenscheinlich auf einen Softwarebug in der Implementierung des DatagramSockets unter Android zurückzuführen ist, war das Problem, dass SIP-Invite Nachrichten mit einer SDP-Nachricht als Payload nicht verschickt werden konnten. Die Komplexität des Softwarebugs wurde noch erhöht durch die Tatsache, dass er sich auf unterschiedlichen Geräten, trotz gleicher Androidversion, anders darstellte und eine Lokalisierung des Fehlers erschwert wurde.
Verwendete man die MemhoPhone Anwendung im Android Emulator des SDKs funktionierte sie fehlerfrei und die SIP Nachricht wurde korrekt ver-

schickt. Auf dem Nexus S wurde die Nachricht nicht verschickt und es wurde keine Exception geworfen. Erst nach einem Test auf einem anderen Gerät (*HTC Dream*) wurde beim Senden der SIP Nachricht eine Exception (`de.consistec.api.javax.sip.SipException: Operation not permitted`) geworfen und die Stelle des Softwarefehlers war ausgemacht. Allerdings war die Ursache, geschweige denn die Lösung des Problems noch nicht gefunden.

Mehrere Tests mit Zugriffen auf einen beliebigen DatagramSocket und auch auf den SIP Socket funktionierten problemlos, so dass immer noch unklar war weshalb Zugriffe auf den Socket einmal funktionieren und ein anderes mal fehlschlagen.

Das Problem wurde schließlich im Payload der SIP Nachricht lokalisiert, deren fest einprogrammierter SDP-Payload zum Fehlschlagen des Sendens via DatagramSocket führte. Änderte man den String des SDP-Payloads auf andere Werte, wurden die SIP/SDP Nachrichten versandt. Das Problem wurde schlussendlich durch die Verwendung der SdpFactory des SIP Stacks umgangen. Baute man den SDP-Payload der SIP Nachricht mit deren Hilfe zusammen, kam es zu keiner Exception mehr und alle SIP-Invite Nachrichten wurden auf allen getesteten Plattformen verschickt.

Zur Lokalisierung des Bugs wurde neben dem Debugger zusätzlich die Android Anwendung Shark[161] und tcpdump[162] auf dem Nexus S eingesetzt um zu verifizieren ob die SIP-Invite Nachrichten von dem Gerät verschickt werden.

Das Android Nexus S musste vorher gerootet werden damit tcpdump für das mitschneiden des Netzwerkverkehrs eingesetzt werden konnte. Als Rooting bezeichnet man man bei Android Smartphones das Freischalten des root Systemzugangs zum Android Betriebssystem[163].

13.10.3 JNI Nutzung

Die Verwendung des Java Native Interfaces ist recht aufwendig, da immer wieder der selbe Quellcode für wenig Funktionalität geschrieben werden muss. Dies wird auch als Boilerplate Code bezeichnet. So müssen beispielsweise übergebene Byterrays durch zusätzliche Funktionsaufrufe von der Javaebene kopiert werden, bevor sie benutzt werden können. Zusätzlich muss der Speicher der angeforderten Variablen, sofern es sich nicht um primitive Datentypen handelt, auch wieder freigegeben werden (Beispiel im Quellcodeauszug 13.3).

Noch aufwendiger ist der Aufruf von Java Funktionen aus dem, C-Quellcode. Hierfür muss zunächst anhand der Methodensignatur, wie dem Methodennamen, sowie den Übergabe- und Rückgabeparametern die MethodenID Angefordert werden, bevor

Listing 13.3: Bytearrays von Java nach C kopieren

```
jint Java_de_consistec_rtp_OrtpWrapper_sendZRTPThroughSocket(JNIEnv*
env, jobject thiz, jbyteArray data) {

    // Kopiere das bytearray von der Java VM
    jbyte* localRTPPacket = (*env)->GetByteArrayElements(env, data,
NULL);
    // Arbeite mit localRTPPacket hier
    // Gebe den Speicher des angeforderten Arrays wieder frei
    (*env)->ReleaseByteArrayElements(env, data, localRTPPacket, 0);
}
```

Listing 13.4: Java Methodenaufruf von C

```
jstring Java_de_consistec_rtp_OrtpWrapper_setupJNI(JNIEnv* env,
jobject thiz) {

    jclass cls = (*env)->GetObjectClass(env, thiz);
    jmethodID testFunctionID = (*env)->GetMethodID(env, cls,
"testCallToJavaFromC", "()V");
    // Testcall
    (*env)->CallVoidMethod(env, thiz, testFunctionID);
}
```

sie aufgerufen werden kann (dargestellt in Listing 13.4).

Zudem ist die Einarbeitung zeitaufwendig und die Dokumentation lässt stellenweise zu wünschen übrig. Die Aufwendige Verwendung des JNI ist ein weiterer Grund weshalb die Menge der deklarierten nativen Methoden und insbesondere der Datenaustausch zwischen Android SDK und NDK Ebene der Anwendung möglichst gering gehalten wurden. Weiterführende und ausführlicherer Informationen zum Java Native Interface sind unter [151] verfügbar.

13.10.4 RTP

ortp Die Inbetriebnahme der ortp Bibliothek verlief weitestgehend problemlos. In der online verfügbaren Version aus dem git Repository ist bereits ein `Android.mk` Makefile integriert, bei dem nur noch die Pfade angepasst werden mussten um es mit dem Android NDK zu kompilieren.

Problematischer war es mittels ortp über einen einzelnen UDP Socket gleichzeitig zu senden und zu empfangen. Das Senden funktionierte problemlos und wurde auch vom Kommunikationspartner empfangen. Zu Empfangen oder sogar gleichzeitiges senden und empfangen über den gleichen Socket funktionierte nicht.

Die Lösung dieser Problematik lag in der Konfiguration des ortp Stack be-

gründet. Eine Deaktivierung des ortp *Connected Mode* führte dazu, dass RTP Pakete gesendet und empfangen werden konnten. Die Lösung dieses Problems zu finden war daher problematisch, da bei der Funktion `void rtp_session_set_connected_mode( RtpSession* session, bool_t yesno )` weder an der Funktionsdefinition, noch in der Dokumentation erkennbar war, dass dieser Konfigurationsparameter darauf Einfluss haben kann.

libsrtp Integration Im Quellcode und in den Buildskripten der Win32 und *nix Version der ortp Bibliothek war die Integration der libsrtp vorgesehen. In den Buildskripten der Android Version war dies allerdings nicht mehr vorgesehen. Die libsrtp musste daher von Hand integriert werden auf Basis der Informationen der beiliegenden Buildskripte des GNU Build Systems[164].
Schwieriger gestaltete sich danach die Inbetriebnahme der libsrtp. In den Konfigurationsskripten musste der CPU-Typ auf CISC konfiguriert werden, damit die kryptographischen Algorithmen (namentlich, die *Cryptoengine*) der libsrtp funktionieren können. Und das trotzdem dass es sich bei der im Nexus S verbauten ARM CPU um eine RISC Architektur handelt. Nach der Konfigurationen auf den "falschen" CPU Typ CISC funktionierte die in ortp integrierte libsrtp tadellos und konnte für die Verschlüsselung der RTP Pakete verwendet werden.

ortp ZRTP Integration Die Bibliothek ortp hat eine Netzwerkpaketerkennung integriert und verwirft alle nicht erwünschten Pakete. In der verwendeten ortp Version aus dem SVN werden alle Pakete die nicht den Protokollen RTP, beziehungsweise RTCP zuzuordnen sind oder keine Pakete des Session Traversal Utilities for NAT (STUN) Protokolls[165] sind, verworfen. Um die ZRTP-Erkennung und Pufferung zu integrieren, musste die ortp Bibliothek an mehreren Stellen erweitert werden.
Zunächst wurde eine ZRTP Paketerkennung integriert die eingehende Pakete anhand der RTP Protokollversion und des ZRTP Magic-Cookies (ein Hexstring im ZRTP-Header der ZRTP-Pakete ausweist) prüft ob sie dem ZRTP Protokoll zuzuordnen sind. Die ZRTP Paketerkennung im RTP Stack zu integrieren hat auch zur Folge, dass dies wie im Kapitel 13.4 angesprochen, nicht durch die ZRTP4J Bibliothek in der Java Ebene durchgeführt werden muss. Wurde das eingehende ZRTP Paket identifiziert, wird es in eine vom RTP-Puffer separierte Queue abgelegt, die analog zum RTP-Puffer implementiert wurde, damit später nicht erneut der Pakettyp überprüft werden muss.
Der letzte Schritt zur ZRTP Integration in ortp war die Implementierung der ZRTP Empfangsfunktionen, mit denen man gepufferte ZRTP Pakete aus der ZRTP Empfangsqueue auslesen kann.

13.10.5 ZRTP

ZRTP4J war für die Verwendung mit der RTP- und Multimedia-Bibliothek Java Media Framework vorbereitet und die Dokumentation von ZRTP4J bezog sich auf die ins Java Media Framework integrierte Version. Um ZRTP4J mit ortp nutzen zu können, musste die Initialisierung und Integration von ZRTP4J von Hand implementiert werden.

Neben der Implementierung des ZRTP4J Callbackinterfaces für das ZRTP Paethandling, wurde auch ein Sendepuffer für ausgehende ZRTP Pakete implementiert. Der ZRTP Sendepuffer wurde durch Linux Systemqueues realisiert, die im Android NDK integriert sind (*<sys/queue.h>*). Die Verwendung eines Sendepuffers liegt nicht zuletzt auch darin begründet, dass die RTPManager und ZRTPManager Threads so unabhängig voneinander arbeiten können und nicht bis zur vollständigen Abarbeitung einzelner Pakete aufeinander warten müssen. So kommt es beim Senden und Empfangen von ZRTP oder RTP Paketen nicht zu Verzögerungen beim Zugriff auf den Socket, da koordiniert an der gleichen Stelle der Anwendung nacheinander für RTP und ZRTP auf den Socket zugegriffen wird.

Da ZRTP4J für die Realisierung aller kryptografischen Komponenten auf die externe Java Bibliothek Bouncy Castle Crypto APIs angewiesen ist, kommt es beim Deployment der Anwendung erneut zu Java Packagename Collisions und die Bouncy Castle Bibliothek wird nicht mitinstalliert. Android hat bereits die Bibliothek Bouncy Castle Crypto APIs (genau wie die Bibliothek J-SIP) integriert, wodurch es zu diesem Verhalten kommt.

ZRTP4J verwendet daher die in Android integriert Version der Bibliothek und funktioniert trotzdem tadellos. Daher wurde auch auf eine Refaktorisierung in andere Java Packages der Crypto Bibliothek verzichtet, bleibt als Option aber erhalten falls es doch noch zu Komplikationen durch die Verwendung der in Android integrierte Version der Bouncy Castle Crypto APIs kommen sollte.

13.11 Software Private Branch Exchange

Die Software die als SIP-Proxy und SIP-Registrar verwendet wurde, kurz gesagt, die Software die als Private Branch Exchange (PBX) eingesetzt wurde, war zunächst Asterisk[166]. Als Private Branch Exchange bezeichnete man früher eine Private Telefon- oder Nebenstellenanlage. Dieser Begriff wird heute auch für Softwaretelefonanlage verwendet.

Asterisk wurde zunächst in der Version 1.6.2.9-2 unter Debian Linux eingesetzt. Da das ZRTP-Protokoll nicht unterstützt wurde und damit auch nicht weitergeleitet werden konnte (*RTP-Relay*) und auch die Direkte Medienverbindung ohne Asterisk

als RTP-Relay nicht funktionierte, wurde Asterisk auf Version 1.8.4.4 aktualisiert. Auch bei dieser Version wurden eingehende ZRTP Pakete verworfen und nicht weitergeleitet. Eine Direkte RTP Verbindung konnte bei dieser Asterisk Version ebenfalls nicht aufgebaut werden.

Aus diesem Grund wurde Asterisk gegen die ebenfalls freie PBX Freeswitch[167] getauscht. Freeswitch kann mit ZRTP Support kompiliert und eingesetzt werden. Freeswitch leitet eingehende ZRTP Pakete weiter und kann auch als ZRTP-Relay agieren und damit ZRTP-Pakete weiterleiten.
Mit Freeswitch konnte die ZRTP Funktionalität der Anwendung zur sicheren Sprachkommunikation unter Android getestet werden

Der Wechsel der Software PBX hatte noch andere positive Nebeneffekte. Durch den zweimaligen Wechsel der Software PBX wurden zudem Fehler in der Implementierung der SIP Funktionalität festgestellt und behoben die andernfalls unentdeckt geblieben wären.

Abschließend kann über die ZRTP Unterstützung bei den frei verfügbaren Software PBXs gesagt werden, dass sie noch unzureichend ist. Außer der Software Freeswitch konnte keine Software PBX gefunden werden die ZRTP direkt unterstützt. Jede Software PBX die eine direkte Medienverbindung zulässt, kann allerdings für die Signalisierung von RTP Verbindungen verwendet werden, über die man dann auch ZRTP Pakete zwischen den Kommunikationspartner transportieren kann. Mittels ZRTP-Masquerading kann zudem jede Software PBX, die auch RTP unterstützt, für die Weiterleitung von ZRTP-Nachrichten verwendet werden[168]. ZRTP-Masquerading ist allerdings nicht Teil des offiziellen Standards unter [123].

Teil IV

Schluss

14 Zusammenfassung

Mit dieser Arbeit konnte belegt werden, dass eine mobile Anwendung zur verschlüsselten Sprachkommunikation auf Basis des Android Betriebssystems erstellt werden kann und welche Technologien dazu am geeignetsten sind.

Bei der Erstellung der Arbeit lag der Schwerpunkt auf zwei Bereichen. Zunächst wurde eine *Technologieevaluation* durchgeführt um einen Überblick über die Technologien zu erstellen, die man zur abhörsicheren Sprachkommunikation einsetzen kann. Diese wurden im Kontext der eigens zusammengestellten *Anforderungen an sichere digitale Sprachkommunikation* miteinander verglichen und bewertet.

Der zweite Schwerpunkt lag in der prototypischen *Implementierung* einer Anwendung zur verschlüsselten Sprachkommunikation, deren Design ausführlich vorgestellt wurde. Dabei wurde insbesondere auch auf die Schwierigkeiten und Fallstricke eingegangen, auf die man bei der Softwareentwicklung unter Android und bei der Entwicklung von abhörsicherer Sprachkommunikationssoftware trifft. Insbesondere der Entwicklung von nativen Anwendungen in der Programmiersprache C und dem Java Native Interface auf Basis des Android OS kam besondere Beachtung zuteil.

Es ist mit der vorliegenden Arbeit gelungen eine Übersicht über aktuell eingesetzte Technologien zur Signalisierung, zum Medientransport und für das Key Management des Secure Real-Time Protokolls zu erstellen. Diese Technologieevaluation kann für andere Arbeiten als Basis dienen.

Mit dem Prototyp zur verschlüsselten Sprachkommunikation konnte das Konzept und die Funktion der evaluierten Technologien verifiziert werden. Aufgrund der modularisierten Architektur, dem durchdachten Design und der Verwendung von Bibliotheken die unter BSD-Lizenz veröffentlicht sind, wurde eine solide Basis für ein kommerziell vermarktbares Produkt geschaffen.

15 Ausblick

Die in dieser Master-Thesis erstellte Evaluation und der darauf basierende Prototyp erfüllen alle aufgestellten Anforderungen an die sichere Sprachkommunikation. Die prototypische Implementierung zeigt, dass man auf der Android Plattform eine Anwendung zur verschlüsselten Sprachkommunikation realisieren kann.

In diesem Kapitel werden einige Perspektiven für die Anwendung zur sicheren Sprachkommunikation vorgestellt und Ideen für eine Fortführung dieser Arbeit gegeben.

Anonymität Die in dieser Arbeit vorgestellten Technologien gewährleisten keinerlei Anonymität der Kommunikationspartner auf dem Kommunikationsweg. Es wäre allerdings wünschenswert, dass zwei Peers anonym miteinander kommunizieren können. Zum einen für Außenstehende anonym, so dass nicht nachgewiesen werden kann wer miteinander kommuniziert, zum anderen aber auch für die Kommunikationspartner selbst anonym, um die Identität zu Verschleiern.

Aufgrund der Authentifizierungsfeatures von ZRTP können sich zwei Kommunikationspartner zwar wiedererkennen nachdem sie einmal miteinander kommuniziert haben, dadurch kann aber nicht auf die Identität des Peers geschlossen werden.

Mögliche Einstiegspunkte sind die Webseite des TOR Projekts[169], eine Papersammlung des freien anonymisierenden Darknets Freenet[170], sowie zwei Erweiterungen des SIP Protokolls – dem User-Agent-Driven Privacy Mechanism for SIP[171] und den Privacy Mechanism for the Session Initiation Protocol[172].

ZRTP Masquerading Ein weiteres lohnenswertes Feature für die Anwendung zur sicheren Sprachkommunikation wäre die Integration von ZRTP Masquerading[168]. Die Erweiterung ist zwar kein Bestandteil des offiziellen ZRTP Standards, sie wurde aber von ZORG[173] publiziert einem Projekt, das von einer Sicherheitsfirma gesponsert wird.

Durch die Verwendung von ZRTP Masquerading, werden ZRTP Pakete so maskiert, dass sie von einem RTP-Relay Server als RTP Pakete erkannt werden und auch weitergeleitet werden. Dadurch wird keinerlei Unterstützung von

Seiten der Software PBX für das ZRTP Protokoll benötigt, da die ZRTP Unterstützung in Software PBXs ohnehin nicht sehr weit verbreitet ist.
Ein möglicher Einstiegspunkt für die Umsetzung dieses Features ist die Webseite der ZRTP-Bibliothek von ZORG[173], die Beschreibung des ZRTP Masquerading von ZORG[168], sowie die offiziellen RFCs von RTP[85] und ZRTP[123].

Teil V

Anhang

A URN

URN steht für User Requirements Notation und ist eine graphische Modellierungssprache zur Erhebung, Analyse, Spezifizierung und Validierung von Anforderungen an Systeme. Die User Requirements Notation ist untergliedert in zwei Subsprachen, der Goal-Oriented Requirements Language (GRL) und in die Sprache Use Case Maps (UCM) (Vgl. [41, 42]).

GRL Mit der Goal-Oriented Requirements Language werden die nicht-funktionalen Anforderung an ein System formuliert um Zusammenhänge und Anforderungen an das Gesamtsystem und Einzelkomponenten darzustellen und zu bewerten.

UCM Durch Use Case Maps können funktionale Anforderungen an ein System dargestellt und bewertet werden. Mit Use Case Maps werden Anwendungsszenarien für die Komponenten aus den GRL analysiert und bewertet.

Beide Diagrammtypen können nach ihrer Erstellung automatisch evaluiert werden, wenn die Diagramme mit Hilfe einer Software, wie beispielsweise jUCMNav[174] erstellt wurden. jUCMNav erwies sich in der Praxis allerdings als langsam zu bedienendes und fehlerhaftes Werkzeug.

Die URN-Diagramme in dieser Thesis wurden mit Hilfe von Visio[175] und dem Visio-Plugin Sandrila SDL[176] erstellt, das die URN Visio-Shapes in der freien Version integriert hat. Visio ermöglicht allerdings keine automatische Evaluierung der Diagramme.

Der praktische Einsatz von URN ist nicht sehr weit verbreitet, was sich auch in der Anzahl der online verfügbaren Informationen bemerkbar macht – Diese sind nur sehr spärlich verfügbar. So existiert zwar ein sehr kurzer Artikel über URN in der deutschen Wikipedia[177], in der englischen Wikipedia beispielsweise fehlen Information über URN allerdings völlig.
Zum Einarbeiten verweise ich daher an den offiziellen Standard, der einsteigerfreundlich strukturiert und geschrieben ist.

Meiner Meinung nach ist URN als Werkzeug für die initiale Anforderungsanalyse gut geeignet. Durch die Möglichkeit verschiedene Alternativen für die Lösung eines Problems darzustellen und zusätzlich auch zu bewerten, kann es gut in Meetings oder Arbeitsgruppen eingesetzt werden um in Kooperation ein Problem darzustel-

len und ansatzweise zu lösen.
Die Einarbeitung in URN ist sehr leicht und schnell, da einfach verständliche Symbole verwendet werden. Mit Hilfe einer Übersichtstabelle, die im URN-Standard als Anhang mitgeliefert wird, kann jeder auf Anhieb URN-Diagramme lesen.

Ich würde für die Anforderungsanalyse allerdings SysML bevorzugen, da SysML nicht auf die Anforderungsanalyse beschränkt ist, sondern auch für die Beschreibung der Systemarchitektur und Vorgänge geeignet ist. SysML wird im folgenden Kapitel vorgestellt.

B SysML

Die Systems Modeling Language[178], oder kurz SysML ist ebenso wie URN eine Modellierungssprache. SysML ist genau wie URN auch für die Anforderungsanalyse geeignet und kann URN ersetzen. Die Systems Modeling Language ist im Gegensatz zu URN aber nicht auf die bloße Anforderungsanalyse beschränkt, sondern ist eine universelle Modellierungssprache, ähnlich wie die Unified Markup Language UML[179].
SysML ist eng verwandt mit UML, ist allerdings nicht auf die Analyse, Design und Implementierung von objektorientierten Softwarearchitekturen beschränkt, sondern vielseitiger einsetzbar. SysML baut auf UML auf und erweitert den Funktionsumfang und das Einsatzgebiet auf die Spezifizierung, Analyse, Design, Verifizierung und Validierung von Systemen im Allgemeinen.
Einige Diagrammtypen wurden direkt von UML übernommen, wie das Sequence und das Statemachine Diagramm. Andere wurden modifiziert, wie das Activity oder das Block Diagramm und zwei Diagrammtypen sind vollständig neu entwickelt worden (Requirement Diagramm und Parametrics Diagramm) (Vgl. [178]).

SysML wurde in dieser Thesis in allen Diagrammen eingesetzt, mit denen das Konzept und die Implementierung des Softwaresystems MemhoPhone beschrieben wurde. Der modulare Aufbau der Diagramme und die Möglichkeit unterschiedliche Abstraktionsebenen in einem Diagramm zu vereinen, machen es aus meiner Sicht hervorragend geeignet um Softwaresysteme zu beschreiben. Einzelne Module können spezifisch benannt werden und dann in separaten Diagrammen detaillierter erläutert werden. Dadurch hat man die Möglichkeit, Diagramme übersichtlich zu gestalten, ohne den Detailgrad einzelner Komponenten außer Acht zu lassen.

SysML ist aus meiner Sicht absolut empfehlenswert für die Beschreibung und das Design von Softwaresystemen und Systemen im Allgemeinen. Der Einstieg fiel mir sehr leicht, da die SysML-Dokumentation dank sehr zahlreicher Beispiele hervorragend geeignet ist, die Sprache zu erlernen. Für weiterführende Informationen verweise ich an dieser Stelle auf die offizielle Dokumentation oder an Literatur von [180] zu diesem Thema.

Erstellt man Diagramme mit spezieller Modellierungssoftware, wie beispielsweise dem SysML/UML Modellierungswerkzeug Papyrus[181], kann man sich Quellcode

generieren lassen und dadurch ein Softwaresystem vollständig grafisch entwerfen. Die SysML-Diagramme in dieser Thesis wurden mit Visio und einem SysML/UML Templatepaket erstellt das unter [182] frei verfügbar ist.

C Kryptographische Grundlagen

In diesem Kapitel werden einige kryptographische Grundlagen und Algorithmen erläutert die für das Verständnis dieser Arbeit hilfreich sind. Für tiefer gehende Information zu Kryptographie verweise ich an das Buch von [144] und an das Buch von [183], die beide einen sehr guten Einstieg in kryptographische Verfahren bieten. Die Erläuterungen in diesem Kapitel basieren weitestgehend auf diesen beiden Büchern.

C.1 Blockchiffren

Bei der Verschlüsselung mit Blockchiffren werden Blöcke fester Länge auf Blöcke derselben Länge abgebildet. Man kann sie auf unterschiedliche Weise benutzen und beliebig lange Texte damit verschlüsseln.

Listing C.1: Blockchiffren[183, Def. 4.6.1]

Unter einer Blockchiffre versteht man ein Verschlüsselungsverfahren, in dem Klartext- und Schlüsseltextraum die Menge Σ^n aller Wörter der Länge n über einem Alphabet Σ sind. Die Blocklänge n ist eine natürliche Zahl.

Um längere Texte zu verschlüsseln, können Blockchiffren in unterschiedlichen Verfahren angewandt werden (Vgl. [183]).

ECB *Electronic Codebook Mode*. Ein Text wird in gleichlange Blöcke der Größe n aufgeteilt die dann einzeln verschlüsselt werden. Fehlende Zeichen des letzten Blockes werden durch Zufallszahlen auf die Länge n ergänzt. Ein Problem des ECB-Modus bei Blockchiffren ist, dass ein bestimmter Klartext immer zum gleichen Ciphertext verschlüsselt wird (Vgl. [183]).

CBC *Cipherblock Chaining Mode*. Dieser Operationsmodus von Blockchiffren hebt die Nachteile des ECB auf. Bei diesem Verfahren hängt der Ciphertext nicht nur vom Klartext und dem Schlüssel ab, es werden bei der Verschlüsselung auch die vorangehenden Blöcke mit verrechnet. Die Verschlüsselung ist damit kontextabhängig. Gleiche Muster an unterschiedlichen Stellen werden dadurch unterschiedlich verschlüsselt.

Durch die Verwendung unterschiedlicher Initialisierungsvektoren, werden im CBC-Modus der gleiche Klartext unterschiedlich verschlüsselt (Vgl. [183]).

CFB *Cipher Feedback Mode*. Der CBC-Modus ist gut geeignet um lange Nachrichten zu verschlüsseln, allerdings muss gewartet werden bis ein gesamter Ciphertextblock fertig verschlüsselt wurde, bevor mit dem Entschlüsseln begonnen werden kann.
Der CFB-Modus behebt dieses Problem, indem Klartextblöcke nicht direkt durch die Blockverschlüsselungsfunktion E_k verschlüsselt werden, sondern durch Addition $mod\,2$ entsprechender Schlüsselblöcke verschlüsselt. Dadurch kann die Entschlüsselung auf Empfängerseite nahezu simultan mit der Verschlüsselung begonnen werden.
Der CFB-Modus kann bei Public-Key Kryptographie nicht eingesetzt werden, da Alice und Bob den gleichen Schlüssel benötigen (Vgl. [183]).

OFB *Output Feedback Mode*. Der OFB führt Änderungen am CFB-Modus ein um auf Übertragungsfehler besser reagieren zu können. Wenn Bits bei der Übertragung verändert werden, hat das nur Auswirkungen auf ein Bit des Klartextes. Dies liegt darin begründet, dass die Verschlüsselung von Klartextblöcken nicht vom vorhergehenden Klartext abhängt, sondern nur von der Position.
Ein Nebeneffekt dieses Verfahrens ist, dass ein Klartext leichter verändert werden kann, wenn man den OFB-Ciphertext manipuliert, als ein Ciphertext der im CBC-Modus verschlüsselt wurde.
Der OFB-Modus macht aus einem Blockchiffre einen Stromchiffre (Vgl. [183]).

ICM *Integer Counter Mode*. Der ICM funktioniert ähnlich wie der OFB-Modus und kann wie ein Stromchiffre betrieben werden. ICM verwendet allerdings eine number-used-once als Initialisierungsvektors eines jeden Blockes, der bei jedem Block hochgezählt wird. Dadurch bringt der ICM die Vorteile mit sich, dass er leicht parallelisierbar und beweisbar sicher ist und das Chiffretextblöcke in zufälliger Reihenfolge entschlüsselt werden können (Vgl. [184]).

C.2 Stromchiffren

Stromchiffren konvertieren Klartext bitweise zu Chiffretext. Ein Schlüsselstromgenerator generiert hierzu einen Schlüsselstrom, der mit dem Klartext Bit für Bit XOR verknüpft wird. Die Sicherheit beruht darauf, dass der Schlüsselstromgenerator kontinuierlich Zufallszahlen liefert. Liefert der Schlüsselstromgenerator kontinuierlich Nullen, wird der Klartext gar nicht verschlüsselt. Zur Verschlüsselung und zur Entschlüsselung, werden die Schlüsselstromgeneratoren an einen Schlüssel gekoppelt, der einen Bitstrom anhand des Schlüssels generiert. (Vgl. [144]).

Stromchiffren verallgemeinern das Prinzip einiger Blockchiffre Operationsmodi, dass Klartextblöcke kontextabhängig verschlüsselt werden. Sie lassen sich zudem effizient in Hardware implementieren (Vgl. [183]).

C.3 AES

Laut [183] wurde der *Advanced Encryption Standard* (AES) vom National Institute of Standards and Technology nach einem vierjährigen Auswahlprozess um einen Nachfolger des DES Algorithmus zu finden, 2001 standardisiert. AES löst damit DES ab. Der Chiffre des AES wird Rijndael-Chiffre genannt, wobei der AES ein Spezialfall des Rijndael-Chiffres ist.

Der Rijndael-Chiffre ist ein Blockchiffre und besitzt eine variable Blockgröße von 128, 192 oder 256 Bit. Die Schlüssellänge ist ebenfalls variabel und kann 128 Bit lang (AES-128), 192 Bit lang (AES-192) oder 256 Bit lang sein (AES-256) (Vgl. [74]).

C.4 Diffie-Hellman Schlüsselaustausch

Laut [144] ist der Diffie-Hellman Algorithmus (DH), der erste Public-Key-Algorithmus und wurde 1976 patentiert. Die Sicherheit des Algorithmus liegt in dem Problem der Berechnung des diskreten Logarithmus in endlichen Körpern. Berechnet man stattdessen Potenzen in Körpern mit der gleichen Ordnung, ist dies im direkten Vergleich eine einfache mathematische Berechnung.

Das Diffie-Hellman Schlüsselaustauschprotokoll eignet sich wenn zwei Kommunikationspartner einen gemeinsamen Schlüssel über einen öffentlichen Kanal berechnen möchten. Besonderes Merkmal des DH-Protokolls ist, dass die beiden Kommunikationspartner vorher keinen Kontakt zueinander gehabt haben müssen. Der Ablauf des DH-Protokolls ist in Definition C.2 dargestellt.

Selbst wenn der Kommunikationskanal abgehört wurde, ist der Angreifer jetzt nicht im Besitz des Schlüssels, da er nur im Besitz von n, g, X und Y ist. Das Problem lässt sich ausschließlich durch die Berechnung des diskreten Logarithmus lösen.

Die Sicherheit ist von der Auswahl von n und g abhängig. $(n-1)/2$ sollte eine Primzahl sein und n sollte sehr groß gewählt sein. g kann eine beliebige Zahl sein, so lange es primitiv $mod\, n$ ist oder eine große Untergruppe der multiplikativen Gruppe $mod\, n$ erzeugen kann (Vgl. [144]).

Listing C.2: Diffie-Hellmann Schlüsselaustausch

1. Alice wählt eine große Zufallszahl x und sendet Bob:
   $X = g^x mod\, n$
2. Bob wählt ebenfalls eine große Zufallszahl y und sendet an Alice:
   $Y = g^y mod\, n$
3. Als dritten Schritt berechnet Alice:
   $k = Y^x mod\, n$
4. Bob kann dann genau wie Alice k' berechnen:
   $k^{'} = X^y mod\, n$
5. Alice und Bob sind nun in Besitz des gleichen Schlüssels:
   $k = k^{'}$

C.5 Elliptische Kurven

Elliptische Kurven über endliche Körper, insbesondere über Primkörper sind für die Kryptographie sehr wichtige Funktionen. Kryptographie mit elliptischen Kurven wird auch als *Elliptic Curve Cryptography* bezeichnet (ECC).

Elliptische Kurven können zur Public Key Kryptographie eingesetzt werden und sind bisher die wichtigste Alternativen zu Verfahren die auf dem RSA Algorithmus basieren. Diese Alternative ist auch wichtig, da niemand für die Sicherheit von RSA garantieren kann[183].

Der entscheidende Vorteil von ECC ist die Länge der Schlüssel und dem daraus resultierenden geringeren Berechnungsaufwand gegenüber RSA-basierenden Verfahren. Wo RSA-Verfahren 1024 Bit lange Schlüssel benötigen, kommen ECC-Verfahren mit 162 Bit langen Schlüsseln aus. Die Berechnung von elliptischen Kurven ist zwar eigentlich komplexer und aufwändiger, durch die geringe Schlüssellänge wird dies aber kompensiert.

Verfahren die auf dem diskreten Logarithmus beruhen, wie der hier vorgestellte Diffie-Hellman Algorithmus lassen sich sehr einfach auf Elliptische Kurven portieren und haben dann ebenfalls den Vorteil bei geringerer Schlüssellänge, weniger Speicher zu verbrauchen und schneller berechenbar zu sein. Eine Elliptische Kurve über endliche Primkörper ist wie folgt definiert:

Sei F_p ein endlicher Primkörper und gelte für $a, b \in F_p$:

Listing C.3: Elliptische Kurve (*Bedingung für non-Singularität*)

$$4a^3 + 27b^2 \not\equiv 0$$

Eine elliptische Kurve $E(F_p)$ über F_p ist über die Parameter $a, b \in F_p$ definiert und besteht aus der Menge an Lösungen oder Punkten $P = (x, y)$ für $x, y \in F_p$ zur Gleichung:

Listing C.4: Elliptische Kurve (Gleichung)

$$y^2 \equiv x^3 + ax + b \, mod \, p$$

D ZRTP-Handshake

Ein vollständig mitgeschnittener ZRTP-Handshake wird in Abbildung D.1 dargestellt. Darin ist der in Kapitel 10.3 (Abb. 10.1) vereinfacht dargestellte ZRTP-Handshake, gut nachvollziehbar.

Zu beachten sind in dieser Abbildung die mehrfach versendeten Pakete, wenn deren Erhalt durch den Kommunikationspartner noch nicht bestätigt wurde.

282 79.967830	10.10.35.112	10.10.21.18	ZRTP	210 Hello Packet
283 79.969951	10.10.35.112	10.10.21.18	ZRTP	70 HelloACK Packet
291 80.065427	10.10.21.18	10.10.35.112	ZRTP	182 Hello Packet
302 80.265769	10.10.21.18	10.10.35.112	ZRTP	182 Hello Packet
314 80.453839	10.10.21.18	10.10.35.112	ZRTP	70 HelloACK Packet
315 80.466178	10.10.21.18	10.10.35.112	ZRTP	182 Hello Packet
316 80.505948	10.10.21.18	10.10.35.112	ZRTP	174 Commit Packet
318 80.666499	10.10.21.18	10.10.35.112	ZRTP	174 Commit Packet
320 80.967038	10.10.21.18	10.10.35.112	ZRTP	174 Commit Packet
352 81.586403	10.10.21.18	10.10.35.112	ZRTP	174 Commit Packet
392 82.233774	10.10.35.112	10.10.21.18	ZRTP	70 HelloACK Packet
393 82.234036	10.10.35.112	10.10.21.18	ZRTP	70 HelloACK Packet
394 82.234345	10.10.35.112	10.10.21.18	ZRTP	174 Commit Packet
395 82.234752	10.10.35.112	10.10.21.18	ZRTP	526 DHPart1 Packet
396 82.235051	10.10.35.112	10.10.21.18	ZRTP	526 DHPart1 Packet
397 82.235457	10.10.35.112	10.10.21.18	ZRTP	526 DHPart1 Packet
398 82.241545	10.10.35.112	10.10.21.18	ZRTP	526 DHPart1 Packet
403 82.293743	10.10.21.18	10.10.35.112	ZRTP	526 DHPart2 Packet
412 82.458257	10.10.21.18	10.10.35.112	ZRTP	526 DHPart2 Packet
429 82.758710	10.10.21.18	10.10.35.112	ZRTP	526 DHPart2 Packet
461 83.359566	10.10.21.18	10.10.35.112	ZRTP	526 DHPart2 Packet
518 84.460751	10.10.35.112	10.10.21.18	ZRTP	134 Confirm1 Packet
519 84.461094	10.10.35.112	10.10.21.18	ZRTP	134 Confirm1 Packet
520 84.461291	10.10.35.112	10.10.21.18	ZRTP	134 Confirm1 Packet
521 84.461442	10.10.35.112	10.10.21.18	ZRTP	134 Confirm1 Packet
523 84.473881	10.10.21.18	10.10.35.112	ZRTP	134 Confirm2 Packet
524 84.478023	10.10.35.112	10.10.21.18	ZRTP	70 Conf2ACK Packet

Abbildung D.1: ZRTP-Handshake (Wireshark Mitschnitt)

Literaturverzeichnis

[1] U.N.G. Assembly. "Universal declaration of human rights". In: *Resolution adopted by the General Assembly* 10 (1948), p. 6. URL: http://www.ohchr.org/EN/UDHR/Pages/Language.aspx?LangID=ger.

[2] Bundesrepublik Deutschland. *Grundgesetz für die Bundesrepublik Deutschland*. Deutschland, May 1949. URL: http://www.bundestag.de/dokumente/rechtsgrundlagen/grundgesetz/index.html.

[3] European Union. *Indect Project Website*. 2011. URL: http://www.indect-project.eu/.

[4] European Commission. *Indect Cordis Project Website*. 2011. URL: http://cordis.europa.eu/fetch?CALLER=FP7_PROJ_EN&ACTION=D&DOC=4&CAT=PROJ&QUERY=011f30e52539:b685:00e1e967&RCN=89374.

[5] Kai Biermann. "Indect – der Traum der EU vom Polizeistaat". In: *Zeit Online* (Sept. 2009). URL: http://www.zeit.de/digital/datenschutz/2009-09/indect-ueberwachung.

[6] Matthias Monroy. "Allround-System für europäische Homeland Security". In: *Telepolis (online)* (Jan. 2010). URL: http://www.heise.de/tp/artikel/31/31802/1.html.

[7] European Parliament. *BERICHT über die Existenz eines globalen Abhörsystems für private und wirtschaftliche Kommunikation (Abhörsystem ECHELON) (2001/2098 (INI))*. Tech. rep. European Uniom, 2001.

[8] W. Mazurczyk and K. Szczypiorski. "Steganography of VoIP Streams". In: (2008).

[9] *GSM Sniffing*. Chaos Computer Club. 2010. URL: http://events.ccc.de/congress/2010/Fahrplan/attachments/1783_101228.27C3.GSM-Sniffing.Nohl_Munaut.pdf.

[10] Michael Friedrichs. "Hacker-Konferenz: Handygespräche abhören und mitschneiden". In: *teltarif.de* (Aug. 2010). URL: http://www.teltarif.de/imsi-catcher-hacker-lauschangriff-defcon/news/39602.html.

[11] Uli Ries. “Lauschangriff für jedermann”. In: *Spiegel Online* (Aug. 2010). URL: http://www.spiegel.de/netzwelt/netzpolitik/0,1518,709624,00.html.

[12] Björn Brodersen. “Abhören von Telefonaten ist relativ leicht”. In: *teltarif.de* (2004). URL: http://www.teltarif.de/arch/2004/kw09/s12985.html.

[13] Directorate General for Telecommunication and Post of the Ministry of Economic Affairs (EZ). *Transport of Intercepted IP Traffic*. Tech. rep. Directorate General for Telecommunication and Post of the Ministry of Economic Affairs (EZ), 2002. URL: http://www.gliif.org/LI_standards/TIIT-v1.0.0.0.pdf.

[14] IAB and IESG. *IETF Policy on Wiretapping*. Tech. rep. IETF RFC 2804, May, May 2000. URL: https://tools.ietf.org/html/rfc2804.

[15] Bundesjustizamt. *Übersicht Telekommunikationsüberwachung (Maßnahmen nach Paragraph 100a StPO) für 2009*. Tech. rep. Bundesjustizamt, 2009. URL: http://www.bundesjustizamt.de/cln_108/nn_2036868/DE/Themen/Buergerdienste/Justizstatistik/Telekommunikation/downloads/Uebersicht__TKUE__2009,templateId=raw,property=publicationFile.pdf/Uebersicht_TKUE_2009.pdf.

[16] Bundesrepublik Deutschland. *Strafprozessordnung*. Deutschland, 1877.

[17] Inc. Gartner. “Gartner Says Worldwide Mobile Phone Sales Grew 17 Per Cent in First Quarter 2010”. In: *Gartner Newsroom* (2010). URL: http://www.gartner.com/it/page.jsp?id=1372013.

[18] Inc. Gartner. “Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent”. In: *Gartner Newsroom* (2011). URL: http://www.gartner.com/it/page.jsp?id=1764714.

[19] Matt Hamblen. “Gartner: Mobile phones to overtake landlines in business by 2011”. In: *Computerworld (Online)* (2008). URL: http://www.computerworld.com/s/article/9123733/Gartner_Mobile_phones_to_overtake_landlines_in_business_by_2011.

[20] ETSI. *Digital Cellular Telecommunications System (Phase 2+); Mobile Application Part (MAP) Specification*. Tech. rep. ETSI, July 2000.

[21] B. Elad, B. Eli, and K. Nathan. *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication*. A5/1 Hack. 2003. URL: http://cryptome.org/gsm-crack-bbk.pdf.

[22] O. Dunkelman, N. Keller, and A. Shamir. "A practical-time attack on the A5/3 cryptosystem used in third generation GSM telephony". In: *Proceedings of the 30th Annual Cryptology Conference (CRYPTO 2010)*. 2010. URL: http://eprint.iacr.org/2010/013.pdf.

[23] E. Biham, O. Dunkelman, and N. Keller. "A related-key rectangle attack on the full KASUMI". In: *Advances in Cryptology-ASIACRYPT 2005* (2005), pp. 443–461. URL: http://www.ma.huji.ac.il/~nkeller/kasumi.ps.

[24] BSI. *Sicherheit von Mobiltelefonen nach GSM-Standard*. Tech. rep. Bundesamt für Sicherheit in der Informationstechnik, 2009. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Sicherheitsberatung/Sicherheitshinweise/2009-07-10_Sicherheitshinweis_GSM_pdf.pdf.

[25] H. Mosemann and M. Kose. *Android*. Hanser Verlag, 2009. ISBN: 3446417281.

[26] OpenHandsetAlliance. "Open Handset Alliance Releases Android SDK". In: *www.openhandsetalliance.com* (2007). URL: http://www.openhandsetalliance.com/press_111207.html.

[27] Canalys. "Canalys Smart Phone Analysis, Quarterly Shipment Data, Q4 2010". In: *Website* (2011). URL: http://www.canalys.com/pr/2011/r2011013.html.

[28] Inc. Gartner. *Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012*. 2011. URL: http://www.gartner.com/it/page.jsp?id=1622614 (visited on 09/15/2011).

[29] Alexander Sirotkin. "The Java API to Android's telephony stack". In: *Linux J.* 2009 (183 2009). ISSN: 1075-3583. URL: http://www.linuxjournal.com/magazine/java-api-androids-telephony-stack.

[30] ISO. *ISO/IEC 9945-1 (1996). ISO/IEC Standard 9945-1:1996 Information Technology -Portable Operating System Interface (POSIX)- Part 1: System Application Program Interface (API) [C Language]*. Tech. rep. Institute of Electrical and electronic Engineers, 1996.

[31] developer.android.org. *Android Architecture*. 2011. URL: http://developer.android.com/images/system-architecture.jpg (visited on 09/27/2011).

[32] Oliver Diedrich. "Die Architektur von Android". In: *c't* 7 (2011), pp. 122–127.

[33] opensource.org. *Open Source Initiative OSI - The BSD License:Licensing*. 2011. URL: http://www.opensource.org/licenses/bsd-license.php (visited on 09/22/2011).

[34] R. Stewart. *Stream Control Transmission Protocol*. Tech. rep. IETF RFC 4960 September, 2007. URL: http://tools.ietf.org/html/rfc4960.

[35] Gerd Siegmund. *Technik der Netze*. 5. Heidelberg: Hüthig, 2002.

[36] A. Badach. *Voice over IP-die Technik: Grundlagen, Protokolle, Anwendungen, Migration, Sicherheit*. Hanser Verlag, 2007. ISBN: 3446406662.

[37] protocols.com. *ISDN Protocols described*. 2011. URL: http://www.protocols.com/pbook/isdn.htm.

[38] ITU Recommendation. *Q.700 Specifications of Signalling System No. 7*. Mar. 1993. URL: http://www.itu.int/rec/T-REC-Q.700-199303-I/en.

[39] ITU Recommendation. *H.323 Packet-based multimedia communications systems*. Dec. 2009. URL: http://www.itu.int/rec/T-REC-H.323-200912-I.

[40] M. Spencer, K. Shumard, B. Capouch, et al. *IAX2 - Inter-Asterisk eXchange Version 2*. Tech. rep. IETF RFC 5456, February, 2010. URL: http://tools.ietf.org/html/rfc5456.

[41] ITU Recommendation. *Z.150 - User Requirements Notation (URN) – Language requirements and framework*. 2011. URL: http://www.itu.int/itudoc/itu-t/aap/sg17aap/history/z150/z150.html.

[42] ITU Recommendation. *Z.151 - User requirements notation (URN) – Language definition*. 2008. URL: http://www.itu.int/rec/T-REC-Z.151/en.

[43] ITU Recommendation. *G. 114 - One-way transmission time*. 2000. URL: http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-G.114-200305-I!!PDF-E&type=items.

[44] ITU Recommendation. *P.800 - Methods for subjective determination of transmission quality*. 1996. URL: http://www.itu.int/rec/T-REC-P.800-199608-I/en.

[45] ITU Recommendation. *BS.1534 - Method for the subjective assessment of intermediate quality level of coding systems*. 2003. URL: http://www.itu.int/rec/T-REC-P.800-199608-I/en.

[46] ITU Recommendation. *G. 711 - Pulse Code Modulation (PCM) of Voice Frequencies*. 1988. URL: http://www.itu.int/rec/T-REC-G.711-198811-I/en.

[47] S. Aidarous et al. *Managing IP Networks: Challenges and Opportunities*. John Wiley & Sons, Inc., 2003.

[48] A.H.M. Amin. "VoIP Performance Measurement Using QoS Parameters". In: *International Conference on Innovations in Information Technology 2005*. Citeseer. 2005. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.6117&rep=rep1&type=pdf.

[49] D.R. Kuhn, T.J. Walsh, and S. Fries. *Security Considerations for Voice Over IP Systems*. Tech. rep. NIST, 2005.

[50] A. Adelsbach et al. "VoIPSEC Studie zur Sicherheit von Voice over Internet Protocol". In: *Bundesamt für Sicherheit in der Informationstechnik* (2005).

[51] E. Eren and D. Kai-Oliver. "Voice over IP Security Mechanisms State of the art, risks assesment, concepts and recommendations". In: *Internetworking 2006* (2007). URL: http://www.decoit.de/cms/upload/pdf/IW06_eren_detken_VoIP_final.pdf.

[52] P. Kocher, J. Jaffe, and B. Jun. "Differential Power Analysis". In: *Crypto '99 Proceedings*. 1998. URL: http://www.cryptography.com/resources/whitepapers/DPA.pdf.

[53] P. Kocher. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". In: *Advances in Cryptology CRYPTO '96*. Springer. 1996, pp. 104–113. URL: http://www.cryptography.com/public/pdf/TimingAttacks.pdf.

[54] C.V. Wright et al. "Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations". In: *2008 IEEE Symposium on Security and Privacy*. Ieee. 2008, pp. 35–49.

[55] T. Ylonen et al. *The Secure Shell (SSH) Authentication Protocol*. Tech. rep. IETF RFC 4252 January, 2006. URL: https://tools.ietf.org/html/rfc4252.

[56] N. Borisov, I. Goldberg, and E. Brewer. "Off-the-record communication, or, why not to use PGP". In: *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*. ACM. 2004, pp. 77–84. URL: http://www.cypherpunks.ca/otr/otr-wpes.pdf.

[57] Deutsche Wikipedia. "Alice und Bob". In: *Wikipedia DE* (2011). URL: https://secure.wikimedia.org/wikipedia/de/w/index.php?title=Alice_und_Bob&oldid=90572525 (visited on 09/22/2011).

[58] Hubert Zimmermann. "OSI reference model – The ISO model of architecture for open systems interconnection". In: *Communications, IEEE Transactions on* 28.4 (1980), pp. 425–432.

[59] ISO. *Information Technology - Open Systems Interconnection - Basic Reference model: The Basic Model*. 1994. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip.

[60] Elektronik Kompendium. "Datenübertragung im Mobilfunk". In: *Kommunikationsvtechnik-Fibel* (2010). URL: http://www.elektronik-kompendium.de/sites/kom/0910141.htm.

[61] Mile Davidovic. *RIL and datacall (CSD)*. June 2009. URL: https://groups.google.com/group/android-developers/browse_thread/thread/99d1f701ce0cfbd6.

[62] Savenger. *CSD Datacall support*. Oct. 2010. URL: https://code.google.com/p/android/issues/detail?id=11726.

[63] E-Plus. *E-Plus Netzabdeckung 2011*. 2011. URL: http://eis03sn1.eplus-online.de/geo/portal/umts (visited on 09/14/2011).

[64] O2. *O2 Netzabdeckung 2011*. 2011. URL: https://www.o2online.de/nw/support/mobilfunk/netz/index.html (visited on 09/14/2011).

[65] T-Mobile. *T-Mobile Netzabdeckung 2011*. 2011. URL: http://www.t-mobile.de/funkversorgung/inland/ (visited on 09/14/2011).

[66] Vodafone. *Vodafone Netzabdeckung 2011*. 2011. URL: https://www.vodafone.de/privat/hilfe-support/netzabdeckung.html? (visited on 09/14/2011).

[67] W. Xiao et al. "Voice Over IP (VoIP) Over Cellular: HRPD-A and HSDPA/HSUPA". In: *IEEE VEHICULAR TECHNOLOGY CONFERENCE*. Vol. 62. 4. IEEE; 1999. 2005, p. 2785. URL: https://www.itu.int/ITU-D/imt-2000/DocumentsIMT2000/TechnicalArticles2008/VoIP_EVDO_HSPA.pdf.

[68] ITU, M. Lazhar, and M. Hakim. *LTE Overview – Design Targets and Multiple Access Technologies and Multiple Access Technologies*. 2010. URL: http://www.itu.int/ITU-D/arb/COE/2010/4G/Documents/Doc4-LTE%20Workshop_TUN_Session3_LTE%20Overview.pdf.

[69] Nokia Siemens Networks. *LTE performance for initial deployments*. 2011. URL: http://www.nokiasiemensnetworks.com/NR/rdonlyres/4B75329B-3750-4BBB-8320-7113613AAB64/0/LTE_measurement_A4_1302.pdf (visited on 09/27/2011).

[70] hsdpa-umts-verfuegbarkeit.de. *Halbjahresbericht 2010 zur UMTS Verfügbarkeit – 70Netzabdeckung erreicht*. 2010. URL: http://www.hsdpa-umts-verfuegbarkeit.de/blog/2010/07/07/halbjahresbericht-2010-zur-umts-verfuegbarkeit-70-netzabdeckung-erreicht/ (visited on 09/14/2011).

[71] N. Katugampala, A. Kondoz, and S. Villette. "Secure voice over GSM and other low bit rate systems". In: *COLLOQUIUM DIGEST-IEE*. IEE; 1999. 2003, pp. 3–3.

[72] N. Katugampala et al. "Real time data transmission over GSM voice channel for secure voice and data applications". In: *Secure Mobile Communications Forum: Exploring the Technical Challenges in Secure GSM and WLAN, 2004. The 2nd IEE (Ref. No. 2004/10660)*. IET. 2004, pp. 7–1. ISBN: 0863414613.

[73] N. Katugampala et al. "Real time end to end secure voice communications over gsm voice channel". In: *13th European Signal Processing Conference*. 2005.

[74] NIST. *Advanced Encryption Standard (AES) (FIPS PUB 197) - Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. 2001. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (visited on 09/23/2011).

[75] Google, Android, and OpenHandsetAlliance. *Android Supported Media Formats*. 2011. URL: http://developer.android.com/guide/appendix/media-formats.html (visited on 09/16/2011).

[76] S.W. Shin et al. "Designing a unified speech/audio codec by adopting a single channel harmonic source separation module". In: *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*. IEEE. 2008, pp. 185–188. URL: http://202.114.89.42/resource/pdf/1495.pdf.

[77] Jürgen Schnitzler. "A 13.0 kbit/s wideband speech codec based on SB--ACELP". In: *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*. Vol. 1. IEEE. 1998, pp. 157–160. URL: http://www.ind.rwth-aachen.de/fileadmin/publications/schnitzler98.pdf.

[78] J. Makinen et al. "AMR-WB+: a new audio coding standard for 3rd generation mobile audio services". In: *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*. Vol. 2. IEEE. 2005, pp. ii–1109. ISBN: 0780388747. URL: http://eipa-acc.ericsson.com/res/thecompany/docs/journal_conference_papers/multimedia_technologies/amr-wb_a_new_audio_coding_standard_for_3rd_generation_mobile_audio_services.pdf.

[79] C. Birkehammar et al. “New high-quality voice service for mobile networks”. In: *Ericsson Review* (2006).

[80] *Wideband Coding of Speech and Audio Signals using Bandwidth Extension Techniques*. Institute of Communication Systems and Data Processing. Aachen, 2006. URL: http://www.ind.rwth-aachen.de/fileadmin/publications/esch06.pdf.

[81] Gerd Siegmund. *Next Generation Networks*. Hüthig, 2002. ISBN: 3778539639.

[82] J. Postel and ISI. *User Datagram Protocol*. Tech. rep. IETF RFC 768 September, 1980. URL: https://tools.ietf.org/html/rfc768.

[83] M. Allman et al. *TCP Congestion Control*. Tech. rep. IETF RFC 5681 September, 2009. URL: https://tools.ietf.org/html/rfc5681.

[84] E. Kohler et al. *Datagram Congestion Control Protocol (DCCP)*. Tech. rep. IETF RFC 4340 March, 2006. URL: http://tools.ietf.org/html/rfc4340.

[85] Henning Schulzrinne et al. *RTP: A Transport Protocol for Real-Time Applications*. Tech. rep. IETF RFC 3550 July, 2003. URL: https://tools.ietf.org/html/rfc3550.

[86] Henning Schulzrinne et al. *RTP Profile for Audio and Video Conferences with Minimal Control*. Tech. rep. IETF RFC 3551 July, 2003. URL: https://tools.ietf.org/html/rfc3551.

[87] M. Handley et al. *SDP: Session Description Protocol*. Tech. rep. IETF RFC 4566, July, 2006. URL: http://tools.ietf.org/html/rfc4566.

[88] Dorgham Sisalem et al. *SIP Security*. 1st ed. John Wiley & Sons Inc, 2009. ISBN: 0470516364.

[89] J. Sjoberg et al. *RTP Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs*. Tech. rep. IETF RFC 4867 April, 2007. URL: https://tools.ietf.org/rfcmarkup/4867.

[90] M. Baugher et al. *The Secure Real-time Transport Protocol (SRTP)*. Tech. rep. IETF RFC 3711 March, 2004. URL: https://tools.ietf.org/html/rfc3711.

[91] T. Adomkus and E. Kalvaitis. “Investigation of VoIP Quality of Service using SRTP Protocol”. In: *Electronics and Electrical Engineering.–Kaunas: Technologija* (2008), p. 84. URL: http://www.ktu.lt/lt/mokslas/zurnalai/elektros_z/z84/19_ISSN_1392-1215_Investigation%20of%20VoIP%20quality%20of%20service%20using%20SRTP%20protocol.pdf.

[92] Henning Schulzrinne et al. *Real Time Streaming Protocol (RTSP)*. Tech. rep. IETF RFC 2326, April, 1998. URL: http://tools.ietf.org/html/rfc2326.

[93] Inc. Microsoft. *Real-Time Messaging Protocol (RTMP) specification*. Tech. rep. Microsoft, 2011. URL: http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/%5BMS-MMSP%5D.pdf.

[94] S. Kent, K. Seo, and BBN Technologies. *Security Architecture for the Internet Protocol*. Tech. rep. RFC 4301, December, 2005. URL: https://tools.ietf.org/html/rfc4301.

[95] Adobe. *Real-Time Messaging Protocol (RTMP) specification*. Tech. rep. Adobe, 2009. URL: http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf.

[96] E. Rescorla et al. *Datagram Transport Layer Security*. Tech. rep. RFC 4347, April, 2006. URL: https://tools.ietf.org/html/rfc4347.

[97] T. Dierks et al. *The Transport Layer Security (TLS) Protocol Version 1.2*. Tech. rep. RFC 5246, August, 2008. URL: https://tools.ietf.org/html/rfc5246.

[98] *SIP: Session Initiation Protocol*. Tech. rep. IETF RFC 3261 June, 2002. URL: https://tools.ietf.org/html/rfc3261.

[99] IBM. *SIP Invite Signalling*. 2011. URL: http://www.ibm.com/developerworks/websphere/techjournal/0606_burckart/0606_burckart_images/figure2.gif (visited on 09/27/2011).

[100] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. Tech. rep. IETF RFC 2616 June, 1999. URL: https://tools.ietf.org/html/rfc2616.

[101] J. Franks et al. *HTTP Authentication: Basic and Digest Access Authentication*. Tech. rep. IETF RFC 2617 June, 1999. URL: https://tools.ietf.org/html/rfc2617.

[102] E.C. Cha, H.K. Choi, and S.J. Cho. "Evaluation of security protocols for the Session Initiation Protocol". In: *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*. IEEE. 2007, pp. 611–616. URL: http://hit.skku.edu/~hkchoi/pubs/sip_icccn2007.pdf.

[103] C. Kaufman et al. *Internet Key Exchange Protocol Version 2 (IKEv2)*. Tech. rep. RFC 5996, September, 2010. URL: hhttps://tools.ietf.org/html/rfc5996.

[104] P. Gupta and V. Shmatikov. "Security analysis of voice-over-IP protocols". In: (2007).

[105] J. Peterson et al. *Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)*. Tech. rep. IETF RFC 4474 August, 2006. URL: https://tools.ietf.org/html/rfc4474.

[106] J. Elwell and Siemens Enterprise Communications Ltd. *Connected Identity in the Session Initiation Protocol (SIP)*. Tech. rep. IETF RFC 4916 June, 2007. URL: https://tools.ietf.org/html/rfc4916.

[107] ITU Recommendation. *H.225 - Call signalling protocols and media stream packetization for packet-based multimedia communication systems*. 2009. URL: http://www.itu.int/rec/T-REC-H.225.0-200912-I/en.

[108] ITU Recommendation. *H.245 - Control protocol for multimedia communication*. May 2011. URL: http://www.itu.int/rec/T-REC-H.245/en.

[109] ITU Recommendation. *H.323 security: Framework for security in H-series (H.323 and other H.245-based) multimedia systems*. 2005.

[110] ITU Recommendation. *H.235.3 - H.323 security: Hybrid security profile*. 2005. URL: http://www.itu.int/rec/T-REC-H.235.3-200509-I/en.

[111] P. Saint-Andre and Cisco Systems. *Extensible Messaging and Presence Protocol (XMPP): Core*. Tech. rep. RFC 6120, March, 2011. URL: http://tools.ietf.org/html/rfc6120.

[112] S. Ludwig et al. "Xep-0166: Jingle". In: *XMPP Standards Foundation* (2009). URL: http://xmpp.org/extensions/xep-0166.html.

[113] Ed. A. Melnikov et al. *Simple Authentication and Security Layer (SASL)*. Tech. rep. RFC 4422, June, 2006. URL: http://tools.ietf.org/html/rfc4422.

[114] P. Saint-Andre and Jabber Software Foundation. *End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol*. Tech. rep. RFC 3923, October, 2004. URL: http://tools.ietf.org/html/rfc3923.

[115] D. Meyer et al. *XTLS: End-to-End Encryption for the Extensible Messaging and Presence Protocol (XMPP) Using Transport Layer Security (TLS)*. Tech. rep. draft-meyer-xmpp-e2e-encryption-02, June, 2009. URL: http://tools.ietf.org/html/draft-meyer-xmpp-e2e-encryption-02.

[116] N. Greene et al. *Media Gateway Control Protocol Architecture and Requirements*. Tech. rep. RFC 2805, April, 2000. URL: http://tools.ietf.org/html/rfc2805.

[117] ITU Recommendation. *H.248.1: Gateway control protocol: Version 3*. 2005. URL: http://www.itu.int/rec/T-REC-H.248.1/en.

[118] F. Andreasen, B. Foster, and Cisco Systems. *Media Gateway Control Protocol (MGCP) Version 1.0*. Tech. rep. RFC 3435, January, 2003. URL: http://tools.ietf.org/html/rfc3435.

[119] M. Arango et al. *Media Gateway Control Protocol (MGCP) Version 1.0*. Tech. rep. RFC 2705, October, 1999. URL: http://tools.ietf.org/html/rfc2705.

[120] D. Endler and M. Collier. *Hacking exposed VoIP: voice over IP security secrets & solutions*. McGraw-Hill Osborne Media, 2007. ISBN: 0072263644.

[121] H. Tschofenig et al. *Requirements and Analysis of Media Security Management Protocols, RFC 5479*. Tech. rep. IETF RFC 5479 April, 2009. URL: http://tools.ietf.org/rfcmarkup?doc=5479.

[122] M. Baugher et al. *The Use of Timed Efficient Stream Loss-Tolerant Authentication (TESLA) in the Secure Real-time Transport Protocol (SRTP)*. Tech. rep. IETF RFC 4383, February, 2006. URL: http://tools.ietf.org/rfcmarkup/4383.

[123] P. Zimmermann et al. *ZRTP: Media Path Key Agreement for Unicast Secure RTP*. Tech. rep. IETF RFC 6189 April, 2010. URL: http://tools.ietf.org/html/rfc6189.

[124] zrtp.org. *ZRTP Protocol Handshake*. 2011. URL: http://www.zrtp.org/_/rsrc/1290957736061/zrtp-protocol/zrtp-exchange.png (visited on 09/28/2011).

[125] P. Gutmann. "Plug-and-play PKI: A PKI your mother can use". In: *Proceedings of the 12th conference on USENIX Security Symposium-Volume 12*. USENIX Association. 2003, pp. 4–4.

[126] F. Stajano. "The resurrecting duckling — what next?" In: *Security Protocols*. Springer. 2001, pp. 204–214.

[127] John Floroiu and Dorgham Sisalem. "A comparative analysis of the security aspects of the multimedia key exchange protocols". In: *Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications*. ACM. 2009, p. 2. URL: http://iptcomm.org/iptcomm2009papers/1569195003.pdf.

[128] M.T. Ashraf. "ZRTP: A New Approach to Secure VoIP calls." In: *Canadian Journal on Network and Information Security* 1.6 (2010).

[129] J. Arkko et al. *MIKEY: Multimedia Internet KEYing, RFC 3830*. Tech. rep. IETF RFC 3830 August, 2004. URL: http://tools.ietf.org/html//rfc3830.

[130] D. McGrew et al. *Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)*. Tech. rep. IETF RFC 5764 May, 2010. URL: https://tools.ietf.org/html/rfc5764.

[131] D. Wing and Cisco. *DTLS-SRTP Key Transport (KTR) draft-wing-avt-dtls-srtp-key-transport-03*. Tech. rep. RFC draft-wing-avt-dtls-srtp-key-transport-03, March, 2009. URL: https://tools.ietf.org/html/draft-wing-avt-dtls-srtp-key-transport-03.

[132] M. Baugher et al. *GDOI Key Establishment for the SRTP Data Security Protocol draft-ietf-msec-gdoi-srtp-01.txt*. Tech. rep. raft-ietf-msec-gdoi-srtp-01, December, 2007. URL: hhttps://tools.ietf.org/html/draft-ietf-msec-gdoi-srtp-01.

[133] D. Maughan et al. *Internet Security Association and Key Management Protocol (ISAKMP)*. Tech. rep. RFC 2408, November, 1998. URL: https://tools.ietf.org/html/rfc2408.

[134] D. Harkins, D. Carrel, and Cisco Systems. *IThe Internet Key Exchange (IKE)*. Tech. rep. RFC 2409, November, 1998. URL: https://tools.ietf.org/html/rfc2409.

[135] F. Andreasen et al. *Session Description Protocol (SDP) Security Descriptions for Media Streams*. Tech. rep. IETF RFC 4568 , July, 2006. URL: https://tools.ietf.org/html/rfc4568.

[136] certgate. *ccertgate SmartCard microSD*. 2011. URL: http://www.certgate.com/index.php?id=71 (visited on 09/23/2011).

[137] certgate. Inc. *Certgate*. 2011. URL: http://www.certgate.com (visited on 09/23/2011).

[138] certgate. *certgate Security for Android*. 2011. URL: http://www.certgate.com/index.php?id=190 (visited on 09/23/2011).

[139] H. Handschuh and P. Paillier. "Smart Card Crypto-Coprocessors for Public-Key Cryptography". In: (2000). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.7742&rep=rep1&type=pdf.

[140] NXP. *P5CC072 - NXP*. 2011. URL: http://www.nxp.com/acrobat_download2/other/identification/096210.pdf (visited on 09/23/2011).

[141] NXP Philips. *NXP*. 2011. URL: http://www.nxp.com/.

[142] RSA Laboratories. *RSA Algorithm - PKCS #1 v2.1: RSA Cryptography Standard*. 2002. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf (visited on 09/23/2011).

[143] Certicom Research. *SEC 1: Elliptic Curve Cryptography*. 2000. URL: http://www.secg.org/download/aid-385/sec1_final.pdf (visited on 09/23/2011).

[144] Bruce Schneier. *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C*. Addison-Wesley, 1996.

[145] BSI. *Anwendungshinweise und Interpretationen zum Schema (AIS)*. 2004. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Reporte02/0227a_pdf.pdf?__blob=publicationFile (visited on 09/23/2011).

[146] BSI. *BSI-DSZ-CC-0227-2004*. 2004. (Visited on 09/23/2011).

[147] George Orwell. *1984*. 17. Auflage. Ullstein Taschenbuch, 1994. ISBN: 978-3548234106.

[148] English Wikipedia. "Memory hole". In: *Wikipedia EN* (2011). URL: https://secure.wikimedia.org/wikipedia/en/w/index.php?title=Memory_hole&oldid=445726061.

[149] developer.android.com and Inc. Google. *Android API Levels*. 2011. URL: http://developer.android.com/guide/appendix/api-levels.html (visited on 09/22/2011).

[150] mobicents, Inc. *mobicents*. 2011. URL: http://www.mobicents.org/ (visited on 09/22/2011).

[151] Sheng Liang. *The Java Native Interface: Programmer's Guide and Specification*. Addison-Wesley Professional, 1999. URL: http://java.sun.com/docs/books/jni/download/jni.pdf.

[152] debian.org. *Debian Wheezy*. 2011. URL: http://www.debian.org/releases/testing/ (visited on 09/22/2011).

[153] android.org. *Android ADT Plugin*. 2011. URL: http://developer.android.com/sdk/eclipse-adt.html (visited on 09/22/2011).

[154] android.org. *Android SDK Tools*. 2011. URL: http://developer.android.com/sdk/tools-notes.html (visited on 09/22/2011).

[155] android.org. *Android NDK*. 2011. URL: http://developer.android.com/sdk/ndk/index.html (visited on 09/22/2011).

[156] git-scm.com/. *git - the fast version control system*. 2011. URL: http://git-scm.com/ (visited on 09/22/2011).

[157] Google Inc. *Photograph of the Nexus S Developse Smartphone*. 2011. URL: http://www.google.com/nexus/images/features/3-phones.jpg (visited on 09/21/2011).

[158] C. Maia, L. Nogueira, and L.M. Pinho. "Evaluating Android OS for Embedded Real-Time Systems". In: *Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*. 2010.

[159] kerneltrap.org and. *Linux: The Completely Fair Scheduler*. 2007. URL: http://kerneltrap.org/node/8059 (visited on 09/22/2011).

[160] developer.android.com. *Android 2.3 Highlights*. 2011. URL: http://developer.android.com/sdk/android-2.3-highlights.html (visited on 09/22/2011).

[161] market.android.com. *Shark for root*. 2011. URL: https://market.android.com/details?id=lv.n3o.shark (visited on 09/22/2011).

[162] tcpdump.org. *tcpdump, a powerful command-line packet analyzer; and libpcap, a portable C/C++ library for network traffic capture*. 2011. URL: http://www.tcpdump.org/ (visited on 09/21/2011).

[163] Wikipedia (EN). *Rooting (Android OS)*. 2011. URL: https://secure.wikimedia.org/wikipedia/en/w/index.php?title=Rooting_%28Android_OS%29&oldid=448432547 (visited on 09/04/2011).

[164] gnu.org. *Automake is a tool for automatically generating 'Makefile.in' files compliant with the GNU Coding Standards*. 2011. URL: https://www.gnu.org/software/automake/ (visited on 09/21/2011).

[165] J. Rosenberg et al. *Session Traversal Utilities for NAT (STUN)*. Tech. rep. RFC 5389, Octobre, 2008. URL: https://tools.ietf.org/html/rfc5389.

[166] asterisk.org and Digium. *Asterisk powers IP PBX systems, VoIP gateways, conference servers and more*. 2011. URL: https://www.asterisk.org/ (visited on 09/21/2011).

[167] freeswitch.org. *Freeswitch*. 2011. URL: http://www.freeswitch.org/ (visited on 09/22/2011).

[168] zrtp.org. *ZRTP Masquerading*. 2011. URL: http://www.zrtp.org/zrtp-masquerading (visited on 09/22/2011).

[169] Tor Project. *TOR - Anonymity Online Protect your privacy. Defend yourself against network surveillance and traffic analysis*. 2011. URL: https://www.torproject.org (visited on 09/26/2011).

[170] Freenet. *Freenet - The Free Network - Share, Chat, Browse. Anonymously. On the Free Network - Papers*. 2011. URL: https://freenetproject.org/papers.html (visited on 09/26/2011).

[171] M. Munakata et al. *User-Agent-Driven Privacy Mechanism for SIP*. Tech. rep. RFC 5767, April, 2010. URL: https://tools.ietf.org/html/rfc5767.

[172] J. Peterson and Neustar. *A Privacy Mechanism for the Session Initiation Protocol (SIP)*. Tech. rep. RFC 3323, November, 2002. URL: https://tools.ietf.org/html/rfc3323.

[173] zrtp.org. *ZORG*. 2011. URL: http://www.zrtp.org (visited on 09/26/2011).

[174] University of Ottawa. *jUCMNav*. 2011. URL: http://lotos.csi.uottawa.ca/ucm/bin/view/ProjetSEG/WebHome (visited on 09/23/2011).

[175] Microsoft. *Visio*. 2011. URL: https://office.microsoft.com/en-us/visio/ (visited on 09/23/2011).

[176] sandrila.co.uk. *Sandrila SDL*. 2011. URL: http://www.sandrila.co.uk/visio-sdl/index.php (visited on 09/23/2011).

[177] Deutsche Wikipedia. "User Requirements Notation". In: *Wikipedia DE* (2011). URL: https://secure.wikimedia.org/wikipedia/de/w/index.php?title=User_Requirements_Notation&oldid=77483093 (visited on 09/23/2011).

[178] OMG and sysml.org. *OMG Systems Modeling Language Version 1.2*. 2010. URL: http://www.sysml.org/docs/specs/OMGSysML-v1.2-10-06-02.pdf (visited on 09/23/2011).

[179] OMG uml.org. *OMG Unified Modeling Language*. 2010. URL: http://www.omg.org/spec/UML/2.4/Infrastructure/Beta2/PDF (visited on 09/23/2011).

[180] T. Weilkiens. *Systems engineering with SysML/UML: modeling, analysis, design*. Morgan Kaufmann, 2007.

[181] papyrusuml.org. *Papyrus for SysML*. 2011. URL: http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp?P=128&L=EN (visited on 09/23/2011).

[182] softwarestencils.com. *Visio Stencil and Template for SysML 1.0*. 2011. URL: http://softwarestencils.com/sysml/ (visited on 09/23/2011).

[183] J. Buchmann. *Einführung in die Kryptographie*. 4. Springer, 2008.

[184] H. Lipmaa, P. Rogaway, and D. Wagner. *CTR-Mode Encryption*. 2000. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.1353&rep=rep1&type=pdf.

[185] Alan B. Johnston. *SIP: Understanding the Session Initiation Protocol*. 3. Artech House Publishers, 2009. ISBN: 1607839954.

[186] P. Thermos and A. Takanen. *Securing VoIP Networks: Threats, Vulnerabilities, and Countermeasures*. Addison-Wesley Professional, 2007. ISBN: 0321437349.

[187] Patrick Park. *Voice over IP security*. 2008.

[188] ITU Recommendation. *G. 107 - The E-model, a computational model for use in transmission planning*. 2009. URL: https://www.itu.int/rec/T-REC-G.107-200904-I.

[189] A.D. Elbayoumy and S.J. Shepherd. "Stream or block cipher for securing VoIP?" In: *International Journal of Network Security* 5.2 (2007), pp. 128–133.

[190] M. Petraschek et al. "Security and usability aspects of Man-in-the-Middle attacks on ZRTP". In: *Journal of Universal Computer Science* 14.5 (2008), pp. 673–692.

[191] Gerd Siegmund. *Technik der Netze 2 - Neue Ansätze: SIP in IMS und NGN*. 6. Hüthig, 2009.

[192] A. Menezes and P. Van Oorschot. *Handbook of Applied Cryptography*. CRC press, 1996.

[193] ITU Recommendation. "Method for the subjective assessment of intermediate quality level of coding systems". In: *ITU-R BS* (2003), pp. 1534–1.

[194] University of Southern California and John Postel (Ed.) *Internet Protocol - Darpa Internet Programm Protocol Specification*. Tech. rep. RFC 791, September, 1981. URL: https://tools.ietf.org/html/rfc791.

[195] R. Kamal, M.S. Siddiqui, and C.S. Hong. "Solving Interoperability Problem of SIP, H. 323 and Jingle with a Middleware". In: (2011).

[196] Danny Cohen. *Network Voice Protocol (NVP)*. Tech. rep. IETF RFC 741 January, 1976. URL: https://tools.ietf.org/html/rfc741.

[197] J. Fischl et al. *DTLS-SRTP - Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)*. Tech. rep. IETF RFC 5763 May, 2010. URL: http://tools.ietf.org/html/rfc5763.

[198] R. Singhai and A. Sahoo. "VoIP Security". In: *School of Information Technology, Indian Institute of Technology, Bombay* (2006).

[199] connect.de. *Netztest: Vodafone klarer Sieger, O2 verdrängt*. 2010. URL: http://www.connect.de/news/vodafone-klarer-sieger-o2-verdraengt-telekom-von-platz-2-1029920,287.html (visited on 09/14/2011).

[200] zfoneproject.com and Phil Zimmermann. *zfoneproject zrtp – Frequently Asked Questions*. 2011. URL: http://zfoneproject.com/faq.html (visited on 09/21/2011).

[201] Certicom Research. *SEC 1: Elliptic Curve Cryptography*. 2000. URL: http://www.secg.org/download/aid-385/sec1_final.pdf.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

3GPP 3rd Generation Partnership Project

AAC LC-LTP Advanced Audio Coding - Low Complexity/Long-Term Prediction

ACELP Algebraic Code Excited Linear Prediction

AES Advanced Encryption Standard

AES Advanced Encryption Standard

AMR Adaptive Multi-Rate (audio codec)

AMR-NB Adaptive Multi-Rate Narrow Band Codec

AMR-WB Adaptive Multi-Rate Wide Band Codec

BS Base Station

CA Certificate Authorities

CFS Completely Fair Scheduler

CSD Circuit Switched Data

CSRC Contributing Source Identifier

DDoS Distributed Denial of Service

DH Diffie-Hellman

DOA Data Origin Authentication

DoS Denial of Service

ECSD Enhanced CSD

EDGE Enhanced Data Rates for GSM Evolution

eGPRS Enhanced General Packet Radio Service

GDOI-SRTP Group Domain of Interpretation - SRTP

GPRS General Packet Radio Service

GRL Goal-Oriented Requirements Language

GSM Global System for Mobile Communications (früher Groupe Spécial Mobile)

HSCSD High Speed Circuit Switched Data

HSDPA High-Speed Downlink Packet Access

HSPA High Speed Packet Access

HSPA+ Evolved HSPA

HSUPA High-Speed Uplink Packet Access

IKE Internet Key Exchange

IMS IP Multimedia Subsystem

IP Internet Protocol

ISDN Integrated Services Digital Network

ITU International Telecommunication Union

JNI Java Native Interface

KMP Key Management Protocol

LTE 3GPP Long Term Evolution

LTE+ Long Term Evolution Advanced

MG Media Gateway

MGC Media Gateway Controller

MGCP Media Gateway Control Protocol

MIKEY Multimedia Internet KEYing

MitM Man-in-the-Middle (Attack)

MOS Mean Opinion Score

MUSHRA MUltiple Stimuli with Hidden Reference and Anchor

OS Operating System

PBX Private Branch Exchange

PKI Public Key Infrastructure

PSK Pre-Shared Keys

PSTN Public Switched Telephone Network

RAS Registration, Admission and Status

RCTP Real-Time Control Protocol

RTP Real-Time Transport Protocol

RTP-AVP RTP Profile for Audio and Video Conferences with Minimal Control

RTSP Real-Time Streaming Protocol

SAS Short Authentication String

SASL Simple Authentication and Security Layer

SCCP Skinny Client Control Protocol

SDES Security Descriptions for Media Streams

SDP Session Description Protocol

SDP Session Description Protocol

SIP-URI SIP Uniform Ressource Identifier

SIPS Secure Session Initiation Protocol transported over TLS (TCP)

SPIT SPAM over Internet Telephony

SRTCP Secure Real-Time Control Protocol

SRTP Secure Real-Time Protocol

SSRC Synchronization Source Identifier

STUN Session Traversal Utilities for NAT

SysML Systems Modeling Language

TESLA Timed Efficient Stream Loss-Tolerant Authentication

TLS Transport Layer Security

UA User Agent

UAC User Agent Client

UAS User Agent Server

UCM Use Case Maps

UE User Equipment

UE User Equipment

UMTS Universal Mobile Telecommunications System

URL Uniform Ressource Locator

URN User Requirements Notation

W-LAN Wireless LAN