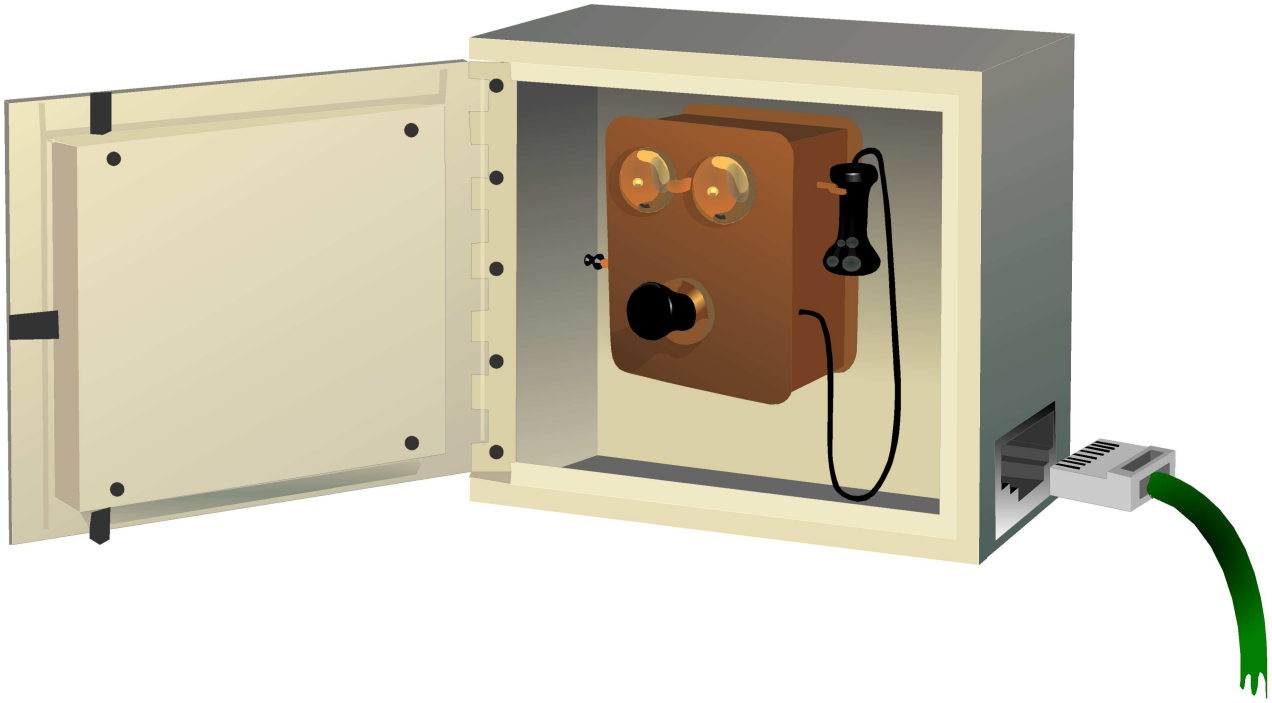
Zürcher
Hochschule
Winterthur

SIP Security



Praktische Diplomarbeit 2003 Sna 03/2

Fachhochschule:	Zürcher Hochschule Winterthur
Autoren:	Andreas Gisler, IT00
	Manfred Loretz, IT00
	Andreas Stricker, IT00
Dozent:	Prof. Dr. Andreas Steffen
Diplomarbeit:	DA Sna 03/2
Zeitraum:	Donnerstag, 4.09.2003 – Montag, 27.10.2003

Inhalt

1 Einführung

Inhalt

1.1 Zusammenfassung

Das Signalisierungs-Protokoll SIP (Session Initiation Protocol) gewinnt in der VoIP-Umgebung immer mehr an Bedeutung. Es stellt eine Alternative zum binär-codierten H.323 dar und weist diesem gegenüber einige Vorteile auf. Im stets wachsenden VoIP-Markt wird sicherlich bald der Wunsch nach einer gesicherten Kommunikation entstehen. Der SIP-Standard definiert verschiedene Methoden wie der Verbindungsaufbau und die Übermittlung der Daten gesichert werden kann. Bis jetzt existiert kein Client, der diese unterschiedlichen Möglichkeiten nutzt.

Im Rahmen dieser Diplomarbeit soll ein Client entwickelt oder erweitert werden, mit welchem der Verbindungsaufbau authentifiziert und verschlüsselt werden kann. Eine Möglichkeit besteht in der Verwendung des Verschlüsselungsstandards S/MIME, welcher bereits bei der sicheren E-Mail Kommunikation eingesetzt wird. Weiter soll abgeklärt werden, ob der symmetrische Session-Schlüssel mittels dem Session Description Protocol (SDP) ausgetauscht werden kann oder ob das MIKEY Protokoll verwendet werden muss. Die Audio- und Video-Nutzdaten können mit dem Secure Real-time Transport Protokoll (SRTP) gesichert übertragen werden.

Ein Ziel der Diplomarbeit bestand darin, die verschiedenen Software-Konzepte zu analysieren und zu entscheiden, welches realisiert werden kann. Als Basis für die Implementierung von SIP können verschiedene Open Source Stacks dienen. Auf der Seite der SIP-Clients existiert zur Zeit kein Open Source Projekt, welches sich zur Weiterentwicklung eignet. Das Endziel der Diplomarbeit ist eine sichere SIP Verbindung herzustellen und über den gesicherten Medienkanal Daten zu senden.

In der vorliegenden Arbeit wurde primär ein eigener Client entwickelt, der die Möglichkeiten der sicheren Übertragung in einer VoIP-Umgebung beherrscht. Benutzt wird das verschlüsselte Session Description Protokoll, welches in einer SIP Nachricht enthalten ist. Das Session Description Protokoll wird mit S/MIME geschützt. Der übermittelte Session-Schlüssel wird in der Folge für die Verschlüsselung des SRTP-Kanals gebraucht. Es ist nun möglich eine sichere End-zu-End-Verbindung herzustellen, welche die Multimedia-Daten (Audio und Video) verschlüsselt und authentifiziert.

Die Integration von Sprache und einer graphischen Benutzeroberfläche sind Bestandteil von zukünftigen Erweiterungen. Zur Zeit zeigt der Client die Möglichkeit zur sicheren Kommunikation auf und kann ohne Probleme erweitert oder auf andere Plattformen portiert werden. Damit ist der Client offen für zukünftige Entwicklungen.

Letztendlich werden VoIP-Anwendungen immer mehr Verbreitung finden und dadurch wird der Bedarf eines kryptographischen Schutzes für die Kommunikation zunehmen. Unsere Lösung erfüllt diesen Wunsch und schliesst somit eine bestehende Lücke in VoIP-Umgebungen. Nun hängt es von den Firmen und Benutzern ab, diese angebotenen Möglichkeiten auch zu nutzen.

Winterthur, 27.Oktober 2003

Andreas Gisler Manfred Loretz Andreas Stricker

1.2 Abstract

The Session Initiation Protocol (SIP) became more and more important over the last few years. SIP is based on accepted and applicable Internet standards in contrast to other Voice-over-IP (VoIP) protocols like H.323. With a growing VoIP market the wish for secure communication will be there sooner or later. These interests are especially on the possibility to protect the connection establishment and the transmitted data. The different methods are described in the SIP standard. At the moment, there is no client using all the possibilities for secure communication.

As a part of the diploma thesis a client should be written or extended which can encrypt and authenticate the connection establishment. The same cryptography standard as in secure e-mail communication called S/MIME had to be used. This work has to clarify if the session key can be exchanged via a protected Session Description Protocol (SDP). Another upcoming standard called MIKEY should also be evaluated. In VoIP there is one protocol for session establishment and another to carry the voice or video payload. The audio and video data can be transported with the Secure Real-time Transport Protocol (SRTP). This is also an upcoming standard and currently the only possibility to use.

One target was to analyze different software concepts and to decide which one can be implemented. There are several open source SIP stacks that can serve as a basis to implement and test the functionality of SIP. On the client side no actual open source project fulfils the requirements to enhance it. The final aim of this diploma thesis was to establish a secure SIP connection and transmit data through a secured payload channel.

Primarily, in this work an own client was developed using the possibilities for secure transmission in the VoIP environment. It uses an encrypted Session Description Protocol that is embedded in a SIP message to exchange a session key. The SDP itself is packed into S/MIME. The transmitted key is then used to encrypt the multimedia payload over a Secure Real-time Transport Protocol (SRTP). This means that it is possible to establish a secure connection with end-to-end encryption and authentication for multimedia data like voice and video.

The integration of voice and a graphical user interface is a target for future work. Currently the client is a proof of concept implementation. It is, nonetheless, extensible and portable and will not hinder further development.

In the end applications using VoIP will increase and as a consequence of this, the need to cryptographically protect the signalisation and data in SIP and RTP will become more important. Our solution provides a possibility to protect all these things and closes a gap in the world of VoIP. It depends on the companies and the users to implement a VoIP infrastructure and use a secure client.

2 Projektbeschreibung

Inhalt

2.1 Aufgabenstellung gemäss Ausschreibung

Kommunikation

Praktische Diplomarbeiten 2003 - Sna 03/2

SIP Security

Studierende:

Andreas Gisler, IT3a

Manfred Loretz, IT3a

Andreas Stricker, IT3a

Termine:

Ausgabe: Donnerstag, 4.09.2003 10:00 im E523

Abgabe: Montag, 27.10.2003 12:00

Beschreibung:

Als Alternative zum binär-codierten H.323 Voice-over-IP Protokoll hat das ASCII-basierte Session Inititiaton Protokoll (SIP) in den letzten Jahren sehr an Popularität gewonnen. Der SIP RFC 3261 beschreibt eine Anzahl von Methoden, wie der Verbindungsaufbau und die Übermittlung der Nutzdaten kryptografisch gesichert werden können. In der Praxis existieren aber noch fast keine sicheren Implementationen.

In der ausgeschriebenen Diplomarbeit soll der SIP Verbindungsaufbau auf der Basis von S/MIME authentisiert und verschlüsselt werden. Als Grundlage für die Entwicklung soll vom SIP 2.0 Stack reSIProcate ausgegangen werden.

Audio/Video Nutzdaten können mit dem Secure Real-time Transport Protocol (SRTP) gesichert werden. Im Rahmen der Diplomarbeit soll abgeklärt werden, ob die symmetrischen Sessionschlüssel via das S/MIME geschützte SDP Protokoll ausgetauscht werden können oder ob eventuell das MIKEY Protokoll verwendet werden muss.

Ziele:

Festlegen der zu realisierenden Leistungmerkmale in einem Pflichtenheft.

Erarbeiten einer oder mehrerer Konzeptvarianten. Konzeptentscheid.

Implementierung und Test der S/MIME und SRTP Funktionalität auf der Basis von reSIPprocate (SIP 2.0) oder Vovida (SIP 1.0).

Dokumentation der SIP Sicherheitslösung.

Kurzanleitung für das Aufsetzen einer sicheren SIP Verbindung.

Infrastruktur / Tools:

Raum: E523

Rechner: 3 PCs unter Windows 2000 / SuSE Linux

Hardware: 2 Headphones und/oder Telefonhörer

SW-Tools: Gnu C/C++ Compiler, Vovida, reSIPprocate, SRTP, OpenSSL 0.9.7b

Literatur / Links:

Daniel Kaufmann u. Andreas Stricker, ZHW Projektarbeit SS 2003

SIP Security

- Henning Schulzrinne, SIP Home Page

http://www1.cs.columbia.edu/sip/

- IETF RFC 3261 (Section 26 - Security Considerations)

SIP: Session Initiation Protocol

- IETF Internet Draft <draft-ietf-avt-srtp-09.txt>

The Secure Real-time Transport Protocol

- IETF Internet Draft <draft-ietf-msec-mikey-07.txt>

MIKEY: Multimedia Internet Keying

- Vovida SIP 1.0 Implementation

http://www.vovida.org

- reSIPprocate SIP 2.0 Implementation (partiell ?)

http://resiprocate.sf.net

- OpenSource Library implementing SRTP

http://srtp.sourceforge.net/spec.html

- OpenSSL Library implementing S/MIME

http://www.openssl.org/docs/apps/smime.html

Winterthur, 4. September 2003

Prof. Dr. Andreas Steffen

2.2 Analyse der Aufgabenstellung

Das Signalisierungs-Protokoll Session Initiaton Protokoll (SIP) gewinnt in der Welt von Voice-over-IP (VoIP) immer mehr an Bedeutung. In SIP ist das Session Description Protocol (SDP) eingebettet, welches zur Beschreibung und Aushandlung der nötigen Parameter für die Sitzung dient. Für die Übertragung der Nutzdaten wird das Real-time Transport Protocol (RTP) verwendet. Mit der immer weiter zunehmenden Verbreitung von VoIP wächst auch das Verlangen die Signalisierungs- und Nutzdaten in SIP und RTP kryptographisch zu sichern. Das Bedürfnis kryptographische Merkmale bei der Kommunikation anzuwenden, wird durch die Offenheit von IP-Netzwerken noch zusätzlich verstärkt.

Bei genauer Betrachtung sind vor allem folgende Punkte wichtig und werden als kryptographische Merkmale gefordert:

- **Authentifizierung:** Überprüfung der Integrität von den übermittelten Daten durch digitale Signaturen. Damit soll sichergestellt werden, dass die Daten auch vom angeblichen Absender stammen und auf dem Weg zum Empfänger nicht durch eine dritte Partei verändert wurden.
- **Verschlüsselung:** Umwandlung von Klartext oder Daten in eine nicht erkennbare Form mit Hilfe eines Verschlüsselungs-Algorithmus. Damit soll verhindert werden, dass Daten von Dritten eingesehen oder sogar verändert werden können.

Im SIP Standard [RFC3261] sind verschiedene Sicherheitsmechanismen spezifiziert um die oben aufgelisteten Merkmale anzuwenden und die Forderungen zu erfüllen. Bei den spezifizierten Sicherheitsmechanismen besitzt S/MIME am meisten Funktionen und bietet eine grosse Flexibilität. Allerdings wird durch die Public Key Verschlüsselung in S/MIME die Anforderung an Zertifikate und die nötigen Schlüssel gestellt. Zum jetzigen Zeitpunkt existiert nur ein Open Source SIP-Stack, der S/MIME implementiert und als solcher frei verfügbar ist. Zur Zeit gibt es keinen Client der diese Funktionalität nutzt und somit ist eine sichere Kommunikation mit SIP momentan unmöglich.

Mit dem Secure Real-time Transport Protokoll [SRTP] wird ein Profil für RTP [RFC1889] definiert, welches Authentifizierung, Verschlüsselung und Replay-Schutz anbietet. Das Dokument [SRTP] befindet sich noch im Status eines Drafts, aber eine Implementation mit einem Subset der in [SRTP] spezifizierten Funktionen ist bereits in einem Open Source Stack verfügbar.

Als Grundlage für die Verschlüsselung des SRTP-Kanals wird ein symmetrischer Schlüssel benötigt, welcher nur den Endpunkten einer Kommunikation bekannt sein darf. Somit darf bei der Übertragung des Schlüssels keine dritte Partei in den Besitz des Schlüssels kommen. Für diesen Zweck muss eine geeignete Methode gefunden werden, um den symmetrischen Schlüssel auszutauschen. Es bieten sich vor allem zwei Varianten an.

Der Schlüssel wird als zusätzliches Attribut im Session Description Protocol (SDP) mitgesendet werden, welches im SIP eingebettet ist. Da SIP und SDP normalerweise im Klartext gesendet wird, muss eine Verschlüsselung der SIP-Meldungen mittels S/MIME stattfinden.

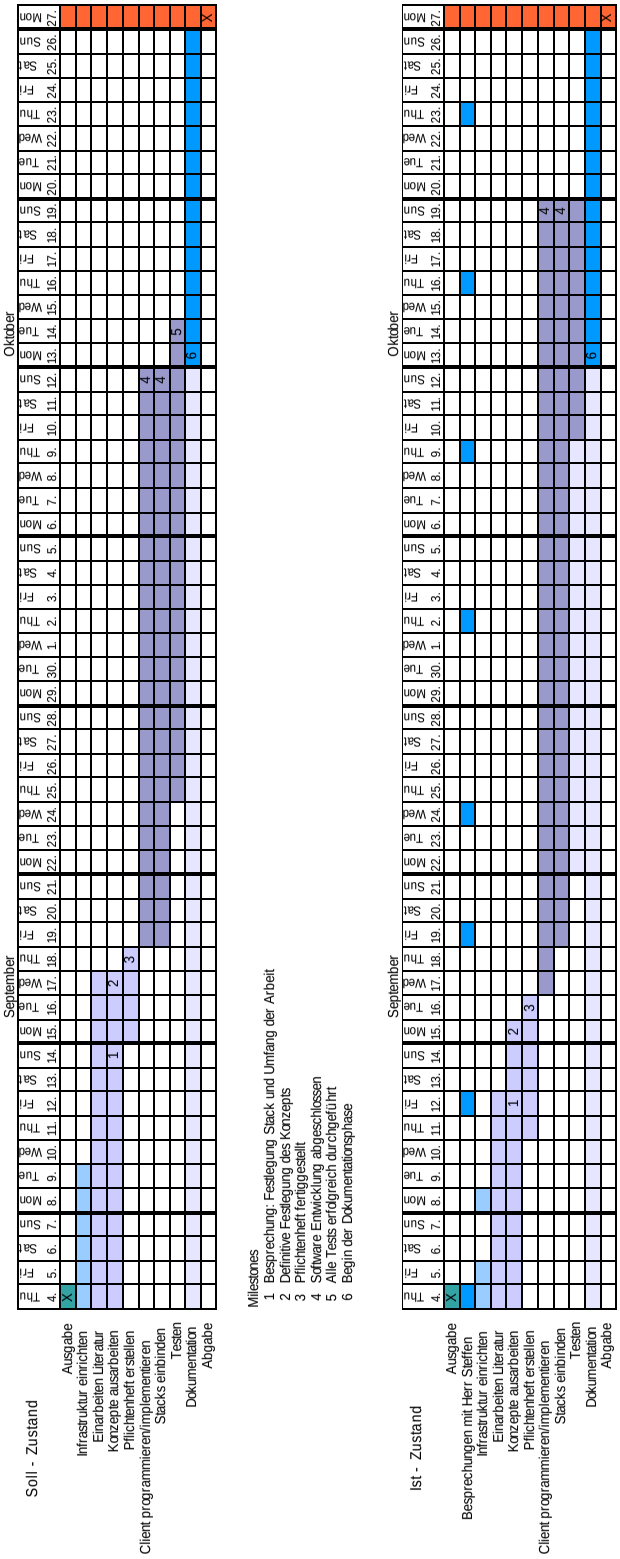
Als zweite Variante bietet sich das MIKEY an, welches für die sichere Aushandlung von Schlüsseln spezialisiert ist. Diese Variante soll zum Einsatz kommen, wenn eine sichere Übertragung über SDP nicht sichergestellt werden kann.

In der Arbeit soll gezeigt werden, wie die kryptographischen Anforderungen erfüllt und die nötigen Grundlagen bereitgestellt werden können.

3 Projektplan

Inhalt

3.1 Zeitplan


Soll - Zustand
September
Oktober
Ausgabe
Infrastruktur einrichten
Einarbeiten Literatur
Konzepte ausarbeiten
Pflichtenheft erstellen
Client programmieren/implementieren
Stacks einbinden
Testen
Dokumentation
Abgabe
Milestones
1 Besprechung: Festlegung Stack und Umfang der Arbeit
2 Definitive Festlegung des Konzepts
3 Pflichtenheft fertiggestellt
4 Software Entwicklung abgeschlossen
5 Alle Tess erfolgreich durchgeführt
6 Begin der Dokumentationsphase
Ist - Zustand
September
Oktober
Ausgabe
Besprechungen mit Herr Steffen
Infrastruktur einrichten
Einarbeiten Literatur
Konzepte ausarbeiten
Pflichtenheft erstellen
Client programmieren/implementieren
Stacks einbinden
Testen
Dokumentation
Abgabe

3.2 Vergleich Soll / Ist des Projektplanes

Der Zeitplan konnte von uns nicht immer genau eingehalten werden. Die Unterschiede entstanden vor allem durch unvorhergesehene Probleme, die mehr Zeit beanspruchten als wir dachten. Am Anfang waren wir dem Zeitplan sogar voraus. Dieser Zeitgewinn kam dadurch zu Stande, dass wir uns entschieden einen groben Überblick über die Literatur zu verschaffen und uns erst in die Tiefe einzulesen, wenn wir diese Literatur für die Implementierung wirklich benötigten. Durch diese Entscheidung war es nicht mehr notwendig, in den entsprechenden RFC's alle Kapitel gleich am Anfang komplett durchzulesen. Da wir relativ schnell viele Infos über die einzelnen Stacks zusammengetragen hatten, konnten wir uns früher Konzepte überlegen. In der Folge ging es an das genaue Ausarbeiten unserer Vorstellungen und mit der Erstellung eines Pflichtenheftes wurde begonnen. Unsere Erkenntnisse wurden erstmals am 12. September mit Herr Steffen besprochen, woraus in der Folge das Konzept und das Pflichtenheft verfeinert wurden. Durch weitere Entscheidungen und unter Berücksichtigung der Vorgaben von Herrn Steffen wurde dann das definitive Pflichtenheft erstellt.

In der Folge gab die Aufgabe den Client zu programmieren sehr viel zu tun und nahm schlussendlich auch deutlich mehr Zeit in Anspruch, als von uns vorgesehen war. Dieser grosse und unvorhergesehene Zeitaufwand wurde vor allem durch Probleme verursacht, deren Suche teilweise einen ganzen Tag beanspruchte. Auch die noch nicht getesteten Komponenten seitens des SIP-Stacks reSIProcate stellten uns teilweise vor seltsame Probleme, deren Lösungen nicht immer sofort ersichtlich waren.

Um uns die Programmierung zu vereinfachen, wurden zuerst einzelne Module geschrieben, wie beispielsweise für die SRTP-Verbindung. Zu jedem Modul erstellten wir ein Testprogramm, um damit die Funktionalität im kleinen Rahmen testen zu können. Aus diesem Grund ist auch das Testen im Projektplan in zwei verschiedene Phasen unterteilt. Der erste Teil bezeichnet die Tests der einzelnen Module, für welches es immer ein eigenes Testprogramm gibt. Später ist damit der ganze Client mit dem kompletten Funktionsumfang gemeint.

Auch die Dokumentation ist in zwei Phasen gegliedert, da wir anfänglich nur vereinzelt an dieser gearbeitet haben und erst am Schluss den ganzen Tag jeweils für das Dokumentieren unserer Arbeit verwendeten. In der Woche vom 13. Oktober bis zum 19. Oktober teilten wir die Arbeit auf, so dass sowohl weiter am Source Code als auch an der Dokumentation gearbeitet wurde. Dies war nötig, da der Client noch nicht so weit wie geplant fertig gestellt war und die Dokumentation trotz allem begonnen werden musste. Durch diese Entscheidung war es gewährleistet, dass sowohl der Client als auch die Dokumentation weiter vorangetrieben wurden.

3.3 Meilensteine

Die ersten 3 Meilensteine konnten etwas früher als geplant abgeschlossen werden. Dies konnte vor allem durch die schnelle Einarbeitung erreicht werden. Der vierte Meilenstein (Entwicklung abgeschlossen) ist leider nicht vollends erreicht. Dies ist vor allem durch den grossen Zeitaufwand und die Probleme mit dem SIP-Stack reSIProcate entstanden. Trotzdem ist zur Zeit der Abgabe ein lauffähiger Client vorhanden, der den Schlüsselaustausch per SDP bewerkstelligt und Nachrichten zwischen zwei Endpunkten austauscht. Das Testen ist nicht vollständig abgeschlossen und noch nicht genügend ausgeführt worden. Hier sind noch weitere intensive Tests von Nöten um allfällige Fehler erkennen zu können. Die Dokumentation konnte wie gewünscht geschrieben und abgeschlossen werden.

3.4 Stufen

Die Entwicklung soll in kleinen Stufen vorangetrieben werden. Dabei sind folgende Punkte zu erreichen:

Name	Voraussetzung	Beschreibung
SIP-Verbindung	Keine	Aufbau einer SIP-Verbindung
SDP-Passwort	SIP-Verbindung	Übertragung eines Passwortes in SDP
S/MIME-Verschlüsselung	SIP-Verbindung	Verschlüsselung des SDP-Protokolls
SRTP-Kanal	SDP-Passwort	Sicherer Datenkanal mit symmetrischer Verschlüsselung
S/MIME-Authentifizierung	SIP-Verbindung	Authentifizierung des Verbindungspartners
Voice-Übertragung	SRTP-Kanal	Übertragung von Sprache mit G.711

Table 1Aufteilung in zu erreichende Teilstufen

Die S/MIME Verschlüsselung und Authentifizierung sind noch nicht vollständig abgeschlossen. Hier besteht das Problem im Stack von reSIProcate, welcher in dieser Hinsicht beim analysieren fehlerhaft ist. Auch die Voice-Übertragung konnte bis zum Ende der Diplomarbeit noch nicht eingebunden werden. Zur Zeit werden Nachrichten zwischen den beiden Endpunkten übertragen, es findet also kein Auslesen der Soundkarte und anschliessendes Kodieren statt. Der Rest konnte während der Diplomarbeit vollends programmiert und in das Programm implementiert werden.

4 Pflichtenheft

Inhalt

4.1 Übersicht

Im Rahmen der Projektarbeit „SIP Security“ [PA_SIP] wurden verschiedene Lösungen vorgestellt, wie VoIP Anwendungen, basierend auf dem Session Initiation Protocol (SIP), gegen Angriffe und Manipulationen geschützt werden können.

Nun soll im Rahmen der Diplomarbeit „SIP Security“ eine realisierbare Möglichkeit implementiert werden, so dass als „Proof-of-Concept“ eine sichere Verbindung hergestellt werden kann.

4.2 Ist-Zustand

Verschiedene Mechanismen sind bereits in SIP [RFC3261] für eine verschlüsselte Kommunikation und eine gegenseitige Authentifizierung zwischen den Kommunikationspartnern spezifiziert. Leider gibt es nur wenige SIP Stacks die bereits alle Merkmale des [RFC3261] beinhalten. Zur Zeit implementiert nur der Stack vom Open Source Projekt reSIProcate die benötigten Mechanismen, welche für eine Authentifizierung und gesicherte Übertragung notwendig sind.

Für den Datenkanal besteht nach [SRTP] ein Protokoll für die Verschlüsselung und Authentifizierung der zu übertragenden Daten. Eine nicht aktuelle Implementation von [SRTP] existiert bereits als Open Source Stack.

Bis anhin existiert aber kein Open Source Client der sich die Funktionalität von reSIProcate und SRTP zu Nutze macht.

4.3 Ziele

Es ist das Ziel dieses Projektes einen SIP Client zu programmieren, welcher auf dem verfügbaren reSIProcate und dem SRTP Stack aufsetzt und die speziellen Merkmalen der Stacks verwendet. Damit soll bewiesen werden, dass eine sichere Verbindung mit S/MIME und SRTP realisiert werden kann. Es ist nicht das Ziel dieses Projektes einen SIP-Client zu erstellen, welcher bezüglich Auswahl von Sprachcodecs und graphischer Benutzeroberfläche mit bereits existierenden SIP-Clients konkurrenziert.

4.3.1 S/MIME-gesicherter Verbindungsaufbau

Für die Authentifizierung der beiden Verbindungsendpunkten und zur Verschlüsselung der SIP Nachrichten wird S/MIME verwendet. Zu diesem Zweck muss der Client nach [RFC3261] den Gebrauch von Endbenutzer-Zertifikaten unterstützen und die Zertifikate in einem Schlüsselring verwalten.

4.3.2 Übertragung eines Passwortes mit SDP

SDP sieht ein Feld für die Übertragung des Passwortes vor. Die Verwendung dieses Feldes macht nur Sinn, wenn das SDP-Protokoll verschlüsselt wird, da sonst das Passwort unverschlüsselt versendet wird. Die Verschlüsselung übernimmt S/MIME.

Das Passwort gilt nur für eine Verbindung, so dass bei jedem Verbindungsaufbau jeweils ein Passwort generiert wird. Eine frühere Projektarbeit (True Random Number Generator) hat gezeigt, dass die Qualität von `/dev/random` bei Linux dafür ausreicht und deshalb hier verwendet werden kann. Als Alternative soll auch betrachtet werden, welche Möglichkeiten durch OpenSSL bereit gestellt werden.

4.3.3 Aufbau eines sicheren Voice-Kanals über SRTP

Mit dem über SDP übertragenen Master-Key und Master-Salt wird nun eine SRTP-Verbindung zwischen den beiden Endpunkten hergestellt. Damit können die Endpunkte über eine verschlüsselte Leitung kommunizieren.

4.3.4 Optionale Sprachübertragung

Für den Beweis einer sicheren Kommunikation reicht der Austausch von Text-Nachrichten. Optional soll

ein Sprachkanal zur Verfügung gestellt werden, über welchen sich die Teilnehmer verständigen können. Da unsere Kommunikation innerhalb eines lokalen Netzwerkes stattfindet, stellt der Sprachcodec G.711 eine gute Wahl dar.

4.3.5 Optionales GUI

Für die Bedienung des SIP Clients soll optional eine graphische Benutzeroberfläche erstellt werden. Mit der graphischen Benutzeroberfläche soll die Bedienung des SIP Clients erleichtert werden.

4.4 Verhalten

4.4.1 Funktionen

Der Client ist in der Lage, SIP-Verbindungsanfragen anzunehmen und selber Verbindungen aufzubauen. Dabei muss der Client fähig sein, den Verbindungsaufbau auch mit S/MIME durchzuführen. Über SDP kann sowohl ein Passwort gesendet, wie auch empfangen werden. Mit diesem Passwort wird ein sicherer Datenkanal über SRTP hergestellt.

Zusammengefasst sind folgende untenstehende Funktionen notwendig.

4.4.1.1 Als Empfänger

- Annahme von SIP-Verbindungsanfragen
- S/MIME entschlüsseln
- Bei Verbindungen ohne S/MIME kann keine Sicherheit gewährleistet werden, auf diesen Umstand soll hingewiesen werden
- Aushandeln der Parameter des Datenkanals über SDP
- Passwort aus SDP extrahieren
    - Im Debug-Modus auch unverschlüsseltes SDP akzeptieren
    - Im Betriebs-Modus nur Passwörter in verschlüsselten SDP-Paketen akzeptieren
- SRTP-Verbindungen annehmen und entschlüsseln

4.4.1.2 Als Sender

- SIP Verbindungsaufbau
- Passwort generieren und im verschlüsselten SDP mitsenden
    - Im Debug-Modus auch im nicht verschlüsselten SDP mitsenden
- Aushandeln der Parameter des Datenkanals über SDP
- SDP mit S/MIME verschlüsseln
- SRTP-Verbindungen aufbauen

5 Grundlagen

Inhaltsverzeichnis

In diesem Kapitel werden die Grundlagen für unsere Diplomarbeit behandelt. Als eine kleine Einführung beginnen wir mit einem Abschnitt über die Verschlüsselung von Klartext in Schlüsseltext. Anschliessend folgt ein Abschnitt über die relevanten Protokolle und Standards.

5.1 Verschlüsselung

Bei der Verschlüsselung wird lesbarer Klartext in unlesbaren Schlüsseltext transformiert. Im normalen Gebrauch werden auch oft die aus dem Englischen übernommenen Begriffe Plaintext für Klartext und Ciphertext für Schlüsseltext verwendet. Der Prozess um den nicht lesbaren Schlüsseltext wieder in lesbaren Klartext zu verwandeln wird als Entschlüsseln bezeichnet.

Unter der Bezeichnung Klartext können alle Arten von Daten verstanden werden, welche durch Mensch oder Maschine lesbar sind. Es muss sich also nicht zwingend um Text handeln, sondern kann sich auch um Daten in binärer Form handeln. Schlüsseltext kennzeichnet das entstandene Produkt, welches durch das Verschlüsseln des Klartextes entstanden ist und für Mensch und Maschine nicht mehr interpretierbar ist.

Es existieren zwei Varianten um eine Verschlüsselung und Entschlüsselung durchzuführen. Bei der ersten Variante handelt es sich um die Secret Key Verschlüsselung oder auch symmetrische Verschlüsselung. Die zweite Variante wird Public Key Verschlüsselung oder auch asymmetrische Verschlüsselung genannt.

5.1.1 Secret Key Verschlüsselung

Die Secret Key Verschlüsselung ist auch unter dem Namen symmetrische Verschlüsselung bekannt. Es ist die Besonderheit dieses Verfahrens, dass der gleiche Schlüssel zum Verschlüsseln und Entschlüsseln verwendet wird, daher der Name symmetrische Verschlüsselung. Dieser Schlüssel muss absolut geheim gehalten werden und darf somit nur den Instanzen bekannt sein, die in der Lage sein sollen den Schlüsseltext zu entschlüsseln. Aus der zweiten Anforderung ist ersichtlich, wieso dieses Verfahren auch Secret Key Verschlüsselung genannt wird.

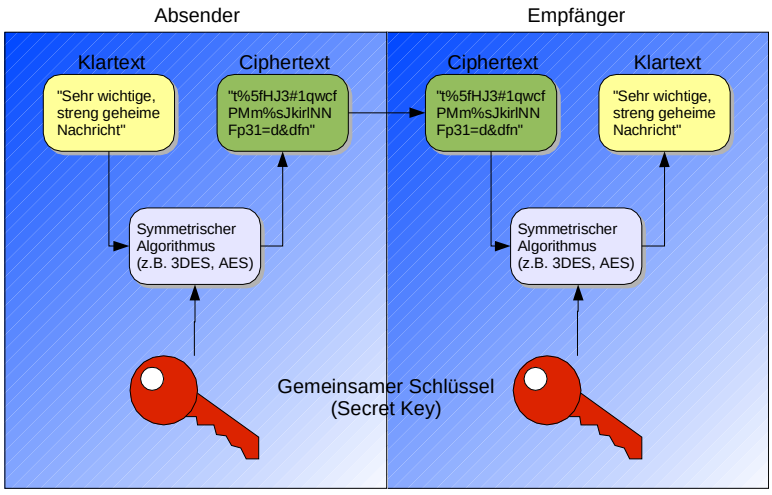


Abbildung 1 Symmetrische Verschlüsselung: Beide Parteien verwenden für die Ver- und Entschlüsselung denselben Schlüssel

Der Vorteil dieses Verfahrens liegt darin, das entsprechende Chiffrieralgorithmen sehr schnell sind, also nicht rechenintensiv und dadurch grosse Mengen an Klartext transformieren können.

Der Nachteil ist, dass der Schlüssel absolut geheim gehalten werden muss. Der Schlüssel darf nur den Instanzen bekannt sein, die in der Lage sein sollen, den chiffrierten Schlüsseltext zu lesen. Aus dieser Anforderung entsteht das Problem, dass der Schlüssel über einen sicheren Weg zu den Instanzen verteilt werden muss. Zur Lösung dieses Problems braucht es einen sicheren Kanal um den Schlüssel zu verteilen, was mit einem grossen Aufwand verbunden ist. Dieser Aufwand wächst exponentiell mit der Anzahl der vorhandenen Teilnehmern. Für jede sichere Verbindung zwischen den Instanzen wird ein geheimer Schlüssel benötigt, welcher nur den Instanzen bekannt sein darf, welche an der Kommunikation teilnehmen dürfen.

Es gibt eine grosse Auswahl von Algorithmen, die nach dem Prinzip der Private Key Verschlüsselung arbeiten. Zu den bekanntesten Algorithmen gehören DES, 3DES, RC2, RC5 und AES (Rijndael).

5.1.2 Public Key Verschlüsselung

Die zweite Variante trägt den Namen Public Key Verschlüsselung und ist auch unter dem Namen asymmetrische Verschlüsselung bekannt. Es ist das Merkmal dieses Verfahrens, dass nicht der gleiche Schlüssel zum Verschlüsseln und Entschlüsseln verwendet wird. Die Prozesse der Ver- und Entschlüsselung besitzen jeweils einen eigenen Schlüssel. Aus diesem Merkmal ist auch die Bezeichnung asymmetrische Verschlüsselung entstanden. Dabei darf der Schlüssel, um eine Nachricht für einen Empfänger zu chiffrieren, allen bekannt sein und muss sogar öffentlich bekannt gemacht werden. Dies erklärt den zweiten Namen dieses Verfahrens: Public Key Verschlüsselung. Der zum öffentlichen Schlüssel korrespondierende Schlüssel darf nur dem Empfänger der Nachricht bekannt sein, muss also absolut geheim gehalten werden und wird aus diesem Grund auch privater Schlüssel genannt.

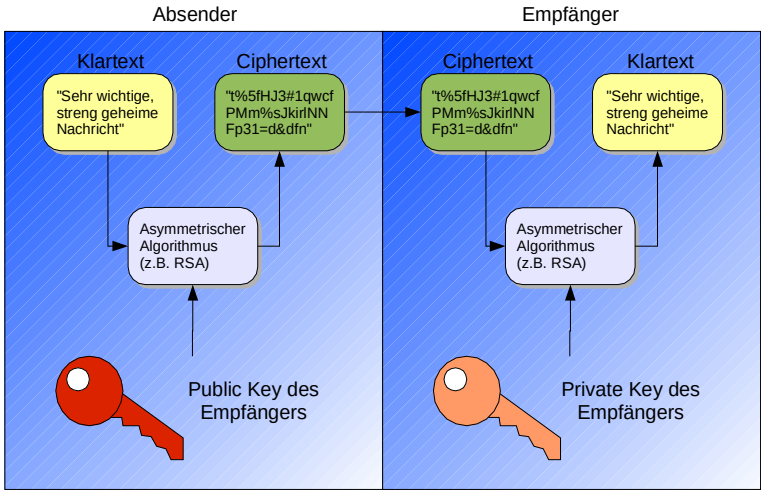


Abbildung 2 Asymmetrische Verschlüsselung: Beide Parteien verwenden unterschiedliche Schlüssel. Die Information wird mit dem Public Key des Empfängers verschlüsselt und mit dessen Private Key entschlüsselt.

Public Key Verfahren beruhen auf der Idee von mathematischen Ein-Weg-Funktionen, welche in Abhängigkeit vom öffentlichen Schlüssel des Empfängers, einen Klartext in einen Schlüsseltext verwandeln. Dieser Vorgang benötigt relativ wenig Rechenleistung, aber die Umkehrfunktion ist extrem aufwändig, so dass ein Angreifer nicht in der Lage ist, den Klartext aus dem übertragenen Schlüsseltext in einem vernünftigen Zeitrahmen zu berechnen. Der öffentliche Schlüssel kann nur zum Verschlüsseln verwendet werden und bietet beim Versuch den Schlüsseltext wieder zu Entschlüsseln keinen Nutzen. Erst mit dem privaten Schlüssel des Empfängers kann der Schlüsseltext wieder in Klartext verwandelt werden.

Mit Hilfe der Public Key Verschlüsselung wird eine bessere Skalierbarkeit in grossen Netzwerken erreicht. Für jede Instanz wird nur ein Schlüsselpaar benötigt, bestehend aus einem öffentlichen Schlüssel für alle Sender und einem privaten Schlüssel für den Empfänger. Darin liegt auch der grosse Vorteil dieses Verfahrens gegenüber der Private Key Verschlüsselung.

Der Nachteil der Public Key Verschlüsselung liegt darin, dass sie viel rechenintensiver ist als die Secret Key Verschlüsselung. Dadurch wird dieses Verfahren vor allem für kleine Datenmengen gebraucht, um zum Beispiel zu Beginn einer Übertragung einen gemeinsamen, symmetrischen Schlüssel auszutauschen.

Ein weiterer Nachteil der Public Key Verschlüsselung besteht darin, dass eine Verbindung zwischen einem Empfänger und seinem öffentlichen Schlüssel hergestellt werden muss. Für diesen Zweck gibt es Zertifikate, welche sicherstellen das ein bestimmter, öffentlicher Schlüssel auch zur richtigen Instanz gehört. Für die Erstellung, Verteilung und Verwaltung von Zertifikaten wird eine Public Key Infrastruktur (PKI) benötigt, eine solche PKI muss zuerst mal aufgebaut werden.

Im Vergleich zur Private Key Verschlüsselung, gibt es weniger Algorithmen, die nach dem Prinzip der Public Key Verschlüsselung arbeiten. Zu erwähnen sind hier die Algorithmen RSA und Diffie-Hellman.

5.1.3 Kombiniertes Verfahren

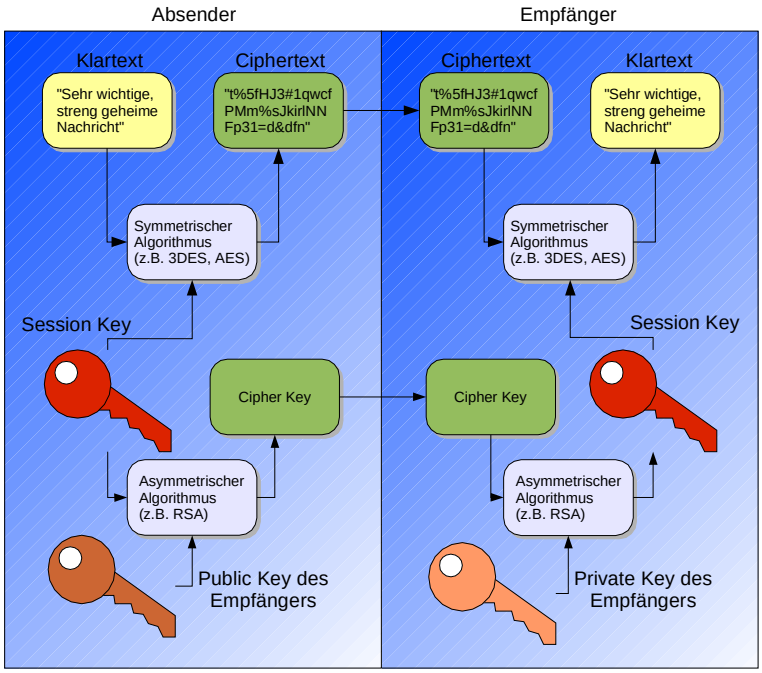


Abbildung 3 Bei der Kombination beider Verfahren wird lediglich der Session Key asymmetrisch chiffriert. Die eigentliche Information wird mit dem Session Key symmetrisch verschlüsselt.

Um die Vor- und Nachteile der Secret Key und Public Key Verschlüsselung auszugleichen, liegt es auf der Hand, die beiden Verfahren miteinander zu kombinieren. So werden die eigentlichen Daten mit einem symmetrischen Algorithmus und einem zufällig gewählten Schlüssel, dem so genannten Session Key, chiffriert. Mit dem asymmetrischen Algorithmus wird dann lediglich der Session Key verschlüsselt.

5.2 Message Digest und digitale Signaturen

Im vorausgehenden Abschnitt haben wir die Verfahren vorgestellt, mit denen man die Daten vor den Augen dritter Personen verbergen kann. In diesem Abschnitt geht es nun darum, jemandem zu beweisen, dass der empfangene Inhalt auch wirklich vom angegebenen Sender ist. In der digitalen Welt wird dieses Verhalten mit digitalen Signaturen ermöglicht. Bevor allerdings auf die digitalen Signaturen eingegangen werden kann, müssen die Grundlagen über so genannte Message Digest bekannt sein. Aus diesem Grund folgt zuerst ein Abschnitt über Message Digest's und die damit verbundenen Hash-Funktionen.

5.2.1 Message Digest

Ein so genannter Message Digest hat eine feste Grösse und dient als eindeutiger Fingerabdruck für ein Dokument oder Daten beliebiger Grösse. Die Grösse des Message Digest bestimmt dabei die Menge von darstellbaren Fingerabdrücken. Die heute gebräuchlichen Message Digest besitzen eine Grösse zwischen 128 bis 256 Bits, mit denen ungefähr zwischen 10^{38} und 10^{77} unterschiedlichen Fingerabdrücken dargestellt werden können.

Für die Berechnung eines guten Message Digest werden spezielle Ein-Weg-Hash-Funktionen verwendet. Um einen Message Digest zu berechnen, sollte die verwendete Ein-Weg-Hash-Funktion folgende Eigenschaften besitzen:

- Die Berechnung des Message Digest soll schnell und effizient sein, womit das Hashing von Nachrichten in der Grösse von mehreren Gigabyte möglich wird.
- Da die Nachricht meistens viel grösser als sein Hashwert ist, produzieren mehrere unterschiedliche Dokumente einen gleichen Message Digest. Darum soll es praktisch unmöglich sein, für einen gegebenen Message Digest ein Dokument zu finden, welches bei Verwendung der Ein-Weg-Funktion den gegebenen Message Digest produziert. Aus diesem Grund wird eine gute Hash-Funktion auch als Ein-Weg-Hash-Funktion bezeichnet.

- Der Wert des Message Digest soll von jedem einzelnen Bit der gegebenen Nachricht abhängen. Wenn ein einzelnes Bit innerhalb der Nachricht seinen Wert ändert, ein Bit zur Nachricht hinzugefügt oder entfernt wird, dann sollen im Durchschnitt die Hälfte der Bits des Message Digest ihren Wert in pseudo-zufälliger Weise ändern. Somit soll die Ein-Weg-Hash-Funktion für zwei fast ähnliche Nachrichten, zwei vollkommen unterschiedliche Message Digest produzieren.

Durch das Merkmal der Pseudo-Zufälligkeit von Ein-Weg-Hash-Funktionen und der riesigen Anzahl von möglichen Fingerabdrücken, wird es eigentlich unmöglich, dass zwei verschiedene Nachrichten den gleichen Fingerabdruck produzieren. Für alle heutigen Anwendungen kann das Produkt einer guten Hash-Funktion sozusagen als eindeutiger Fingerabdruck für die gehashte Nachricht betrachtet werden.

Die heute bekanntesten und am meisten verbreiteten Hash-Funktionen sind der Message Digest #5 (MD5) und der Secure Hash Algorithm (SHA).

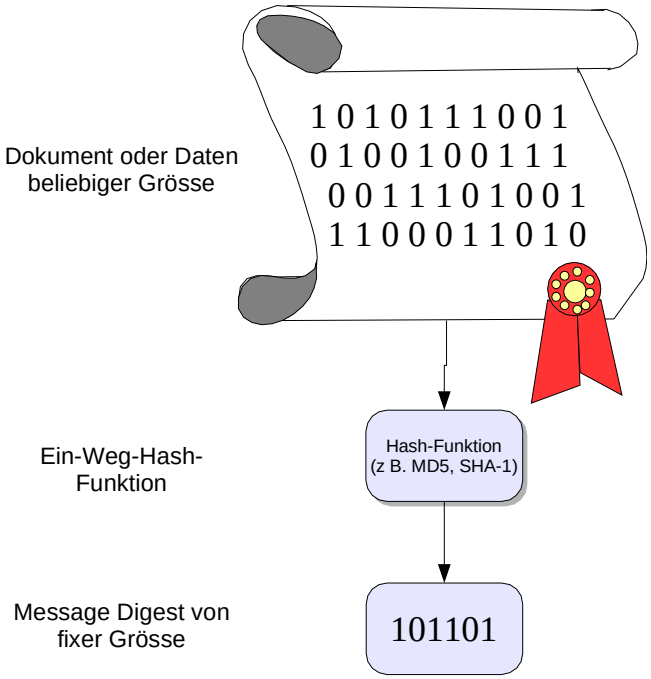


Abbildung 4 Message Digest: Mit einer Ein-Weg-Hash-Funktion für ein Dokument oder Daten beliebiger Grösse, wird ein Message Digest fixer Grösse erzeugt.

5.2.2 Digitale Signaturen

Wir haben bis jetzt die Verschlüsselung und das Hashen kennen gelernt. Um eine digitale Signatur zu erzeugen, greifen wir auf beide Merkmale zurück. Als erstes wird mit Hilfe einer Ein-Weg-Hash-Funktion ein Message Digest von einer Nachricht gebildet. Anschliessend wird der Message Digest mit dem privaten Schlüssel des Unterzeichners und mit einem asymmetrischem Algorithmus für die Verschlüsselung chiffriert. Das Produkt bildet die digitale Signatur des Unterzeichners über die signierte Nachricht.

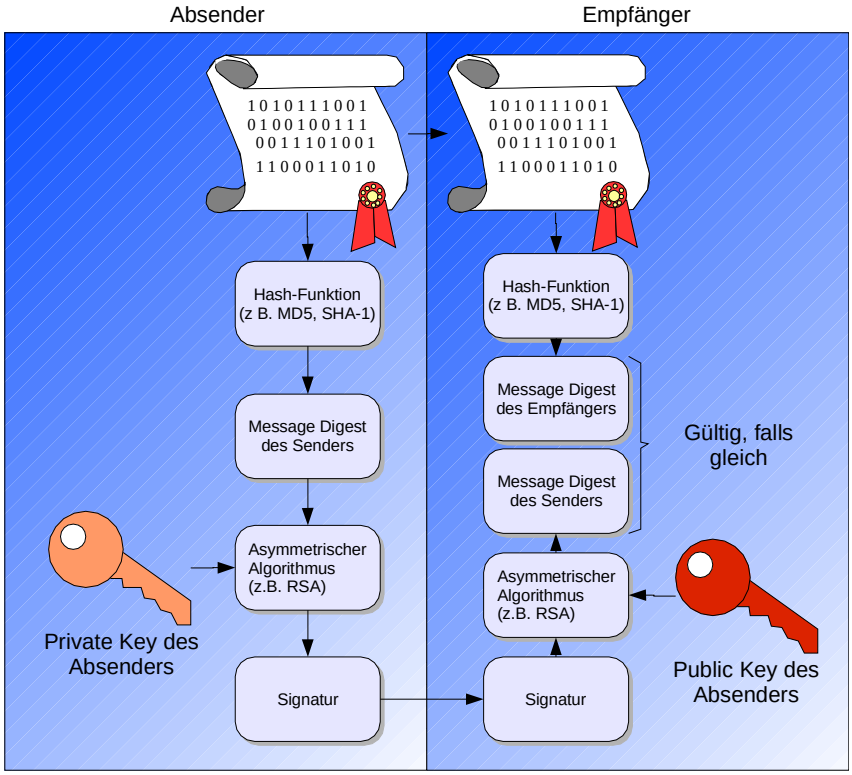


Abbildung 5 Digitale Signatur mit Public Key Verschlüsselung

Um die digitale Signatur zu überprüfen, muss der Empfänger den Message Digest über die empfangene Nachricht berechnen und die digitale Signatur mit dem öffentlichen Schlüssel des Autors entschlüsseln. Falls der berechnete Message Digest mit dem entschlüsselten Message Digest übereinstimmt, muss die Nachricht vom Unterzeichner stammen, da nur dieser den privaten Schlüssel besitzt.

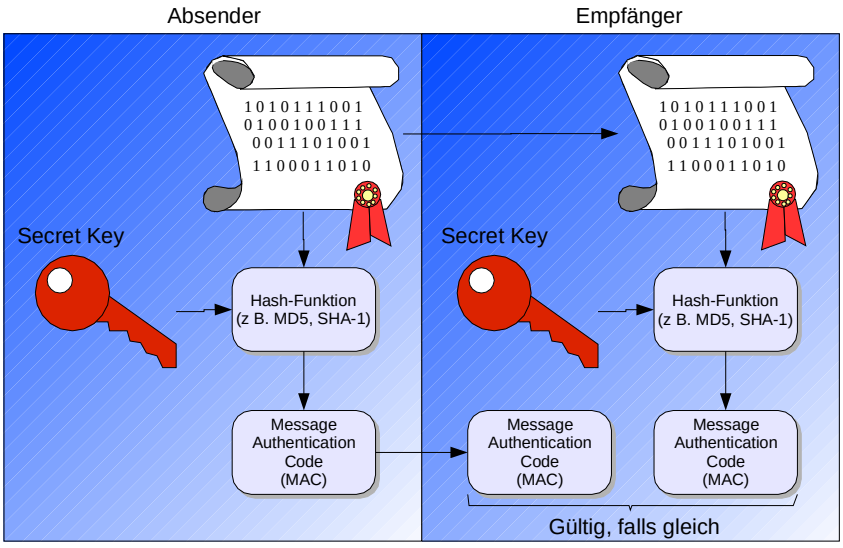


Abbildung 6 Digitale Signatur mit Secret Key Verschlüsselung

Meistens wird die digitale Signatur mit dem Verfahren der Public Key Verschlüsselung angewendet, also mit einem privaten Schlüssel für das Signieren und einem öffentlichen Schlüssel für die Kontrolle der Unterschrift. Allerdings können auch Signaturen mit Hilfe des Secret Key Verfahrens erstellt werden. Dazu wird für die Berechnung eines Message Digest der geheime Schlüssel in die Berechnung miteinbezogen. Der Empfänger der Nachricht und der digitalen Signatur berechnet nach dem gleichen Algorithmus und mit seinem eigenen geheimen Schlüssel einen unabhängigen und

selbstständigen Message Digest. Stimmen der empfangene Message Digest und der berechnete Message Digest überein, kann man sicher sein, dass die Nachricht und die Signatur von der Partnerinstanz stammte. Die Signatur mit Hilfe der Secret Key Verschlüsselung ist auch unter der Bezeichnung Message Authentication Code (MAC) bekannt.

Für eine digitale Unterschrift mit Secret Keys spricht vor allem die Effizienz und die Schnelligkeit des Verfahrens, da digitale Unterschriften mit geheimen Schlüsseln leicht und einfach zu berechnen sind. Damit eignet sich die Signatur mit Secret Keys vor allem für die Authentifizierung bei hohen Datenraten. Der Nachteil dieses Verfahrens liegt vor allem darin, dass der Empfänger ebenfalls im Besitze des geheimen Schlüssels sein muss, was wieder zum Problem der sicheren Verteilung des geheimen Schlüssels führt.

Mit dem Problem der Verteilung des geheimen Schlüssels sieht man sich bei der digitalen Unterschrift mit Public Keys nicht konfrontiert. Die öffentlichen Schlüssel können frei und offen verfügbar gemacht werden, so dass jeder die Authentizität der Nachricht prüfen kann. Wie bereits erwähnt ist dieses Verfahren extrem zeitaufwendig.

5.3 Protokolle und Standards

5.3.1 Abstract Syntax Notation One (ASN.1)

Eine abstrakte Syntax bietet eine Menge von formalen Regeln, um die Struktur von Objekten zu beschreiben. Die Beschreibung der Struktur erfolgt dabei unabhängig von der Präsentation für den Benutzer und die maschinenabhängigen Kodierungstechniken, welche für die Speicherung und Übertragung von Objekten verwendet werden.

Das Abstract Syntax Notation One (ASN.1) [X.680] ist ein Beispiel einer solchen Sprache. ASN.1 ist besonders gut für die Präsentation von komplexen, variablen und erweiterbaren Datenstrukturen geeignet, wie sie oft in modernen Anwendungen für die Kommunikation benutzt werden.

ASN.1 ist für uns sehr wichtig, da viele von benötigte Standards und Protokolle mit Hilfe von ASN.1 beschrieben sind. Neben PKCS#7, CMS und S/MIME wird ASN.1 auch in vielen weiteren Standards und Applikationen verwendet. X.509 Zertifikate, Simple Network Management Protocol (SNMP) und die Management Information Base (MIB) sind nur ein paar wenige Bespiele.

Für die Beschreibung der Strukturen definiert ASN.1 eine Auswahl von einfachen und strukturierten Typen. In unseren behandelten Standards sind die einfachen Typen OCTET STRING und OBJECT IDENTIFIER oft erwähnt. Beim einfachen Typ OCTET STRING handelt es sich um eine Zeichenkette. Der Typ OBJECT IDENTIFIER ist ein Bezeichner für ein Objekt, wie ein Algorithmus, ein Attributtyp oder zum Beispiel eine Registrierungsbehörde, welche weitere Bezeichner für Objekte definiert. Der Wert eines OBJECT IDENTIFIER kann eine beliebige Anzahl von Komponenten besitzen, die allgemein jeden Wert einer ganzen, nicht negativen Zahl annehmen kann. Die Objektbezeichner in ASN.1 sind in einer hierarchischen Baumstruktur organisiert, welche es ermöglicht, jedem Objekt einen globalen und eindeutigen Wert zuzuweisen.

Mit Hilfe von ASN.1 können weitere Typen definiert werden. Die Syntax um in ASN.1 einen Typ zu definieren besitzt folgende Struktur:

```
Syntax: <type name> ::= <type>

Beispiel: Data ::= OCTET STRING
```

Neben der Definition von weiteren Typen, gestattet ASN.1 auch die Definition von Untertypen aus bereits bestehenden Typen und besitzt folgende Syntax:

```
Syntax: <subtype name> ::= <type>(<constraint>)

Beispiel: IpAddress ::= OCTET STRING(SIZE (4))
```

Die Syntax um einem definierten Typ oder Untertyp einen Wert zu zuweisen, besitzt folgende Struktur:

```
Syntax: <value name> <name> ::= <value>

Beispiele:
content Data ::= '0ABCDF5873894618FE1294638276456646FF'H
internet OBJECT IDENTIFIER ::= { iso(1) org(3) dod(6) 1}
```

Neben den einfachen Typen können in ASN.1 mit den Schlüsselwörtern SEQUENCE, SEQUENCE OF, SET und SET OF auch strukturierte Typen deklariert werden. In unseren betrachteten Standards werden vor allem strukturierte Typen mit dem Schlüsselwort SEQUENCE erzeugt. Dieser strukturierte Typ stellt eine Sammlung von einer beliebigen Anzahl Variablen gleicher oder unterschiedlicher Typen dar. Die Reihenfolge der Variablen innerhalb der Struktur ist von Bedeutung und muss auch bei der Zuweisung von Werten und der Übertragung eingehalten werden. Unten stehendes Beispiel ist aus Kapitel 5.3.6.2 Generelle Syntax und beschreibt eine Sequenz mit dem Namen ContentInfo, welche aus den Feldern contentType und content besteht. Das Feld contentType ist vom Typ ContentType und ist als OBJECT IDENTIFIER definiert. Das zweite Feld content wird jeweils durch den Wert des OBJECT IDENTIFIERs in contentType bestimmt.

```
ContentInfo ::= SEQUENCE {
          contentType ContentType,
          content [0] EXPLICIT ANY DEFINED BY contentType
}

ContentType ::= OBJECT IDENTIFIER
```

Weiter werden in den RFC's und damit auch in unseren Erläuterungen oft viele strukturierte Typen mit SET OF deklariert. Mit SET OF werden Sammlungen ermöglicht, welche eine beliebige Anzahl Objekte enthalten, die alle vom gleichen Typ sind. Die Reihenfolge dieser Objekte innerhalb dieser Sammlung ist unwichtig. Unser Beispiel ist aus Kapitel 5.3.6.4 SignedData Type und enthält eine Sammlung von Identifiern, welche die verschiedenen Message Digest-Algorithmen kennzeichnen.

```
DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier
```

Neben den schon bereits angesprochenen Typen ist auch der Typ CHOICE zu erwähnen. Wie der Name andeutet stellt dieser Typ eine Auswahl zwischen verschiedenen, auswählbaren Typen dar. Es werden also mehrere Optionen innerhalb von CHOICE angegeben, von denen eine ausgewählt wird. Als Beispiel ist SignerIdentifier aus Kapitel 5.3.6.4 SignerInfo aufgeführt. Die Auswahl SignerIdentifier stellt uns die Felder issuerAndSerialNumber vom Typ IssuerAndSerialNumber und subjectKeyIdentifier vom Typ SubjectKeyIdentifier zur Verfügung. Eines dieser Felder wird ausgewählt und mit den entsprechenden Werten gefüllt:

```
SignerIdentifier ::= CHOICE {
          issuerAndSerialNumber IssuerAndSerialNumber,
          subjectKeyIdentifier [0] SubjectKeyIdentifier
}
```

5.3.2 Encoding Rules

Die Objekte werden mit ASN.1 beschrieben und über einen physikalischen Übertragungskanal übertragen. Die Regeln für die Kodierung definieren die Abbildung von den Objekten in ASN.1, in die Syntax für die Übertragung der Objekte.

Die Basic Encoding Rules (BER) und Distinguished Encoding Rules (DER) [X.690] sind nur zwei Beispiele von Encoding Rules, welche für die Abbildung von ASN.1 Objekten als eine Zeichenkette (octet strings) definiert werden, um diese über einen oktett-orientierten Kommunikationskanal zu

transportieren.

Nach der Abbildung besteht ein ASN.1 Objekt aus einem Identifier-, einem Längen- und einem Content-Feld. Die Werte dieser Felder sind abhängig vom Typ des kodierten Objektes, den Attributwerten und den Encoding Rules.

Die unterschiedlichen Encoding Rules sind aus den verschiedenen Bedürfnissen entstanden, welche durch den Gebrauch von ASN.1 in unterschiedlichen Anwendungen verursacht werden. Die verschiedenen Encoding Rules sind auf die Anwendungen angepasst und versuchen diesen Bedürfnissen gerecht zu werden.

5.3.3 Session Initiation Protocol (SIP)

Der Abschnitt Session Initiation Protocol (SIP) ist mehrheitlich aus der Projektarbeit PA2 Sna 03/2 [PA_SIP] von Daniel Kaufmann und Andreas Stricker entnommen. Diese Projektarbeit diente als Grundlage für diese Diplomarbeit.

5.3.3.1 Signalisations-Protokoll für Telefonie und Multimedia

Das Session Initiation Protocol (SIP) [RFC3261] ist ein Signalisierungs-Protokoll für Internet-Telefonie und Multimedia Konferenzen. SIP baut Verbindungen auf, ändert deren Parameter und sorgt für einen korrekten Abbau der Verbindung. Neben dem typischen Verbindungsaufbau zweier Gesprächspartner kann SIP auch mehrere Teilnehmer verbinden und ist somit also geeignet um Konferenzgespräche einzuleiten.

Um auch grossen Infrastrukturen gerecht zu werden, wird SIP normalerweise nicht nur zur Signalisierung zwischen den beiden Gesprächspartnern eingesetzt, sondern bietet auch Mechanismen zu Directory-Registrierung und -Anfragen. Da der Aufenthaltsort eines Gesprächspartners häufig ändern kann, läuft der Verbindungsaufbau üblicherweise über einen oder mehrere Proxy-Server. Der Client registriert sich beim Proxy mit den Benutzerdaten. Fortan weiss der Proxy, wo sich der Client befindet und kann so eingehende Gespräche dorthin weiterleiten.

Weitere, relevante RFCs zu SIP: [RFC3263], [RFC3264], [RFC2976], [RFC2327], [RFC3264]

5.3.3.2 SIP Infrastruktur

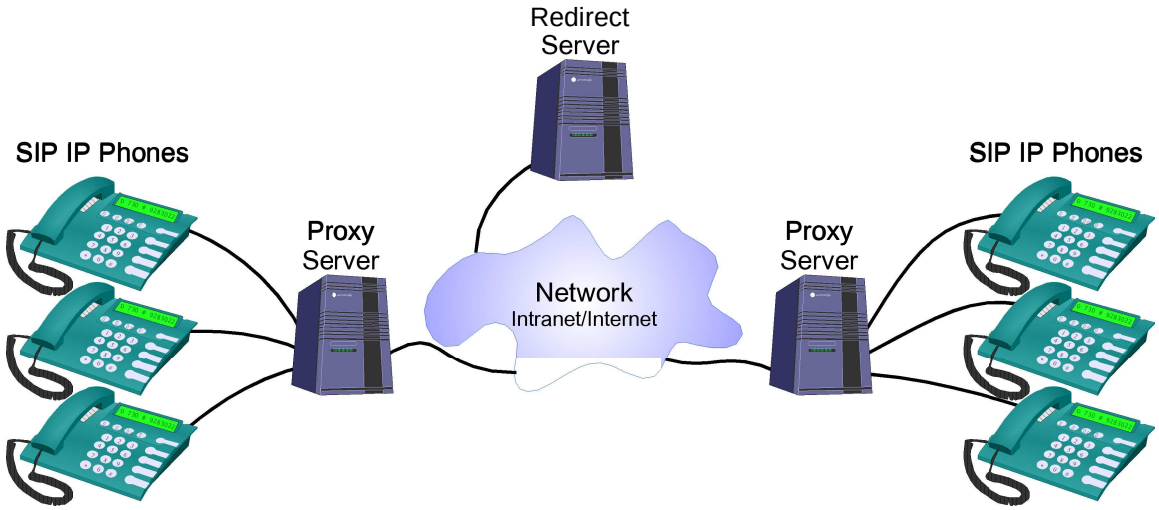


Abbildung 7 Typische SIP IP Telefonie Infrastruktur

Grundsätzlich können zwei UAs (SIP IP Telefon oder Softphone) direkt miteinander kommunizieren. In grösseren Umgebungen ist diese Art der Verbindungsaufnahme nicht handhabbar. Die IP-Adressen können wechseln oder die Benutzer wechseln das Gerät. Zudem sind Zusatzfunktionen, wie Voicemail

und Anrufbeantworter, so nicht realisierbar.

Üblicherweise wird die SIP-Signalisation über eine Proxy/Redirect-Server Infrastruktur abgewickelt. In Abbildung 7 ist eine typische Situation von zwei Firmen oder Niederlassungen aufgeführt[i]. Jeder Proxy-Zuständigkeitsbereich kann eine Vielzahl von UAs bedienen. Der Proxy-Server arbeitet zustandslos und übernimmt eine Art Routing-Funktion. Die genaue Lokation der UAs kennt der Redirect-Server[ii]. Dieser wird normalerweise nicht direkt vom UA angesprochen: Anmeldungen (Registration) und Weiterleitungen werden durch den Proxy weitergeleitet und abgefragt. So muss der UA nur seinen Proxy kennen.

5.3.3.3 SIP Meldungen und Signalisations-Fluss

Einfaches Gespräch

Ein Gespräch läuft idealerweise so wie in Abbildung 8 ab. Alice will Bob anrufen. Alice wird vom Proxy atlanta.com und Bob vom Proxy biloxy.com verwaltet. So schickt Alice die **`INVITE`** Meldung (1) mit Bob's SIP URI an ihren Proxy altlanta.com. Dieser leitet die Meldung weiter an den Proxy von Bob (2) und bestätigt Alice mit **`100 Trying`** (3), dass er den **`INVITE`** erhalten hat. Währenddessen hat der biloxi-Proxy die **`INVITE`** Meldung (4) bereits an Bob weitergereicht. Der atlanta.com Proxy erhält ein **`100 Trying`** (5). Der UA klingelt nun in der Hoffnung, dass Bob das Gespräch annimmt und bestätigt das mit einem **`180 Ringing`** (6). Dieses wird durch die Proxys weitergereicht (7) und kommt bei Alice an (8). Mittlerweile hat Bob das Telefon abgenommen, was mittels eines **`200 OK`** (9) über die Proxys (10) Alice (11) mitgeteilt wird. Alice bestätigt die Einleitung des Gespräches mit einem Acknowledge **`ACK`** (12), dass nun direkt an den UA von Bob gesendet wird. Ebenso direkt wird ein Medien-Kanal zwischen den beiden UAs aufgebaut, über den das Gespräch[iii] abgewickelt wird. Irgendwann ist jedes Gespräch zu Ende: Hier beendet z.B. Bob das Gespräch. Der UA von Bob signalisiert dies mit einem **`BYE`** (13) an Alice. Sie wiederum bestätigt das mit einem **`200 OK`** (14).

SIP URI

Ein Teilnehmer wird anhand einer SIP URI identifiziert. Die SIP URI gleicht einer Email-Adresse. Zum Beispiel **`sip:hugo@sip.atlanta.com`**. Für eine gesicherte Verbindung über TLS gilt analog zu HTTPS bzw. IMAPS das Protokoll-Präfix **`sips`**. Der Domain-Name ist üblicherweise nicht der des UA, sondern der des zuständigen Proxy-Servers. Die reservierten Ports für SIP sind in Tabelle 2 zu beachten.

Eine SIP/SIPS URI kann weitere Parameter enthalten, wie Port und Transport-Protokoll (UDP/TCP).

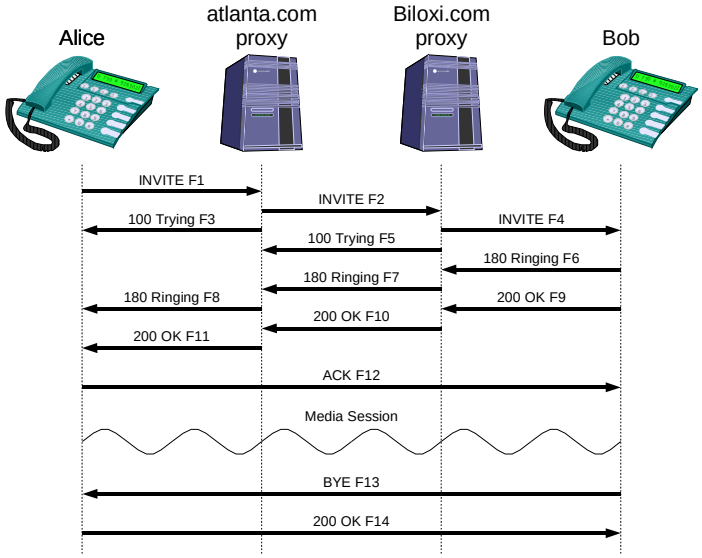


Abbildung 8 SIP Session aus RFC 3261 Seite 11

i Im Folgenden wird eine Niederlassung oder eine Firma mit einem eigenen Proxy als Proxy-Zuständigkeitsbereich bezeichnet.

ii Ein Server muss nicht zwingend ein eigenständiger Rechner sein. Ein Server ist ein Dienst, der auf einer oder mehreren physikalischen Maschinen läuft.

iii Natürlich kann neben Voice auch Video und andere multimediale-Daten versendet werden.

Protokol	Port	Bedeutung
SIP	5060 UDP/TCP	SIP unverschlüsselt als Datagramm oder Stream
SIPS	5061 TCP	SIP mit TLS verschlüsselt über Stream

Tabelle 2 Reservierte Ports für SIP

SIP Nachricht

Im Folgenden schauen wir tiefer in das Protokoll anhand echter Mitschnitte einer einfachen Verbindung. Dazu wurde der Versuchsaufbau wie in Abbildung 9 dargestellt aufgebaut.

UA 2 (160.85.170.139) rufe den UA 1 (160.85.170.138) über den Proxy (160.85.162.81) an.

SIP ist HTTP [RFC2616] nachempfunden und verwendet die gleiche Syntax. Jede Transaktion ist entweder eine Methode (**`INVITE`**, **`ACK`**, **`BYE`**) oder eine Statusmeldung (**`100 Trying`**, **`200 OK`**). Nach der Methode oder Statusmeldung folgen Attribute, welche weitere Parameter transportieren.

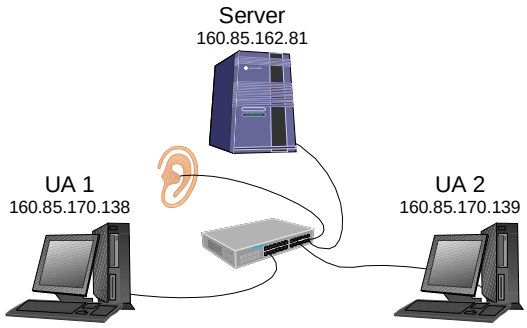


Abbildung 9 Aufbau des Versuches

Die Nutzdaten werden anhand des MIME-Typ unterschieden und folgen dem eigentlichen SIP-Header, getrennt durch eine Leerzeile. Normalerweise transportiert SIP das SDP-Protokoll [RFC2327], in dem die Details über die Medien-Verbindung ausgehandelt werden.

```
INVITE sip:6666@dskt6621.zhwin.ch SIP/2.0
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 INVITE
Max-Forwards: 70
Subject: VovidaINVITE
Contact: <sip:4444@160.85.170.139:5060>
User-Agent: Vovia-SIP-SoftPhone/0.1  (www.vovida.org)
Content-Type: application/sdp
Content-Length: 220
Session Description Protocol not shown here
```

UA 2 schickt an den Proxy-Server die **`INVITE`** Meldung. Diese enthält die SIP URI **`sip:user@host`** und die Protokoll Version **`SIP/2.0`**.

Der folgende Header **`Via`** listet die an dem Transfer beteiligten Knoten auf, zunächst nur mal der UA 2.

Das Ziel – der UA 1 wird im Header **`To`** als SIP-URI angegeben. Im Header **`From`** wird nochmals die eigene UA SIP URI mit einem eindeutigen Bezeichner angegeben.

Jeder Anruf bekommt eine eindeutige ID, um mehrere Gespräche von einem UA auseinander zu halten. Diese ID wird im Header **`Call-ID`** übermittelt. Die ID sollte eindeutig (zufällig) sein.

`CSeq` wird für jede Verwendung einer Methode inkrementiert. Danach folgt der Namen der Methode.

Um die Auswirkung von Schlaufen zu minimieren, wird **`Max-Forwarders`** (ein Zähler) bei jedem Hop dekrementiert. Bei Null wird die SIP-Meldung verworfen.

Das **`Subject`** könnte einen Text für den Anruf beinhalten, allerdings schreiben die meistens UAs wenig sinnvolle Texte hinein.

Bei **`Contact`** wird nochmals die URI für den UA aufgelistet.

Den Typ des UA wird mit **`User-Agent`** mitgeteilt.

Nun folgt die Angabe des MIME-Typ im Header **Content-Type**, sofern die SIP-Meldung weitere Daten mitführt. Die Länge der Daten wird über den Header **Content-Length** mitgeteilt. Dieser muss auch dann vorhanden sein, wenn keine Daten übertragen werden (mit Länge 0).

Im nun folgenden SDP-Teil gibt der UA seine Wünsche für die Medien-Verbindung an, d.h. unterstützte Protokolle, Codecs und dessen Parameter.

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 INVITE
Content-Length: 0
```

Sobald der Proxy-Server das INVITE empfangen hat, bestätigt er das dem UA 2 mit einem **100 Trying**. Nachfolgend folgt eine Kopie der Header damit der UA feststellen kann, zu welchem Anruf die Meldung gehört.

```
INVITE sip:6666@160.85.170.138:5060 SIP/2.0
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=aa6140793e11.4
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=a83f45c7736e.2
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 INVITE
Max-Forwards: 69
Subject: VovidaINVITE
Record-Route: <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>,
              <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>
Contact: <sip:4444@160.85.170.139:5060>
User-Agent: Vovia-SIP-SoftPhone/0.1  (www.vovida.org)
Content-Type: application/sdp
Content-Length: 220
Session Description Protocol not shown here
```

Hier sieht man die gleiche **INVITE**-Meldung wie oben nochmals, diesmal aber vom Proxy-Server zum UA 1, ergänzt um Routing-Informationen:

- Weitere **Via**-Header die den Weg der Meldung durch den Proxy wiederspiegeln
- **Record-Route** die ebenfalls den Weg der Meldung wiedergeben.

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=aa6140793e11.4
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=a83f45c7736e.2
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 INVITE
Content-Length: 0
```

Auch die zweite **INVITE**-Meldung wird mit einem **100 Trying** quittiert.

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=aa6140793e11.4
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=a83f45c7736e.2
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=4c190595
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 INVITE
Record-Route: <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>,
              <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>
Content-Length: 0
```

Der UA 1 klingelt und teilt das mittels dieser Meldung dem UA 2 mit. Allerdings wird diese Meldung zuerst an den Proxy-Server geschickt.

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=4c190595
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 INVITE
Record-Route: <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>,
              <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>
Content-Length: 0
```

Die **180 Ringing** Statusmeldung nach dem Proxy ist von den **Via**-Header gesäubert. Anhand der Route (**Record-Route**) ist der Weg immer noch sichtbar.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=aa6140793e11.4
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=a83f45c7736e.2
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=e79cab79
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 INVITE
Record-Route: <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>,
              <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>
Contact: <sip:6666@160.85.170.138:5060>
Content-Type: application/sdp
Content-Length: 220
Session Description Protocol not shown here
```

UA 1 hat das Gespräch angenommen und teilt dies mit der Status-Meldung **200 OK** dem UA 2 über den Proxy-Server mit. Im angehängten SDP-Teil wird er seine Wahl der Verbindung mitteilen, die sich – sofern möglich – mit den Angaben von UA 2 deckt.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=e79cab79
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 INVITE
Record-Route: <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>,
              <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>
Contact: <sip:6666@160.85.170.138:5060>
Content-Type: application/sdp
Content-Length: 220
Session Description Protocol not shown here
```

Die gleiche Meldung nach dem Proxy-Server, wieder um die **Via**-Header bereinigt.

```
ACK sip:6666@dskt6621.zhwin.ch SIP/2.0
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=e79cab79
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 ACK
Max-Forwards: 70
Route: <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>,
       <sip:6666@160.85.170.138:5060>
Content-Length: 0
```

UA 2 bestätigt die Annahme des Gesprächs mit der Methode **ACK**.

Im Header **Route** wird die Route über die Proxys aufgelistet, über welche die folgenden Nachrichten gesendet werden sollen.

```
ACK sip:6666@160.85.170.138:5060 SIP/2.0
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=aa6140793e11.4
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=770b7e975316.2
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=e79cab79
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 ACK
Max-Forwards: 69
Content-Length: 0
```

Nach dem Proxy fehlen die Route-Header, dafür sind die **Via**-Header wieder präsent.

Nach dem Eintreffen der **ACK**-Meldung werden die beiden UAs eine Media-Verbindung aufbauen und das Gespräch über diese führen.

```
BYE sip:6666@160.85.162.81:5060;maddr=160.85.162.81 SIP/2.0
Via: SIP/2.0/UDP 160.85.170.139:5060
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=e79cab79
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 2 BYE
Max-Forwards: 70
Route: <sip:6666@160.85.162.81:5060;maddr=160.85.162.81>,
       <sip:6666@160.85.170.138:5060>
Content-Length: 0
```

Der UA 2 leitet die Trennung der Verbindung ein, indem er die **BYE**-Methode aufruft. Auch das läuft über den Proxy, da die **Route** definiert ist.

```
BYE sip:6666@160.85.170.138:5060 SIP/2.0
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=aa6140793e11.4
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=9d2222b269b2.2
Via: SIP/2.0/UDP 160.85.170.139:5060
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=e79cab79
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 2 BYE
Max-Forwards: 69
Content-Length: 0
```

Die **BYE**-Meldung wird vom Proxy um die Via-Header ergänzt.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=a41e92033831.4
Via: SIP/2.0/UDP 160.85.162.81:5060;branch=9d2222b269b2.2
Via: SIP/2.0/UDP 160.85.170.139:5060
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=e79cab79
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 2 BYE
Content-Length: 0
```

UA 1 bestätigt den Verbindungs-Abbruch mittels einer **200 OK** Statusmeldung.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 160.85.170.139:5060
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=e79cab79
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 2 BYE
Content-Length: 0
```

Die gleiche Meldung nach dem Proxy. Sobald diese den UA 2 erreicht, gilt das Gespräch als beendet.

SIP Methoden und Status Codes

Die folgenden Tabellen zeigen eine Übersicht über die wichtigsten SIP Methoden und Status-Codes. Es handelt sich um die in [VoIPGW] aufgeführten Tabellen.

Methode	Funktion
INVITE	Ein UA wird damit zu einer Session eingeladen. Diese Nachricht kann bereits Informationen zum Typ der Session enthalten.
ACK	Dieser Request bestätigt den Erhalt einer endgültigen Respond-Message auf ein INVITE. Diese Methode wird nur im Zusammenhang mit INVITE verwendet. Falls das INVITE keine Informationen zum Typ der Session enthält, müssen in dieser Message die Informationen vorhanden sein.
OPTIONS	Durch diese Nachricht können optionale Informationen über den Benutzer weitergegeben werden.
BYE	Der UA teilt mit BYE dem Server mit, dass er die Verbindung beenden möchte. Eine BYE-Message kann entweder vom Anrufenden oder vom Gerufenen gesendet werden.
CANCEL	Bricht einen hängigen Verbindungswunsch ab mit derselben Call-Id, To, From und Cseq wie der vorhergehende Request. CANCEL hat keinen Einfluss auf terminierte Requests, z.B. falls auf ein INVITE bereits ein ACK geschickt wurde.
REGISTER	Ein Client benutzt diese Message, um sich mit seiner Adresse bei einem SIP-Server zu registrieren.

Tabelle 3 SIP Methoden

Status Kategorie	Funktion	Beschreibung	Beispiele
1xx	Informationsstatus	Die Anfrage wurde erhalten, die Anfrage wird gehalten.	100 Trying 180 Ringing
2xx	Erfolgsstatus	Die Aktion wurde empfangen, verstanden und akzeptiert	200 Ok
3xx	Redirectstatus	Weitere Aktionen müssen ausgeführt werden um die Anfrage auszuführen	301 Moved Permanently
4xx	Client-Error	Die Anfrage enthält falsche Angaben oder kann vom Server nicht beantwortet werden.	400 Bad Request 486 Busy-Here
5xx	Server-Error	Beim Server ist ein Fehler beim Beantworten der Anfrage aufgetreten.	503 Service Unavaiable
6xx	Allgemeiner Fehler	Die Anfrage kann von keinem Server beantwortet werden	600 Busy

Tabelle 4 SIP Status Codes

Redirection

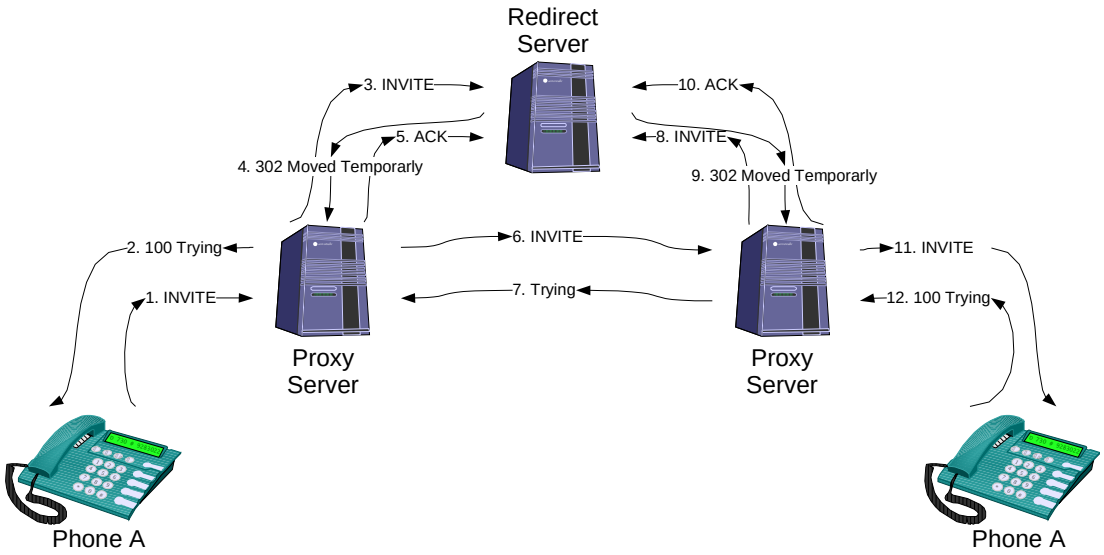


Abbildung 10 Funktion des Redirect-Server

Was passiert, wenn der Proxy-Server den UA nicht kennt? Der Proxy-Server kontaktiert in diesem Fall seinen Redirect-Server, der ihm den entsprechenden Ort mitteilen kann. Der UA wird nie direkt mit dem Redirect-Server kommunizieren.

Nachdem der linke Proxy die **`INVITE`**-Meldung (1) erhalten hat, wird er feststellen, dass er den Ziel-UA nicht kennt. Also fragt er den Redirect Server nach der Route, indem die **`INVITE`**-Meldung (3) an diesen weitergeleitet wird. Als Antwort bekommt er ein **`302 Moved Temporarly`** (4), mit dem rechten Proxy als Ziel. Somit schickt er die **`INVITE`**-Meldung (6) an den rechten Proxy. Auch der rechte Proxy kennt den gewünschten UA noch nicht, so dass er ebenfalls den Redirect-Server anfragen muss (8). In der folgenden **`302 Moved Temporarly`** (9) Meldung, wird nun die Adresse des gewünschten UAs zurückgegeben. Damit kann der Proxy die **`INVITE`**-Meldung (11) an den UA senden.

In den folgenden Meldungen ist der Pfad durch die Proxys durch die **`Via`**- und **`Record-Route`**-Header gegeben, so dass keine weiteren Anfragen an den Redirect-Server nötig sind.

5.3.4 Session Description Protocol (SDP)

Für den Mbone (multicast backbone) wird ein Hilfsprogramm für Multimedia-Anwendungen benötigt. Dieses soll Multimedia-Konferenzen ausschreiben, zum Verteilen der Adresse und spezifischen Informationen dienen, welche für die Teilnahme benötigt werden. SDP [RFC2327] ist zur Beschreibung von solchen Multimediasitzungen vorgesehen und speziell zum Zweck der Bekanntmachung, Einladung und anderen Formen der Initierung solcher Verbindungen. Es werden Informationen über die Media Streams der Sitzungen versendet, die es dem Empfänger erlauben an der Sitzung teilzunehmen.

SDP ist alles in allem also ein Mittel um die Existenz einer Sitzung mitzuteilen und um genügend Informationen zu versenden, die es ermöglichen sollen, an dieser teilzunehmen.

5.3.4.1 Multicast Mitteilung

Dies ist ein häufig benötigter Modus. Der Client meldet eine Konferenzsitzung durch periodisches Aussenden eines Pakets an eine bekannte Multicastadresse und Port. Das Paket verwendet das Session Announcement Protocol (SAP). SAP Pakete sind UDP Pakete mit einem Header, auf welchen die Nutzdaten (Payload) mit der eigentlichen SDP Sitzungsbeschreibung folgt.

5.3.4.2 Email und www Mitteilung

Andere Mittel um solche Sitzungsbeschreibungen zu versenden enthält das Email und World Wide Web. Für beide Verteilungen sollte der MIME-Inhalt application/sdp gebraucht werden. Das ermöglicht das automatische Lancieren von Applikationen zur Sitzungsteilnahme.

5.3.4.3 SDP Inhalt

Einige Informationen müssen im SDP enthalten sein, damit auch genügend Angaben zur eindeutigen Identifizierung vorhanden sind. Hierzu gehören vor allem allgemeine Angaben zur gewünschten Verbindung.

- Sitzungsname und Zweck
- Zeit, wie lange die Sitzung aktiv ist
- Medien, welche die Sitzung umfasst
- Informationen, um diese Medien zu erhalten (z. B. Adresse, Port, Format)
- Weitere mögliche Informationen sind die Bandbreite oder Kontaktinfos über die verantwortliche Person

Mediuminformationen

Damit werden diverse Informationen über das Medium definiert, um dem Empfänger genaue Angaben zu den unterstützten Formaten liefern zu können.

- Mediumtyp (video, audio)
- Transportprotokoll (RTP, UDP, IP)
- Format des Mediums (H.261 video, MPEG video...)
- Adresse und Port für das Medium

5.3.4.4 SDP Spezifikation

Eine SDP Sitzungsbeschreibung besteht aus mehreren Linien Text der Form:

```
<Typ>=<Wert> <Typ>
```

Das SDP Paket selbst besteht aus verschiedenen Ebenen, die mehrere Linien enthält. Zuerst ist die Sitzungsebene, welche sich auf die ganze Sitzung bezieht, gefolgt von mehreren optionalen Medienebenen, welche Einzelheiten über einen einzigen Media Stream enthalten.

Der Start der Sitzungsebene erkennt man am „v=“, die Medienebene wird durch „m=“ gekennzeichnet. Der ganze Aufbau setzt sich nun aus verschiedenen Ebenen zusammen, wobei die Medienebene nie oder mehrmals vorkommen kann.

Sitzungsbeschreibung (Sitzungsebene)

```
 Sitzungsbeschreibung (Sitzungsebene):
        v=  Protokollversion
        o=  Besitzer und Sitzungserkennung
        s=  Sitzungsname
        i=* Sitzungsinformationen
        u=* URI
        e=* eMail Adresse
        p=* Telefonnummer
        c=* Kontaktinformationen
        b=* Informationen über die Bandbreite
        Ein oder mehrere Zeitbeschreibungen (siehe weiter unten)
        z=* Zeitzonenabgleich
        k=* Schlüssel für Verschlüsselung
        a=* ein oder mehrere Attributlinien
        Ein oder mehrere Medienbeschreibungen (siehe weiter unten)

Zeitbeschreibung:
       t=  Wie lange Sitzung aktiv ist(time the session is active)
       r=* 0 oder Wiederholungszeit (Bsp Jeden Montag um 10.00 Uhr)
```

Medienbeschreibung (Medienebene)

```
Medienbeschreibung (Medienebene)
        m=  Medienname und Transprotadresse
        i=* Medientitel
        c=* Kontaktinformationen (optional, wenn bereits im
                                              Sitzungslevel definiert)
        b=* Bandbreiteninformationen
        k=* Schlüssel zur Verschlüsselung
        a=* ein oder mehrere Attributlinien
```

Die mit * gekennzeichneten Linien sind optional und müssen im SDP nicht verwendet werden. Sie dienen zur genaueren Beschreibung der übertragenen Daten.

5.3.4.5 Minimales SDP

Wie bereits erwähnt, wird das Session Description Protocol [RFC2327] gebraucht um die Komponenten des Kommunikationskanals zu beschreiben, welche ausgehandelt werden, wenn ein Endpunkt versucht einen Anruf zu tätigen. Diese Komponenten beinhalten Codecs, Ports und das Streaming-Protokoll, welche eine Einweg- oder Zweiwegübermittlung ermöglichen.

Normalerweise wird dieser SDP Header mit der **INVITE** und der **200 Ok** Nachricht verschickt und beschreibt dann wie ein Data Stream über das Real-time Transport Protocol (RFC 1889) übertragen wird.

Eine mögliche **INVITE** Nachricht mit SDP hat folgenden Aufbau. In der Folge werden die einzelnen

Parameter der Felder noch genauer beschrieben.

```
INVITE sip:
...
Content-Type: application/sdp
Content-Length : 168

v=0
o=- 3565866574 3565866574 IN IP4 192.168.80.134
s=DA SIP Security 2003
c= IN IP4 192.168.80.134
t= 0 0
m=audio 1234 RTP/AVP 0
a=rtpmap :0 PCMU/8000
a=ptime:20
```

Beschreibung der einzelnen Komponenten

v: Versionsnummer

Diese wird immer auf 0 gesetzt und zeigt den Beginn des SDP Inhaltes an.

o: Session origin und Besitzername

Der Aufbau dieser Komponente setzt sich zusammen aus:

```
o = <Username> <SitzungsID> <Version> <Netztyp> <Adresstyp> <Adresse>
Bsp.: o=- 3565866574 3565866574 IN IP4 192.168.80.134
```

zusammen. Der Benutzername ist entweder Loginname des Host oder wird auf „-“ gesetzt. Die Sitzung-ID ist ein numerischer Wert, so dass aus dem Benutzernamen, SitzungsID, Netzwerktyp, Adresstyp und der Adresse eine globale, eindeutige Bezeichnung entsteht. Wenn der Besitzer unbekannt ist, so werden die Session-ID und die Version meistens auf die Startzeit oder einen Zufallswert gesetzt. Der Netztyp ist IN und steht somit für Internet. Beim Adresstyp sind zur Zeit die Parameter für IPv4 und IPv6 definiert. Die Adresse enthält die globale, eindeutige Adresse des Rechners, welcher die Sitzung erstellt hat.

s: Sitzungsname

Dies ist ein Name, der zur Bezeichnung der Sitzung dient.

c: Anschlussinformationen

```
c =  <Netzwerktyp> <Adresstyp> <Kontaktadresse>
Bsp.: c= IN IP4 192.168.80.134
```

In einer Sitzungsankündigung muss das „c=“ Feld entweder in jeder Medienbeschreibung oder in der Sitzungsebene enthalten sein. Wenn im Sitzungs- und Medienlevel diese Informationen mitgegeben werden, so haben die Daten im Medienlevel höhere Priorität. Mit IN beim Netzwerktyp ist das Internet gemeint, beim Adresstyp ist zur Zeit nur IPv4 unterstützt.

t: Zeit, wie lange die Sitzung aktiv ist.

```
t = <Startzeit> <Stopzeit>
```

Das Feld deklariert die Start- und Stopzeit für eine Konferenz. Jedes zusätzliche „t=“ Feld spezifiziert eine weitere Periode, in welcher die Sitzung aktiv ist. Wenn die Sitzung immer zu regelmässigen Zeiten aktiv sein soll, so ist es besser das „r=“ Feld zu benutzen. Die Werte sind dezimale Darstellungen des Network Time Protocol (NTP). Es ist auch möglich die Werte auf 0 zu setzen, was bedeutet, dass die Verbindung als permanent angesehen wird.

```
r = <Wiederholungsintervall> <Dauer der Sitzung> <Gültigkeitsdauer>
```

Standardmässig sind alle Zeitangaben in Sekunden zu lesen.

m: Mediennamen und Transportadresse

```
m =  <Medium> <Port>/<Anzahl der Ports> <Transport> <fmt Liste>
Bsp.: m=audio 1234 RTP/AVP 0 18
```

Hier wird der Medientyp, die Transportadresse und die vom Sender unterstützten Codecs beschrieben. Im ersten Feld Medium gibt es zur Zeit die Möglichkeiten **audio**, **video**, **application**, **data** und **control.** Im zweiten Feld ist der Port angegeben zu welchem der Media Stream gesandt wird. Die Bedeutung des Ports hängt vom verwendeten Netzwerk (des „c=“ Feldes) und vom verwendeten Transportprotokoll ab. So sind beispielsweise bei UDP nur Werte im Bereich von 1024 bis 65535 gültig. Beim Feld Transport gibt es im Moment die Möglichkeiten RTP/AVP und UDP. Im letzten Feld werden die unterstützten Codecs mitgeliefert. Es kann sein, dass hier Codecs enthalten sind, welche vom Empfänger nicht unterstützt werden. In diesem Fall sendet der Empfänger eine Nachricht zurück mit einer Liste von Codecs, die er unterstützt. Falls RTP/AVP für den Transport gebraucht wird, so beinhaltet die <fmt Liste> eine Reihe von Integer, die den Codec genauer bezeichnen. Die komplette Liste kann im [RFC1890] und unter der Internet Assigned Numbers Authorization (IANA) Homepage nachgeschlagen werden.

Identifier	Codec
0	G.711 uLaw
2	G.726-32
3	GSM
4	G.723
8	G.711 aLaw
18	G.729

Table 5 Kurzer Auszug aus dem [RFC1890] mit den Codecs und den zugehörigen Bezeichnern

a: Attributlinien

Jeder Codec kann zusätzliche Information haben, welche über dieses Feld übermittelt werden. So beschreibt PCMU/8000, dass es sich hier um G.711 uLaw handelt und die Abtastraste 8kHz beträgt. Die Linie ptime:20 sagt aus, dass jedes übertragene Paket 20 Millisekunden des Voice Traffic (Sprachverkehrs) beinhaltet.

k: Verschlüsselungsschlüssel

```
k =  <Methode> <Schlüssel>
Bsp.:
k=clear:2f732bb15192975b600cd18696c791a4de676140ff02c70c4072600da703
```

Bei der Methode gibt es weitere Möglichkeiten wie **Base64**, **URI** oder **prompt**. Ist hier **clear** definiert, so wird der Schlüssel für die Übermittlung nicht verändert. Weitere Möglichkeiten sind Base64, um binäre Schlüssel zu übertragen.

5.3.4.6 SDP unseres Clients

Für unser Programm haben wir ein minimales SDP erstellt. Einzig der Schlüssel kam hinzu. Die Start- und Stopzeit ist auf 0 gesetzt, damit wir eine permanente Verbindung haben. Weiter wird im Moment nur ein Medienformat mitgegeben und dies beinhaltet das G.711 uLaw.

```
INVITE sip:
...
Content-Type: application/sdp
Content-Length : 212

v=0
o=- 3565866574 3565866574 IN IP4 192.168.80.134
s=DA SIP Security 2003
c= IN IP4 192.168.80.134
t= 0 0
k=clear:2f732bb15192975b600cd18696c791a4de676140ff02c70c4072600da703
m=audio 1234 RTP/AVP 0
a=rtpmap :0 PCMU/8000
a=ptime:20
```

5.3.5 Multipurpose Internet Mail Extensions (MIME)

Zum guten Verständnis von S/MIME [RFC2633] müssen Kenntnisse über die zugrunde liegenden E-Mail-Formate vorhanden sein, insbesondere über MIME [RFC2045], [RFC2046], [RFC2047], [RFC2048] und [RFC2049]. Damit aber die Bedeutung von MIME erfasst werden kann, müssen wir zurück bis zum traditionellen E-Mail-Standard [RFC822] gehen, welcher immer noch in Gebrauch ist. Aus diesem Grund folgt zuerst eine kurze Einführung in den Standard [RFC822], bevor anschliessend MIME und später in einem separaten Abschnitt S/MIME diskutiert wird.

5.3.5.1 RFC 822

In [RFC822] wird das Format für das Versenden und Empfangen von Texten mittels E-Mail definiert. Im Internet wurde damit ein Standard gebildet, welcher immer noch in Gebrauch ist. Aus Sicht von [RFC822] besteht eine Nachricht aus einem Umschlag und dem eigentlichen Inhalt. Der Umschlag enthält alle nötigen Informationen zur Übertragung und Auslieferung der E-Mail. Das Objekt, das an den Empfänger ausgeliefert wird, nennt man Inhalt. [RFC822] bezieht sich nur auf den Inhalt, beinhaltet aber weiter eine Reihe führender Felder, aus denen das E-Mail-System den Umschlag erzeugen kann. Der Standard [RFC822] wurde darauf ausgelegt, dass Programme auf derartige Informationen zugreifen können.

Die Gesamtstruktur einer zum Standard [RFC822] konformen Nachricht ist sehr einfach. Eine Nachricht besteht im Wesentlichen aus zwei Teilen. Der erste Teil, der Header, wird durch einige Kopfzeilen gebildet. Der zweite Teil beinhaltet beliebigen Text und wird auch Body genannt. Eine Leerzeile trennt den Header und den Body. Eine Nachricht besteht also aus ASCII-Text und alle Zeilen bis zur ersten Leerzeile werden als Kopfzeile betrachtet, welche das Mail-System für die Verwaltung benutzt.

Eine Kopfzeile besteht normalerweise aus einem Schlüsselwort, gefolgt von einem Doppelpunkt und den Parametern des Schlüsselwortes. Die am meisten verwendeten Schlüsselwörter sind: **`From`**, **`To`**, **`Subject`** und **`Date`** welche den Absender, den Empfänger den Betreff und das Datum, an dem die E-Mail aufgegeben wurde, spezifiziert. Das Format lässt die Aufteilung einer langen Kopfzeile der E-Mail in mehrere Zeilen zu.

Der Nachrichtentext der E-Mail besteht nach [RFC822] aus reinem Text (7-Bit-ASCII) und kann beliebig lang sein.

```
From: "Loretz Manfred, loretman" <loretman@zhwin.ch>
To: "Gisler Andreas, gislean1" <gislean1@zhwin.ch>
CC: "Stricker Andreas, stricand" <stricand@zhwin.ch>
Date: Wed, 1 Oct 2003 09:01:54 +0200
Subject: Die Syntax in RFC 822

Hallo. Dies ist der eigentliche Textkörper der Nachricht, welcher durch
eine Leerzeile vom Header der Nachricht getrennt ist.
```

5.3.5.2 Multipurpose Internet Mail Extensions

Beim Gebrauch von SMTP (Simple Mail Transfer Protocol) und anderen E-Mail-Protokollen treten Probleme und Beschränkungen auf. Als Erweiterung zu [RFC822] behandelt MIME einige dieser Probleme und Beschränkungen. In [RFC822], [WIL01] und [KT07]werden die folgenden Beschränkungen für SMTP/822 aufgeführt:

1. SMTP und das E-Mail-System im Internet wurden ursprünglich nur für 7-Bit-ASCII-Text ausgelegt. Somit können keine Texte mit nationalem Zeichensatz übertragen werden, weil hier 8-Bit-Zeichen mit Werten von dezimal 128 und höher vorkommen.
2. Da SMTP auf 7-Bit-ASCII beschränkt ist, können auch keine ausführbaren Dateien und andere binäre Objekte übertragen werden. Um Binärdaten, wie zum Beispiel eine Grafikdatei oder ein ausführbares Programm, mittels E-Mail senden zu können, mussten die binären Dateien in 7-Bit-ASCII-Text übersetzt werden. Ein Reihe von zusätzlichen Mechanismen und Schemata sind für diesen Vorgang erforderlich, aber keines davon ist Standard oder auch nur de-facto-Standard.
3. Ab einer bestimmten Länge, werden die Nachrichten von manchen SMTP-Server abgewiesen.
4. Nicht alle Implementation von SMTP entsprechen vollständig dem SMTP-Standard [RFC821]. Folgenden Probleme treten häufig auf:
    - Auslassen, Hinzufügen oder Verfälschen von Zeilenvorschub
    - Verkürzung oder neue Zeile bei Zeilen über 76 Zeichen
    - Weglassen von „White Space“-Zeichen (Tabulatoren und Leerzeichen) am Zeilenende
    - Auffüllen von Zeilen einer Nachricht auf gleiche Länge
    - Umsetzen von Tabulatoren-Zeichen in mehrere Leerzeichen
5. Die Beschränkungen von [RFC822] werden noch stärker ersichtlich, wenn SMTP-Gateways vorgesehen sind um einen Austausch zwischen E-Mail-Protokollen zu ermöglichen oder den Inhalt der Nachricht zwischen verschiedenen Zeichensätzen zu konvertieren.

Der Zweck von MIME besteht nun darin, die aufgezeigten Probleme in Übereinstimmung mit der existierenden Implementation nach dem Standard [RFC822] zu lösen. MIME soll zur Koordination und Vereinheitlichung der verschiedenen Methoden zum Austauschen von Binärdaten im Internet dienen. Die dazu angewendeten Mechanismen sind in den RFC's [RFC2045], [RFC2046], [RFC2047], [RFC2048] und [RFC2049] festgelegt.

Überblick

Um nun die besprochenen Beschränkungen der E-Mail-Protokolle zu lösen, beschreibt die MIME-Spezifikation [RFC2045] mehrere Mechanismen. MIME löst diese Beschränkungen ohne neue Probleme, bezüglich der Kompatibilität mit bereits bestehenden Mail-Systemen, zu verursachen.

Im einzeln beschreibt die Spezifikation [RFC2045] folgende Elemente:

1. **Header-Felder (Header-Fields):** Es wurden fünf neue Header-Felder eingeführt, welche im Header einer Nachricht stehen können. Die neuen Header-Felder stellen Informationen über den Hauptteil (Body) der Nachricht bereit.

2. **Inhaltstypen (Content Type):** Eine Vielfalt von Inhaltstypen wird definiert, welche eine grosse Menge unterschiedlicher Dateiarten verarbeiten, wie sie in einer Multimedia-Umgebung vorkommen.
3. **Übertragungsverschlüsselung (Content-Transfer-Encoding):** Für die Übertragung der Daten wurden Codierungen definiert. Diese Codierungen gestatten die Umsetzung beliebiger Inhaltsformate in eine Form, die vor der Veränderung durch das E-Mail-System geschützt ist.

Header-Felder (Header-Fields)

Alle neu in MIME [RFC2045] definierten Header-Felder entsprechen den generellen syntaktischen Regeln für Header-Felder wie in [RFC822] spezifiziert und können somit in einem normalen Nachrichten-Header nach [RFC822] vorkommen.

Folgend werden die fünf neuen Header-Felder beschrieben:

- **MIME-Version:** Dieses Feld muss den Wert 1.0 besitzen. Das Vorhandensein dieses Feldes zeigt an, dass die Nachricht mit den RFC's [RFC2045] und [RFC2046] übereinstimmt.
- **Content-Type (Inhaltstypen):** Content-Type spezifiziert die Daten, welche im Körper der Nachricht enthalten sind im Detail, so dass der Empfänger weiss, was er damit anzufangen hat. Somit kann der passende Agent oder Mechanismus zum Darstellen der Daten ausgewählt werden.
- **Content-Transfer-Encoding (Übertragungskodierung):** Dieses Feld bezeichnet die Art der Umwandlung, die verwendet wurde um den Körper der Nachricht in eine für die Übertragung brauchbare Form zu bringen.
- **Content-ID:** Dieser Wert wird zur eindeutigen Identifikation vom MIME-Einheiten in mehreren Kontexten verwendet.
- **Content-Description:** Eine verbale Beschreibung des Nachrichten-Körpers. Dieses Feld ist nützlich, wenn der Nachrichten-Körper ein nicht lesbares Objekt wie beispielsweise Audiodaten enthält.

Eine Implementation, die mit MIME kompatibel ist, muss die Felder MIME-Version, Content-Type und Content-Transfer-Encoding enthalten. Content-ID und Content-Description brauchen nicht vorzukommen und können vom Empfänger ignoriert werden.

MIME-Inhaltstypen (Content Type)

Das Feld Content-Type wird benutzt, um die Art der Daten (media type) im Nachrichten-Body der MIME-Einheit zu spezifizieren. Zu diesem Zweck wird das Feld Content-Type mit dem Wert eines Haupttypes und der genauen Unterart versehen. Die Kombination von Haupttyp und Unterart bezeichnet den genauen Media-Typ (media type). MIME definiert eine grosse Vielfalt von Inhaltstypen und ein grosser Teil des MIME-Standards betrifft die Spezifikation der Inhaltstypen. Die Vielfalt an Inhaltstypen entspricht der Anforderung, eine grosse Menge unterschiedlicher Dateiarten zu verarbeiten, wie sie in einer Multimedia-Umgebung vorkommen.

In [RFC2046] werden sieben Haupttypen mit insgesamt 15 Unterarten definiert. Mit dem Haupttyp wird dabei die allgemeine Form der Daten festgelegt und die Unterart beschreibt das genaue Datenformat. Tabelle 6 zeigt die Inhaltstypen, welche in [RFC2046] definiert sind.

Haupttyp	Unterart	Beschreibung
Text	Plain	Unformatierter Text, ASCII oder ISO8859
	Enriched	Bietet mehr Möglichkeiten der Formatierung
Multipart	Mixed	Die verschiedenen Teile sind unabhängig voneinander, werden aber zusammen übertragen. Sie sollten beim Empfänger in der gleichen Reihenfolge ankommen, in der sie auch in der E-Mail erscheinen. Typ/Unterart jedes Teils ist standardmässig text/plain
	Parallel	Wie mixed, jedoch braucht keine Reihenfolge eingehalten zu werden. Die Reihenfolge der Teile ist nicht von Bedeutung.
	Alternativ	Die unterschiedlichen Teile stellen verschiedene Versionen derselben Information dar. Sie werden mit wachsender Originaltreue sortiert und das Mail-System des Empfängers sollte dem Benutzer die „beste“ Version anzeigen.
	Digest	Wie Mixed, aber Typ/Unterart jedes Teils ist standardmässig message/RFC 822
Message	RFC 822	Der Nachrichten-Körper schliesst eine RFC 822-Nachricht ein
	Partial	Erlaubt das Fragmentieren grosser Nachrichten auf eine Weise, die der Benutzer nachvollziehen kann.
	External-Body	Enthält einen Zeiger auf ein Objekt, das sich woanders befindet
Image	jpeg	Das Bild besitzt das Format JPEG mit der Kodierung JFIF
	gif	Das Bild hat das Format GIF
Video	mpeg	Video, welcher gemäss dem MPEG-Standard kodiert ist
Audio	Basic	Einkanal-8-Bit ISDN uLaw-Verschlüsselung (PCM) mit einer Sample-rate von 8 kHz
Application	PostScript	Adobe PostScript
	Octet-stream	Binäre Daten, die aus Bytes mit jeweils 8 Bit bestehen

Tabelle 6 MIME-Inhaltstypen nach [RFC2046]. Quelle: [STA01]

Zum Verständnis des Haupttyps **`Text`** wird keine weitere Software benötigt, ausser der Unterstützung des verwendeten Zeichensatzes. Die elementarste Unterart von Text heisst **`Plain`** und besteht aus einer Zeichenfolge von ASCII- oder ISO8859-Zeichen. Die Unterart **`Enriched`** gestattet zusätzlich einiges an Formatierungen.

Durch die Verwendung des Haupttyps **`Multipart`** wird angezeigt, dass der Nachrichten-Körper aus mehreren, unabhängigen Teilen besteht. Für diesen Zweck enthält das Feld Content-Type einen zusätzlichen Parameter mit dem Namen Boundary. Dieser Parameter legt die Abgrenzung zwischen den unabhängigen Teilen innerhalb des Nachrichten-Körpers fest und dient somit als Trennzeichen. Natürlich darf das festgelegte Trennzeichen in keinem weiteren Teil der Nachricht vorkommen. Jedes Trennzeichen beginnt auf einer neuen Zeile mit zwei Bindestrichen, gefolgt von dem eigentlichen Trennzeichen des Parameters Boundary. Dem letzten Teil im Nachrichten-Körper folgen ebenfalls zwei Bindestriche und das Trennzeichen mit zwei zusätzlichen Bindestrichen, welche anzeigen dass keine weiteren Nachrichten-Teile folgen. In jedem Teil einer Nachricht kann optional ein normaler MIME-Header vorkommen.

Hier ein einfaches Beispiel einer solchen Multitpart-Nachricht mit dem Untertyp **`Mixed`**. Das Beispiel besteht aus zwei Teilen, die jeweils normalen Text enthalten. (Das Beispiel wurde aus [RFC2046] entnommen und angepasst.):

```
From: "Loretz Manfred, loretman" <loretman@zhwin.ch>
To: "Gisler Andreas, gislean1" <gislean1@zhwin.ch>
CC: "Stricker Andreas, stricand" <stricand@zhwin.ch>
Date: Wed, 1 Oct 2003 09:01:54 +0200
Subject: Beispiel-Nachricht mit MIME
MIME-Version: 1.0
Content-type: multipart/mixed; boundary="simple boundary"

Dies ist die Präambel. Sie sollte ignoriert werden, bietet sich aber für
Erklärungen des Mailerzeugers an Benutzer ohne MIME an.

--simple boundary

Dies ist impliziter ASCII-Klartext. Er endet NICHT mit einem
Zeilenvorschub
--simple boundary
Content-type: text/plain; charset=us-ascii

Dies ist expliziter ASCII-Klartext. Er endet mit einem Zeilenvorschub.

--simple boundary--

Dies ist das Nachwort. Es kann ebenfalls ignoriert werden.
```

In [RFC2046] werden vier Unterarten des Multipart-Typs spezifiziert, welche alle die gleiche Syntax aufweisen. Weitere Unterarten des Multipart-Typs können aber in der Zukunft hinzugefügt werden oder in anderen Orten definiert werden, wie es in [RFC2633] geschehen ist.

Der Typ **Multipart/Mixed** wird verwendet, wenn mehrere, unabhängige Teile des Körpers existieren, die in einer bestimmten Sortierung gebündelt werden müssen. Bei der Unterart **Multipart/Parallel** existieren ebenfalls mehrere unabhängige Teile, allerdings spielt die Reihenfolge keine Rolle. Wenn es das System des Empfängers zulässt, können die Teile beim Empfänger parallel angezeigt werden. Ein Bild oder ein Text könnte beispielsweise während der Anzeige mit einer Stimme kommentiert werden.

Bei der Unterart **Multipart/Alternative** bestehen die verschiedenen Teile aus der gleichen Information, allerdings existieren die verschiedenen Teile in einer unterschiedlichen Darstellung. Hier ein ähnliches Beispiel wie in [RFC2046]:

```
From: "Loretz Manfred, loretman" <loretman@zhwin.ch>
To: "Gisler Andreas, gislean1" <gislean1@zhwin.ch>
CC: "Stricker Andreas, stricand" <stricand@zhwin.ch>
Date: Wed, 1 Oct 2003 09:01:54 +0200
Subject: Beispiel einer Nachricht des Typs Multipart/Alternative
MIME-Version: 1.0
Content-type: multipart/alternative; boundary="boundary42"

--boundary42

Content-Type: text/plain; charset=us-ascii

...hier steht die Klartext-Version der Nachricht...

--boundary42

Content-Type: text/enriched

...und die Version mit Formatierung folgt hier...

--boundary42--
```

In der Unterart **Multipart/Alternative** sind die Teile des Nachrichten-Körpers nach steigender Bevorzugung sortiert. Hat der Empfänger dieses Beispiels die Möglichkeit, die Nachricht im Format **Text/Enriched** anzuzeigen, nimmt der Empfänger diese Möglichkeit wahr. Sonst wird vom Empfänger die Nachricht im Format **Text/Plain** angezeigt.

Die Unterart **Multipart/Digest** wird benutzt, wenn jeder Teil des Nachrichten-Körpers eine Nachricht nach [RFC822] mit eigenem Nachrichten-Header darstellt. Mit dieser Unterart kann eine Nachricht erstellt werden, welche wiederum aus weiteren Nachrichten besteht. Als Beispiel könnte ein Moderator einer Gruppe E-Mails von Teilnehmern sammeln und dann gebündelt in einer einzigen MIME-Nachricht versenden.

Als nächster Typ bietet **Message** eine Reihe von wichtigen Möglichkeiten in MIME. **Message/RFC822** zeigt an, dass der Nachrichten-Körper aus einer kompletten Nachricht einschliesslich Header und Körper besteht. Trotz der Bezeichnung RFC822, kann die eingeschlossen Nachricht eine vollwertige MIME-Nachricht sein und muss nicht nur eine einfache Nachricht nach [RFC822] sein.

In der Unterart **Message/Partial** kann eine grosse Nachricht in eine Reihe kleinerer Nachrichten fragmentiert werden, die der Empfänger wieder zusammen setzen muss. Für die Unterart **Message/Partial** existieren drei Parameter im Feld Content-Type: Die ID, welche die Fragmente einer Nachricht bezeichnet, eine eindeutige Sequenznummer für jedes einzelne Fragment und die gesamte Anzahl von Fragmenten.

Message/External Body bezeichnet eine Nachricht, in welcher der Nachrichten-Körper keine zu übertragenden Daten enthält, sondern lediglich Informationen enthält, von wo man die zu übertragenden Daten erhält. Wie die anderen Unterarten von Message, besitzt auch diese Unterart neben dem äusseren Header eine eingeschlossene Nachricht mit einem eigenen Header. Im äusseren Header ist Content-Type das einzige erforderliche Feld, welches die Unterart und die Zugriffsart auf die Daten identifiziert, wie beispielsweise FTP oder eine lokale Datei. Der innere Header bezieht sich auf die von ihm eingeschlossene Nachricht.

Der Typ **Application** bezieht sich auf andere Arten von Daten, welche nicht in die eine andere Kategorie passen, wie typischerweise entweder nicht interpretierte Binärdaten oder Informationen. Diese Informationen müssen von einer Anwendung bearbeitet werden, bevor sie für einen Benutzer sichtbar sind oder von einem Benutzer weiterverwendet werden können.

MIME-Übertragungskodierung(Content-Transfer-Endcoding)

Die MIME-Übertragungskodierung für Nachrichten-Körper stellt die zweite, grosse Hauptkomponente der MIME-Spezifikation dar. Die Übertragungskodierung hat das Ziel, eine zuverlässige Übertragung der Daten in allen denkbaren Umgebungen zu gewährleisten.

Mechanismus	Beschreibung
7bit	Repräsentation der Daten durch kurze Zeilen (<= 998 Oktette) von ASCII-Zeichen.
8bit	Repräsentation der Daten durch kurze Zeilen (<= 998 Oktette), welche aber auch Nicht-ASCII-Zeichen enthalten können.
binary	Neben Nicht-ASCII-Zeichen können auch Zeilen auftreten, die für die Übertragung durch SMTP zu lang sind.
quoted-printable	Kodiert die Daten auf eine Weise, dass die im Text enthaltenen ASCII-Zeichen im Wesentlichen lesbar sind.
base64	Kodiert die Daten durch Abbildung von 6-Bit-Blöcken der Eingabe auf 8-Bit-Blöcke der Ausgabe, bei denen es sich um druckbare ASCII-Zeichen handelt.
X-Token	Eine benannte Kodierung, welche nicht im Standard enthalten ist.

Tabelle 7 MIME-Übertragungsverschlüsselung nach [RFC2045] Quelle: [STA01]

Das Header-Feld Content-Transfer-Encoding vermag die sechs Werte, welche in Tabelle 7 aufgeführt sind, anzunehmen. Der MIME-Standard [RFC2045] definiert die zwei Methoden **quoted-printable** und **base64** zur Datenkodierung. Die drei Werte **`7bit, 8bit`** und **`binary`** sagen aus, dass keine Kodierung vorliegt und liefern nur Informationen über die Art der Daten. Das Format **`7bit`** empfiehlt sich vor allem für den Gebrauch mit SMTP, während die Formate **`8bit`** und **`binary`** in anderen E-Mail-Übertragungen sinnvoll sein können. Der Wert **`X-Token`** wird gebraucht, wenn ein anderes Kodierungsschema verwendet wird, für das der Name angegeben werden muss. Hier sind hauptsächlich firmen- oder anwendungsspezifische Schema denkbar.

Handelt es sich hauptsächlich um Daten, die druckbare ASCII-Zeichen darstellen, ist die Übertragungsvkodierung **`quated-printable`** sehr nützlich. Im Wesentlichen geht es darum, die Zeichen durch die hexadezimale Repräsentation ihres Codes darzustellen und das Einfügen umkehrbarer (weicher) Zeilenvorschübe, um die Länge der Zeilen innerhalb der Nachricht auf 76 Zeichen zu begrenzen.

`base64`, auch als Radix-64 bekannt, wird gerne als Übertragungskodierung verwendet, wenn es sich hauptsächlich um binäre Daten handelt. Durch die Verwendung der Übertragungskodierung **`base64`**, werden die binären Daten unempfindlich gegenüber der Verarbeitung durch die E-Mail-Programme.

5.3.5.3 Kanonische Form

Das Konzept der kanonischen Form bildet ein wichtiges Konzept in MIME und S/MIME. Es handelt sich hierbei um ein Format, welches vom verwendeten Inhaltstyp abhängt und für die Benutzung zwischen unterschiedlichen System standardisiert wurde. Das Konzept der kanonischen Form steht im Gegensatz zur natürlichen Form, welche systemabhängig sein kann. Tabelle 8 aus [RFC2049] in [STA01] sollte dies verdeutlichen.

Natürliche Form	Der zu übertragende Nachrichten-Körper wird im natürlichen Format des Systems erstellt. Der natürliche Zeichensatz und gegebenenfalls die lokalen Zeilenvorschübe finden Anwendung. Der Körper kann aus einer UNIX-Textdatei, einem Sun-Rasterbild, einer VMS-Indexdatei, Audiodaten in einem systemabhängigen Format oder einem beliebigen anderen Format des jeweiligen lokalen Systems bestehen. Grundsätzlich werden Daten in der „natürlichen“ Form erzeugt, welche sich aus der Art des Mediums ergibt.
Kanonische Form	Der gesamte Körper einschliesslich solcher Informationen wie Satzlänge oder Dateiattribute wird in einer universellen kanonischen Form übersetzt. Sowohl die Art des Mediums des Körpers als auch seine Attribute geben die Natur der kanonischen Form vor. Die Konversion in die saubere, kanonische Form kann eine Konversion des Zeichensatzes, die Transformation von Audiodaten und viele andere, vom jeweiligen Medium abhängige Operationen beinhalten. Sofern jedoch eine Konversion des Zeichensatzes erforderlich ist, muss darauf geachtet werden, die Semantik des Mediums zu verstehen, die starke Implikationen für eine Zeichensatzkonvention haben kann (z.B. hinsichtlich der syntaktischen bedeutungsvollen Zeichen in einer anderen Textart als „Plaintext“).

Tabelle 8 Natürliche und kanonische Form nach [RFC2049]. Quelle: [STA01]

5.3.5.4 Beispiel einer Multipart-Nachricht

Die untenstehende Abbildung nach dem Beispiel aus [RFC1521] gibt einen Eindruck der Komplexität einer Multipart-Nachricht. Die untenstehende Nachricht besteht aus fünf Teilen, die nacheinander angezeigt werden. Zwei Teile mit Klartext führen die Nachricht ein, gefolgt von einer weiteren Multipart-Nachricht mit dem Untertyp **`Parallel`**, einen Teil vom Typ **`Text/Enriched`** und eine abgeschlossene Nachricht in einem Nicht-ASCII-Zeichensatz. Die eingefügte Multipart-Nachricht vom Untertyp **`Parallel`** besteht aus zwei Teilen, welche nebeneinander angezeigt werden können: ein Bild und ein Audio-Fragment.

```
From: "Loretz Manfred, loretman" <loretman@zhwin.ch>
To: "Gisler Andreas, gislean1" <gislean1@zhwin.ch>
CC: "Stricker Andreas, stricand" <stricand@zhwin.ch>
Date: Wed, 1 Oct 2003 09:01:54 +0200
Subject: Beispiel einer Nachricht einer Multipart-Nachricht
MIME-Version: 1.0

Content-Type: multipart/mixed; boundary=unique-boundary-1

Dies ist der Header einer Multipart-Nachricht, Mail-Reader, die dieses
Format verarbeiten können, sollten diesen Header ignorieren. Wenn Sie
diesen Text lesen, sollten Sie vielleicht einen Mail-Reader einsetzen,
der eine Multipart-Nachricht korrekt anzeigen kann.

--unique-boundary-1

...Hier erscheint irgend ein Text...

[Beachten Sie, dass die vorhergehende Leerzeile bedeutet, dass keine
Kopffelder auftreten und dies Text mit dem Zeichensatz US ASCII ist, was
man auch ausdrücklich hätte schreiben können, wie es im nächsten Teil
geschieht.]

--unique-boundary-1
Content-type: text/plain; charset=US-ASCII

Dies könnte ein Teil des vorherigen Teils sein, soll aber den
Unterschied zwischen expliziten und impliziten Texten anzeigen.

--unique-boundary-1
Content-Type: multipart/parallel; boundary=unique-boundary-2

-unique-boundary-2
Content-Type: audio/basic
Content-Transfer-Encoding: base64

...Hier folgen base64-verschlüsselte Audiodaten im Format 8000 Hz
einkanal mu-law...

--unique-boundary-2
Content-Type: image/gif
Content-Transfer-Encoding: base64

...Hier folgen base64-verschlüsselte Bilddaten...

--unique-boundary-2--

--unique-boundary-1
Content-type: text/enriched

Dies ist <bold><italic>Richtext,</italic></bold><smaller>wie er im RFC
1896 definiert wird </smaller><nl><nl>Ist das nicht<bigger><bigger>-
cool?</bigger></bigger>

--unique-boundary-1
Content-Type: message/rfc822

From: (Mailbox in US-ASCII)
To: (Adresse in US-ASCII)
Subject: (Betreff in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

...Hier folgt zusätzlicher Text in ISO-8859-1...

--unique-boundary-1--
```

5.3.6 PKCS#7 und Cryptographic Message Syntax (CMS)

PCKS#7 ist ein Standard der in [PKCS#7] und im Dokument [RFC2315] beschrieben wird. PKCS steht für „Public-Key Cryptography Standard“. Die Public-Key Cryptography Standards sind von der RSA produzierte Spezifikationen, welche die Absicht haben, den Einsatz von Public-Key Kryptographie zu beschleunigen. Der PKCS #7 hat wegen seiner Wichtigkeit für viele RFC's der IETF auch seinen Platz als RFC erhalten.

Die Cryptographic Message Syntax (CMS) wird im Dokument [RFC3369] beschrieben. Diese Syntax wird verwendet um einen beliebigen Inhalt einer Nachricht zu signieren, hashen, authentifizieren oder zu verschlüsseln.

CMS ist abgeleitet vom PCKS #7 Version 1.5 wie er in [RFC2315] beschrieben ist. Wo immer möglich, wurde versucht die Rückwärtskompatibilität zu PKCS #7 zu gewährleisten. Dennoch waren Änderungen notwendig, um zum Beispiel Schlüssel auszuhandeln und die Verschlüsselung von symmetrischen Schlüsseln für das Key-Management aufzunehmen.

CMS beschreibt eine Syntax für die Verkapselung von zu schützenden Daten, auf die kryptographische Merkmale, wie digitale Signaturen und Verschlüsselung angewendet werden. Die Syntax erlaubt eine mehrfache Verkapselung und somit Rekursivität, so dass zum Beispiel ein Umschlag in einen weiteren Umschlag verschachtelt werden kann oder ein Beteiligter eine Unterschrift unter einen Umschlag setzen kann, welcher die eigentlichen digitalen Daten enthält. Weiter erlaubt die Syntax auch frei wählbare Attribute, wie zum Beispiel den Zeitpunkt der durchgeführten Signatur, welche zusammen mit dem Inhalt der Nachricht authentifiziert werden können und bietet für andere Attribute die Möglichkeit, mit einer Signatur verknüpft zu sein.

Die CMS-Werte werden mit Hilfe von ASN.1 beschrieben und mittels BER kodiert. Die Werte werden typischerweise als eine Zeichenkette (octet strings) dargestellt. Während viele Systeme geeignet sind, um beliebige Zeichenketten zuverlässig zu übertragen, ist bekannt das manche Systeme für die elektronische Post nicht dafür geeignet sind. Das Dokument [RFC3369] beschreibt keine Mechanismen für die Kodierung von diesen Zeichenketten um einen zuverlässigen Transport in solchen Systemen zu gewährleisten.

5.3.6.1 Allgemeiner Überblick

Der CMS ist allgemein genug um viele unterschiedliche Inhaltsypen[i] zu unterstützen. Das Dokument [RFC3369] definiert mit dem ContentInfo einen geschützten Inhalt[ii]. ContentInfo kapselt einen einzelnen identifizierten Inhaltstyp und der identifizierte Typ kann weitere Kapselung bieten. Das Dokument [RFC3369] definiert sechs Inhaltstypen: data, signed-data, enveloped-data, digested-data, encrypted-data und authenticated-data. Zusätzlich können weitere Inhaltstypen ausserhalb des Dokumentes [RFC3369] definiert werden. In der aktuellen Version 3 von S/MIME werden nur die Inhaltstypen data, signed-data und enveloped-data gebraucht. Aus diesem Grund beschränken wir unsere genaueren Betrachtungen auf diese Inhaltstypen.

Eine Implementation welche die Spezifikation [RFC3369] erfüllen will, muss den protection content, ContentInfo und die Inhaltstypen data, signed-data und enveloped-data implementieren. Die anderen Inhaltstypen können je nach Bedarf implementiert sein.

5.3.6.2 Generelle Syntax

Folgender OBJECT IDENTIFIER identifiziert den Typ content information (ContentInfo):

i Englisch: content types
ii Englisch:protection content

```
id-ct-contentInfo OBJECT IDENTIFIER ::= {
          iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
smime(16) ct(1) 6
}
```

ASN.1-Typ ContentInfo wird also durch den OBJECT IDENTIFIER id-ct-contentInfo gekennzeichnet. Wie erwähnt ist ein OBJECT IDENTIFIER eine weltweit eindeutige Zeichenkette von ganzen Zahlen. Mit diesem Typ ContentInfo verbindet nun die Cryptographic Message Syntax einen OBJECT IDENTIFIER für einen Inhaltstyp mit dem dazugehörigen Inhalt. Der ASN.1-Typ ContentInfo besitzt dafür folgende Syntax:

```
ContentInfo ::= SEQUENCE {
          contentType ContentType,
          content [0] EXPLICIT ANY DEFINED BY contentType
}

ContentType ::= OBJECT IDENTIFIER
```

Die Felder von ContentInfo haben die folgende Bedeutung:

contentType

Bezeichnet den Typ des zugeordneten Inhaltes. Dabei ist contentType ein weiterer OBJECT IDENTIFIER, welche von der Behörde bestimmt wird, welche auch den Inhaltstyp definiert.

content

Ist der mit dem contentType verknüpfte Inhalt. Der Typ des Inhaltes kann eindeutig mit dem contentType bestimmt werden. In [RFC3369] sind die Inhaltstypen für data, signed-data, enveloped-data, digested-data, encrypted-data und authenticated-data definiert. Es können beliebige weitere Inhaltstypen ausserhalb von [RFC3369] definiert werden, allerdings gilt dafür die Beschränkung, dass für den definierten Inhaltstyp nicht der ASN.1 Typ CHOICE verwendet werden darf.

5.3.6.3 Data Content Type

Der Inhaltstyp data content wird durch den untenstehenden OBJECT IDENTIFIER bezeichnet:

```
id-data OBJECT IDENTIFIER ::= {
          iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7)
1
}
```

Der Inhaltstyp data content ist bestimmt für beliebige Zeichenketten (octet strings), wie zum Beispiel bei ASCII-Textdateien: Die Interpretation des Inhaltes wird der weiterverarbeitenden Anwendung überlassen. Der Inhaltstyp data content sollte den ASN.1 Typ Data haben:

```
Data ::= OCTET STRING
```

S/MIME benutzt diesen OBJECT IDENTIFIER id-data um MIME kodierten Inhalt zu kennzeichnen. Der Gebrauch dieses Identifiers ist in [RFC2633] definiert. Der Inhaltstyp data wird hauptsächlich gebraucht, um ihn in weiteren Inhaltstypen zu verschachteln.

5.3.6.4 Signed-data Content Type

Der Inhaltstyp signed-data besteht aus dem eigentlichen Inhalt eines beliebigen Typs und Null oder mehreren Signaturen. Eine beliebige Anzahl von Unterzeichnern kann jeden Inhalt eines beliebigen Types parallel unterzeichnen.

Die typische Anwendung von signed-data ist allerdings meistens die Unterschrift eines einzigen Unterzeichners auf den Inhaltstyp data. Ein weitere Einsatzmöglichkeit dieses Inhaltstyps besteht im Verbreiten von Zertifikaten und Certificate Revocation Lists (CRL).

Der Vorgang um einen Inhaltstyp signed-data zu erstellen enthält die folgenden Schritte:

1. Für jeden Unterzeichner wird über den zu unterzeichnenden Inhalt ein Message Digest oder Hash-Wert mit dem gewünschten Message-Digest-Algorithmus berechnet. Zusätzlich können weitere nützliche Informationen in die Berechnung eines Message Digest mit einbezogen werden. Dazu wird ein Message Digest über diese zusätzlichen Informationen, zusammen mit dem Message Digest der Nachricht berechnet, das Resultat dieser zweiten Berechnung bildet den endgültigen Message Digest. Um diesen Message Digest zu berechnen, wird ebenfalls der gewünschte Algorithmus des Unterzeichners verwendet.
2. Der Message Digest wird von jedem Unterzeichner mit seinem privaten Schlüssel unterschrieben und somit also seine digitale Signatur für den zu unterzeichnenden Inhalt erzeugt.
3. Für jeden Unterzeichner wird die Signatur und weitere spezifische Informationen zu einem SignerInfo zusammengefasst. Dieser Vorgang wird im Abschnitt 5.3.6.4 SignerInfo noch beschrieben. In diesem Schritt werden möglicherweise Zertifikate und CRL's für jeden Unterzeichner gesammelt.
4. Im SignedData werden alle verschiedenen Message Digest-Algorithmen der Unterzeichner, die SignerInfo von den einzelnen Unterzeichnern und der signierte Inhalt zusammengefasst. Dieser Prozess wird im folgenden Abschnitt 5.3.6.4 SignedData Type erläutert.

Ein Empfänger berechnet selbstständig und unabhängig einen eigenen Message Digest über den empfangenen Inhalt. Dieses Resultat wird zusammen mit dem öffentlichen Schlüssel des Unterzeichners zur Überprüfung der Unterschrift verwendet. Für die Überprüfung der Unterschrift wird die empfangene Signatur mit dem öffentlichen Schlüssel des Unterzeichners entschlüsselt und mit dem Resultat aus der eigenen Berechnung des Message Digest verglichen. Weisen der entschlüsselte Message Digest und der berechnete Message Digest den gleichen Wert auf, ist die Unterschrift gültig und die Herkunft erfolgreich überprüft. Es gibt zwei Varianten um für die Überprüfung der Signatur an den öffentlichen Schlüssel des Senders zu kommen. In Variante eins wird der öffentliche Schlüssel des Unterzeichners mit dem kennzeichnenden Namen des Herausgebers (distinguished name) zusammen mit einer benutzerspezifischen Seriennummer referenziert. Als zweite Variante kann über einen subject key identifier das Zertifikat bestimmt werden, welches den öffentlichen Schlüssel des Senders enthält. Das Zertifikat des Unterzeichners kann für diesen Zweck im Feld certificate von SignedData enthalten sein.

Dieses Unterkapitel ist in mehrere Abschnitte aufgeteilt. Der erste Teil beschreibt den Top-Level Typ SignedData, der zweite Abschnitt behandelt EncapsulatedContentInfo und im dritten Teil wird die für jeden Unterzeichner erstellte SignerInfo beschrieben.

SignedData Type

Der Inhaltstyp Signed-Data wird mit folgendem OBJECT IDENTIFIER erkannt:

```
id-signedData OBJECT IDENTIFIER ::= {
          iso(1)   member-body(2)   us(840)   rsadsi(113549)   pkcs(1)
          pkcs7(7) 2
}
```

Der Inhaltstyp signed-data sollte vom ASN.1 Typ SignedData sein, welcher wie folgt deklariert ist:

```
SignedData ::= SEQUENCE {
          version CMSVersion,
          digestAlgorithms DigestAlgorithmIdentifiers,
          encapContentInfo EncapsulatedContentInfo,
          certificates [0] IMPLICIT CertificateSet OPTIONAL,
          crls [1] IMPLICIT CertificateRevocationLists OPTIONAL,
          signerInfos SignerInfos
}

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo
```

Die Felder des Typs SignedData haben die folgende Bedeutung:

version:

Damit wird die Versionsnummer des verwendeten Syntax gekennzeichnet. Der dazugehörige Wert hängt von den Zertifikaten, dem eContentType und SignerInfo ab.

digestAlgorithms:

Wie es der Name DigestAlgorithms schon sagt, handelt es sich hierbei um eine Sammlung von Kennungen für Message Digest-Algorithmen. Es kann keine bis zu mehreren Elementen in der Sammlung haben. Jedes Element identifiziert einen Message Digest-Algorithmus wie er von mindestens einem Unterzeichner benutzt wurde, mit einer beliebigen Anzahl von dazugehörigen Parametern.

encapContentInfo:

Dieses Feld encapContentInfo ist vom Typ EncapsulatedContentInfo und wird in Abschnitt 5.3.6.4 EncapsulatedContentInfo Type genauer erläutert. Kurz gesagt enthält dieses Feld vom Typ EncapsulatedContentInfo den unterzeichneten Inhalt und einen OBJECT IDENTIFIER, welcher den Typ des unterzeichneten Inhalt deklariert.

certificates:

Dies ist eine optionale Sammlung von Zertifikaten. Somit kann diese Sammlung vorhanden sein oder auch nicht. Diese Feld kann alle Zertifikate enthalten, welche nötig sind um eine Kette von einem Zertifikat eines Unterzeichners zu einem Root- oder Top-Level-Zertifikat herzustellen. Hier können auch die Zertifikate der einzelnen Unterzeichner enthalten sein, damit zur Überprüfung der Unterschrift alle nötigen Zertifikate beim Empfänger vorhanden sind.

crls:

Crls ist eine weitere optionale Sammlung. Diese Sammlung enthält Certificate Revocation Lists (CRLs). Dieses Feld ist gedacht um zusätzliche Informationen zu liefern, damit der Empfänger überprüfen kann, ob die mitgesendeten Zertifikate gültig sind oder auch nicht.

signerInfos:

Dieses Feld enthält ein Set von SignerInfos und somit enthält dieses Set die jeweiligen Informationen zu den einzelnen Unterzeichnern. Der Typ SignerInfo wird im Abschnitt 5.3.6.4 SignerInfo genauer erklärt.

EncapsulatedContentInfo Type

Der Inhalt wird im Typ EncapsulatedContentInfo repräsentiert:

```
EncapsulatedContentInfo ::= SEQUENCE {
          eContentType ContentType,
          eContent [0] EXPLICIT OCTET STRING OPTIONAL
}

ContentType ::= OBJECT IDENTIFIER
```

Die Felder des Typs EncapsulatedContentInfo haben die folgende Bedeutung:

eContentType:

Ein OBJECT IDENTIFIER. Der OBJECT IDENTIFIER kennzeichnet den Typ des Inhaltes in eContent.

eContent:

Stellen die Daten als eine Kette von Oktetten (octet string) dar.

Optional kann der eContent innerhalb vom EncapsulatedContentInfo wegfallen, dadurch wird es möglich, so genannte externe Signaturen zu erstellen. Im Fall von externen Signaturen wird die Signatur über einen Inhalt berechnet, der Inhalt wird aber nicht im EncapsulatedContentInfo eingefügt. Ein eContentType wird zugewiesen, als wäre der Inhalt vorhanden.

SignerInfo

Die Informationen für jeden einzelnen Unterzeichner wird im Typ SignerInfo repräsentiert:

```
SignerInfo ::= SEQUENCE {
          version CMSVersion,
          sid SignerIdentifier,
          digestAlgorithm DigestAlgorithmIdentifier,
          signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
          signatureAlgorithm SignatureAlgorithmIdentifier,
          signature SignatureValue,
          unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL
}

SignerIdentifier ::= CHOICE {
          issuerAndSerialNumber IssuerAndSerialNumber,
          subjectKeyIdentifier [0] SubjectKeyIdentifier
}

SignedAttributes ::= SET SIZE (1..MAX) OF Attribute

UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute

Attribute ::= SEQUENCE {
          attrType OBJECT IDENTIFIER,
          attrValues SET OF AttributeValue
}

AttributeValue ::= ANY

SignatureValue ::= OCTET STRING
```

Kurz erklärt besitzen die Felder vom Typ SignerInfo folgende Bedeutungen:

version

Bezeichnet die Versionsnummer des Syntax. Diese Wert ist abhängig von der Wahl des Wertes beim Typ SignerIdentifier.

sid

Bestimmt das verwendete Zertifikat des Unterzeichners und somit auch den verwendeten, öffentlichen Schlüssel. Der öffentliche Schlüssel wird vom Empfänger benötigt, um die Signatur zu überprüfen. SignerIdentifier bietet zwei Alternativen, um den öffentlichen Schlüssel des Unterzeichners zu bestimmen. Die Alternative issuerAndSerialNumber identifiziert das Zertifikat des Unterzeichners mit Hilfe des kennzeichnenden Namen für den Herausgeber und der eindeutigen Seriennummer des entsprechenden Zertifikates. Als zweite Variante identifiziert der subjectKeyIdentifier das entsprechende Zertifikat. Empfänger müssen beide Varianten unterstützen.

digestAlgorithm

Dieses Feld identifiziert den Message Digest-Algorithmus und die damit verbundenen Parameter. Der in diesem Feld identifizierte Algorithmus soll ebenfalls im Feld digestAlgorithms des damit verbundenen SignedData enthalten sein.

signedAttrs

Enthält eine Liste der Attribute, welche in die Berechnung der Signatur miteinbezogen wurden. Das Feld ist optional, muss allerdings vorhanden sein, falls der eContentType nicht vom Inhaltstyp data ist. An dieser Stelle ist wichtig zu bemerken, das SignedAttributes DER codiert werden müssen, auch wenn der Rest von der Struktur BER codiert ist.

signatureAlgorithm

Dieses Feld identifiziert den Algorithmus und die damit verbundenen Parameter, um die digitale Unterschrift zu erzeugen.

signature

Dieses Feld enthält das Resultat von der Berechnung der digitalen Unterschrift mit dem verwendeten Message Digest-Algoritmus und dem privaten Schlüssel. Die Details der Signatur sind abhängig vom angewendeten Algorithmus.

unsignedAttrs

unsignedAttrs enthält eine Liste der Attribute, welche nicht in die Berechnung der Signatur einbezogen wurden und stellt somit das Gegenstück zum Feld signedAttrs dar. Wie das Feld signedAttrs, ist auch dieses Feld optional.

Die Felder des Typs SignedAttribute und des Typs UnsignedAttribute haben nachstehende Bedeutung:

attrType

Das Feld attrType definiert den Typ des Attributes und wird durch ein OBJECT IDENTIFIER bezeichnet.

attrValues

attrValues ist ein Set, welches die Werte für die Attribute des Feldes attrType enthält. Den Typ eines jeden Wertes im Set kann eindeutig durch attrType bestimmt werden. AttrType kann Beschränkungen bezüglich der Anzahl Werte in einem Set enthalten.

5.3.6.5 Enveloped-data Content Type

Der Inhaltstyp enveloped-data besteht aus einem verschlüsselten Inhalt eines beliebigen Typs und dem einmal oder auch mehrmals verschlüsselten, symmetrischen Schlüssel, welcher vollkommen geheim sein muss. Für jeden Empfänger der Nachricht muss der symmetrische Schlüssel für die Entschlüsselung des Inhaltes mit dem öffentlichen Schlüssel des Empfängers verschlüsselt werden. Die Kombination aus dem verschlüsselten Inhalt und dem chiffrierten, symmetrischen Schlüssel für einen entsprechenden Empfänger wird auch 'digital envelope' genannt. Jeder Inhaltstyp kann im Typ enveloped-data für eine beliebige Anzahl von Empfängern umhüllt werden.

Die typische Anwendung für den Inhaltstyp enveloped-data besteht meistens darin, einen Inhalt vom Typ data oder signed-data in einem enveloped-data umhüllt an einen oder auch mehrere Empfänger zu senden.

Ein Objekt vom Typ enveloped-data entsteht durch die Anwendung der folgenden Schritte:

1. Der erste Schritt besteht darin, dass zur Verschlüsselung des Inhaltes, abhängig vom verwendeten Algorithmus ein zufälliger, symmetrischer Schlüssel erzeugt wird.
2. Der erzeugte, symmetrische Schlüssel wird für jeden Empfänger der Nachricht verschlüsselt. Die Details dieses Vorganges sind vom verwendeten Algorithmus für das Schlüsselmanagement abhängig. Folgende vier Techniken werden unterstützt:
   - Schlüsseltransport
     Der symmetrische Schlüssel wird mit dem öffentlichen Schlüssel des Empfängers verschlüsselt und zum Empfänger übertragen.
   - Vereinbarung eines Schlüssels
     Der öffentliche Schlüssel des Empfängers und der private Schlüssel des Senders werden verwendet um einen paarweisen, symmetrischen Schlüssel zu erzeugen.
   - Symmetrische Verschlüsselung
     Der für die Chiffrierung des Inhaltes verwendete symmetrische Schlüssel wird selber mit einem schon im Voraus verteilten, symmetrischen Schlüssel chiffriert.
   - Passwort
     Der symmetrische Schlüssel wird mit einem Schlüssel verschlüsselt, welcher von einem Passwort oder von einem anderen gemeinsamen Geheimnis abgeleitet wurde.
3. Für jeden Empfänger wird der verschlüsselte, symmetrische Schlüssel und weitere wichtige Informationen in einem RecipientInfo zusammengefasst.
4. Der eigentliche Inhalt wird mit dem symmetrischen Schlüssel verschlüsselt. Möglicherweise muss der Inhalt auf eine bestimmte Blockgrösse aufgefüllt werden.

5. Die RecipientInfo aller Empfänger werden mit dem verschlüsselten Inhalt zusammengefasst, um daraus einen EnvelopedData zu erstellen.

Um nun diesen digitalen Umschlag zu öffnen, muss es einem Empfänger möglich sein, den symmetrischen Schlüssel zurück zu gewinnen, um damit den verschlüsselten Inhalt wiederherzustellen.

Dieses Unterkapitel ist in mehrere Abschnitte aufgeteilt. Der erste Teil in Abschnitt 5.3.6.5 EnvelopedData Type beschreibt den Top-Level Typ EnvelopedData, der zweite Abschnitt 5.3.6.5 RecipientInfo Type behandelt den für jeden Empfänger vorhandenen RecipientInfo.

EnvelopedData Type

Durch unten stehenden OBJECT IDENTIFIER wird der Inhaltstyp enveloped-data gekennzeichnet:

```
id-envelopedData OBJECT IDENTIFIER ::= {
          iso(1)   member-body(2)   us(840)   rsadsi(113549)   pkcs(1)
          pkcs7(7) 3
}
```

Der Inhaltstyp enveloped-data sollte vom ASN.1 Typ EnvelopedData sein:

```
EnvelopedData ::= SEQUENCE {
          version CMSVersion,
          originatorInfo [0] IMPLICIT OriginatorInfo OPTIONAL,
          recipientInfos RecipientInfos,
          encryptedContentInfo EncryptedContentInfo,
          unprotectedAttrs    [1]    IMPLICIT    UnprotectedAttributes
          OPTIONAL
}

OriginatorInfo ::= SEQUENCE {
          certs [0] IMPLICIT CertificateSet OPTIONAL,
          crls [1] IMPLICIT CertificateRevocationLists OPTIONAL
}

RecipientInfos ::= SET SIZE (1..MAX) OF RecipientInfo

EncryptedContentInfo ::= SEQUENCE {
          contentType ContentType,
          contentEncryptionAlgorithm
ContentEncryptionAlgorithmIdentifier,
          encryptedContent [0] IMPLICIT EncryptedContent OPTIONAL
}

EncryptedContent ::= OCTET STRING
UnprotectedAttributes ::= SET SIZE (1..MAX) OF Attribute
```

Die Felder des ASN.1 Typs EnvelopedData haben die folgende Bedeutung:

`version`

Das Feld version enthält die Seriennummer des verwendeten Syntax. Der dazugehörige Wert ist von originatorInfo, recipientInfo und unprotectedAttrs abhängig.

`originatorInfo`

Bietet optionale Informationen über den Absender. Dieses Feld ist nur vorhanden, falls es vom verwendeten Algorithmus für das Keymanagement benötigt wird. Es kann Zertifikate und CRL's enthalten:

`certs`

Eine optionale Sammlung von Zertifikaten. Hier kann das Zertifikat des Absenders mitgeführt werden. Weiter kann dieses Feld alle nötigen Zertifikate mitsenden, damit eine Verbindung vom Zertifikat des Absenders bis zu seinem Root-Zertifikat hergestellt werden kann. Erst durch diese Verbindung und einem vertrauenswürdigen Root-Zertifikat kann auch dem Zertifikat des Senders vertraut werden. Vertrauen bedeutet, dass man sich sicher sein kann, dass der im Zertifikat enthaltene öffentliche Schlüssel auch zum Absender gehört.

crls

Das Feld crls ist eine weitere optionale Sammlung. Diese Sammlung enthält Certificate Revocation Lists (CRLs), welche zusätzliche Informationen liefern, damit der Empfänger überprüfen kann, ob die mitgesendeten Zertifikate gültig sind oder nicht.

recipientInfos

Die Sammlung recipientInfos enthält für jeden Empfänger ein RecipientInfo. Es muss mindestens ein RecipientInfo vorhanden sein.

encryptedContentInfo

Informationen für den verschlüsselten Inhalt.

unprotectedAttrs

Dieses Feld ist eine optionale Sammlung, welche alle Attribute enthält, die nicht verschlüsselt sind.

Kurz erklärt, haben die Felder vom Typ EnryptedContentInfo folgende Bedeutungen:

contentType

Gibt mit einem OBJECT IDENTIFIER den Typ des Inhaltes an.

contentEncryptionAlgoritm

Bezeichnet den Algorithmus und die damit verbundenen Parameter, welche zur Verschlüsselung des Inhalt verwendet wurden. Für alle Empfänger der Nachricht wird der gleiche Algorithmus und gleiche, symmetrische Schlüssel verwendet.

encryptedContent

Dieses optionale Feld enthält das Ergebnis, welches durch die Verschlüsselung des Inhaltes entstanden ist. Falls das Feld nicht vorhanden ist, muss der für das Feld vorgesehene Wert durch andere Hilfsmittel geliefert werden.

RecipientInfo Type

Die Informationen für jeden einzelnen Empfänger sind im Typ RecipientInfo enthalten. Dieser Typ besitzt für jede Technik des Schlüsselmanagement ein anderes Format. Für jeden Empfänger kann eine beliebige, unterstützte Technik für das Schlüsselmanagement benützt werden. In allen Fällen, wird der verschlüsselte, symmetrische Schlüssel zu einem oder mehreren Empfängern transportiert.

Eine Implementation muss für das Schlüsselmanagement folgende Techniken unterstützen: Den Transport eines Schlüssels, die Vereinbarung eines Schlüssels und die symmetrische Chiffrierung mittels eines im Voraus verteilten symmetrischen Schlüssels. Diese Techniken werden durch *ktri*, *kari* und *kekri* bezeichnet. Die Technik für das Schlüsselmanagement mittels eines geheimen Passwortes kann von einer Implementation unterstützt werden und wird durch *pwri* bezeichnet. Eine Implementation kann auch noch andere Techniken unterstützen, welche dann durch *ori* gekennzeichnet sind.

```
RecipientInfo ::= CHOICE {
          ktri KeyTransRecipientInfo,
          kari [1] KeyAgreeRecipientInfo,
          kekri [2] KEKRecipientInfo,
          pwri [3] PasswordRecipientinfo,
          ori [4] OtherRecipientInfo
}

EncryptedKey ::= OCTET STRING
```

Wie erwähnt besitzt dieser Typ RecipientInfo für jede Technik eine anderes Format, wir gehen aber an dieser Stelle nicht näher auf diese unterschiedlichen Formate ein und verweisen dafür auf den entsprechenden Standard in [RFC3369].

Die einzige Gemeinsamkeit, die fast alle Formate besitzen, ist die des symmetrischen Schlüssel für die Chiffrierung des Inhaltes. Dieser Schlüssel wird im Typ EncryptedKey hinterlegt und ist vom Typ OCTET STRING abgeleitet.

5.3.7 S/MIME

5.3.7.1 Einführung

S/MIME (Secure/Multipurpose Internet Mail Extensions) in [RFC2633] bietet einen konsequenten Weg, um gesicherte MIME-Daten zu senden und zu empfangen. Basierend auf dem verbreiteten Internet-MIME-Standard bietet S/MIME den Anwendungen für einen elektronischen Datentransfer die folgenden kryptographischen Sicherheitsmechanismen an: Authentifizierung, Integrität von Nachrichten und die Unleugbarkeit des Ursprungs mittels digitalen Unterschriften, sowie Datenschutz und Datensicherheit durch Verschlüsselung.

S/MIME kann bei traditionellem E-Mail benützt werden, um kryptographische Sicherheitsmechanismen vor dem Versenden hinzuzufügen, welche anschliessend beim Empfänger wieder interpretiert und entfernt werden. Der Einsatz von S/MIME ist aber nicht nur auf E-Mail beschränkt. S/MIME kann mit allen Transport-Systemen benutzt werden, welche MIME unterstützen, wie zum Beispiel HTTP und SIP. S/MIME nutzt dabei die Vorteile der objektbasierten Merkmale von MIME und erlaubt damit einen Austausch von gesicherten Nachrichten in gemischten Transport-Systemen.

5.3.7.2 Überblick über RFC2633

S/MIME [RFC2633] beschreibt ein Protokoll für kryptographische Signaturen und für die Verschlüsselungen von MIME-Daten. Der MIME-Standard in [RFC2045] bis [RFC2049] bietet eine generelle Struktur für den Gebrauch mit Internet-Nachrichten und erlaubt Erweiterungen für neue Inhaltstypen für zusätzliche Anwendungen.

S/MIME beschreibt wie man den Körper einer MIME-Nachricht erstellt, welcher gemäss des Cryptographic Message Syntax (CMS) [RFC3369] mit kryptographischen Merkmalen erweitert wurde. Ebenfalls definiert [RFC2633] den MIME-Inhaltstyp **`application/pkcs7-mime`,** welcher benutzt werden kann, um diese Nachrichten-Körper zu transportieren.

In [RFC2633] wird aufgezeigt, wie der in [RFC1847] definierte Inhaltstyp **`multipart/signed`** verwendet werden muss, um S/MIME signierte Nachrichten zu transportieren. [RFC2633] definiert den neuen MIME-Inhaltstyp **`application/pkcs7-signature`** für signierte Nachrichten und erläutert ebenfalls wie dieser Inhaltstypen gebraucht werden kann, um signierte S/MIME-Nachrichten zu befördern.

Um S/MIME-Nachrichten zu erstellen, muss ein S/MIME-Agent nicht nur den Spezifikationen in [RFC2633] folgen, sondern auch den Spezifikationen in [RFC3369] entsprechen.

Durch den ganzen Standard werden Anforderungen und Empfehlungen gemacht, wie ein Empfänger ankommende Nachrichten verarbeiten muss. Zusätzlich werden eigene Anforderungen und Empfehlungen für die Sender spezifiziert, welche eine Nachricht erstellen und versenden. Im Allgemeinen liegt die beste Strategie darin, liberal zu sein, in dem was man empfängt und konservativ in dem was man versendet. Die meisten Anforderungen werden bezüglich dem Empfang von Nachrichten gemacht, während die Empfehlungen sich mit dem Erstellen und Versenden von Nachrichten beschäftigen.

Die Unterscheidung der Anforderungen zwischen Empfänger und Sender stammt von der Wahrscheinlichkeit, dass es S/MIME-Systeme geben kann, welche Software einbeziehen, die sich von traditionellen Internet-Mail-Agenten unterscheiden. S/MIME kann mit jedem System benützt werden, welches auch MIME transportieren kann. Zum Beispiel, ein automatisierter Prozess, welcher verschlüsselte Nachrichten versendet, muss nicht unbedingt in der Lage sein, eine verschlüsselte Nachricht zu empfangen. Somit werden, falls erforderlich, die Anforderungen und Empfehlungen für die zwei Arten von Agenten separat aufgelistet.

5.3.7.3 S/MIME-Funktionalität

CMS [RFC3369] definiert mehrere Inhaltstypen. Von diesen Inhaltstypen werden gegenwärtig nur die Inhaltstypen Data, SignedData und EnvelopedData für S/MIME verwendet. Ebenfalls wird in S/MIME der Gebrauch vom Inhaltstyp multipart/signed erläutert, welcher schon in [RFC1847] definiert ist. Die

Inhaltstypen Data, SignedData und EnvelopedData werden schon in 5.3.6 PKCS#7 und Cryptographic Message Syntax (CMS) ausführlich beschrieben, trotzdem werden diese Typen nochmals kurz erläutert und dargestellt.

Data Content Type

Der Inhaltstyp Data ist bestimmt für willkürlichen Zeichenketten (octet strings), wie zum Beispiel ASCII-Textdateien. Die Interpretation des Inhaltes ist der Anwendung überlassen. Solche Zeichenketten müssen keine interne Struktur besitzen, sie können es aber. Dieser Inhaltstyp wird genauer in Abschnitt 5.3.6.3 Data Content Type beschrieben.

Allgemein wird der Inhaltstyp data in die Inhaltstypen signedData und envelopedData verschachtelt.

SignedData Content Type

Indem man den Message Digest, des zu unterschreibenden Inhaltes, mit dem privaten Schlüssel des Unterzeichners verschlüsselt, wird eine digitale Signatur erzeugt. Der unterschriebene Inhalt plus die Signatur werden anschliessend mit der base64-Kodierung codiert. Die unterschriebene Nachricht kann nur von einem Empfänger betrachtet werden, der auch über S/MIME verfügt. Die genaue Beschreibung dieses Inhalttyps steht in Abschnitt 5.3.6.4 Signed-data Content Type.

Der Typ signedData wird auch im Zusammenhang mit **`application/pkcs7-signature`** verwendet. **`application/pkcs7-signature`** hat die Spezialität, dass der Inhalt fehlt und nur die Signatur vorhanden ist.

EnvelopedData Content Type

Dieser Inhaltstyp besteht aus dem verschlüsselten Inhalt beliebigen Typs, mit den chiffrierten Schlüsseln für einen oder mehrere Empfänger. Dieser Inhaltstyp wird ausführlicher in Abschnitt 5.3.6.5 Enveloped-data Content Type beschrieben.

Multipart/signed Content Type (Clear Signed Data)

Wird der Inhaltstyp **`multipart/signed`** verwendet, spricht man auch von einer Clear Signed-Nachricht oder auch Clear Signed Data. Auch hier wird eine digitale Signatur des Inhalts gebildet. Im Unterschied zum Inhaltstyp Signed Data wird jedoch die Signatur getrennt vom zu unterschreibenden Inhalt mit **`base64`** kodiert.

Eine Nachricht mit Inhaltstyp **`multipart/signed`** besteht typischerweise aus zwei Teilen. Der erste Teil kann von einem beliebigem Inhaltstyp sein und der zweite Teil ist vom Typ **`application/pkcs7-signature.`** Der S/MIME-Typ **`application/pkcs7-signature`** enthält ein Objekt vom Typ signedData (siehe 5.3.7.3 SignedData Content Type), mit der Spezialität, dass der Inhalt fehlt und nur die Signatur vorhanden ist.

Empfänger ohne S/MIME können damit den Inhalt einer Nachricht lesen, können aber nicht die Signatur überprüfen.

Signed and Enveloped Data

Unterschrift und Verschlüsselung können ineinander verschachtelt werden, so dass unterschriebene Daten, Klartextdaten oder verschlüsselte Daten verschlüsselt oder unterschrieben sein können.

5.3.7.4 Kryptographische Algorithmen

In Tabelle 9 werden die kryptographischen Algorithmen aufgelistet, welche in S/MIME verwendet werden. Es ist hierbei wichtig zu beachten, dass die Tabelle 9 nach der aktuellen S/MIME Version 3 in [RFC2633] erstellt wurde. Die Unterschiede zu S/MIME Version 2 in [RFC2311] sind beträchtlich und werden vor allem bei den Anforderungen bezüglich den kryptographischen Algorithmen stark sichtbar. So kann es gut sein, dass kryptographische Algorithmen, die in Version 2 noch als eine Anforderung

deklariert sind, in Version 3 nur noch wegen der Rückwärtskompatibilität definiert sind. Die Anforderungen von S/MIME werden mit der in [RFC2119] spezifizierten Terminologie bezeichnet:

Funktion	Anforderung
Erzeugen eines Message Digest, mit dem die digitale Unterschrift gebildet wird	Sender und Empfänger müssen SHA-1 unterstützen. Um eine Rückwärtskompatibilität zu MIME Version 2 zu bieten, sollten Empfänger zusätzlich MD5 anbieten.
Verschlüsseln des Message Digest, um die digitale Unterschrift zu erzeugen	Sender und Empfänger müssen DSS unterstützen. Sender sollten zusätzlich die Verschlüsselung mit RSA unterstützen. Empfänger sollten dafür die Überprüfung einer RSA-Unterschrift anbieten.
Sitzungsschlüssel zur Übertragung mit der Nachricht verschlüsseln	Der Sender und der Empfänger müssen Diffie-Hellman unterstützen. Zusätzlich sollte der Agent des Senders die RSA-Verschlüsselung anbieten. Der Empfänger sollte darum die Entschlüsselung mit RSA unterstützen.
Verschlüsseln der Nachricht für die Übertragung mit Einweg-Sitzungsschlüsseln	Der Sender und Empfänger müssen die Verschlüsselung mit DES EDE3 CBC, auch 3DES genannt, unterstützen. Aus Gründen der Kompatibilität sollte ein Empfänger ebenfalls den Algorithmus RC2/40 mit der Schlüssellänge von 40 Bit unterstützen.

Tabelle 9 Kryptographische Algorithmen, die in S/MIME [RFC2633] benutzt werden.

Um einen Message Digest zu erzeugen, verlangt und empfiehlt die S/MIME-Spezifikation SHA-1 mit 160 Bit und zusätzlich sollte für die Rückwärtskompatibilität MD5 mit 128 Bit als Hash-Funktionen angeboten werden. Es bestehen gerechtfertigte Bedenken gegenüber der Sicherheit von MD5, sodass SHA-1 ganz klar die bevorzugte Alternative darstellt. Da MD5 aber so weit verbreitet ist, sollte es auch unterstützt werden.

S/MIME beinhaltet den Digital Signature Standard (DSS) und den RSA-Algorithmus, um digitale Unterschriften zu erzeugen. DSS ist dabei das zwingende Merkmal, während die Unterstützung des RSA-Algorithmus in der Spezifikation nur empfohlen wird.

Um den Sitzungsschlüssel zu verschlüsseln stellt Diffie-Hellman ist oft vorgeschrieben und S/MIME benutzt eine Variante von Diffie-Hellman mit dem Namen ElGamal. Alternativ kann auch das RSA-Verfahren sowohl für die digitale Unterschrift, als auch zur Verschlüsselung des Sitzungsschlüssels Verwendung finden.

Der 3DES muss für die Verschlüsselung und Entschlüsselung von Nachrichten unterstützt werden. Der schwache RC2/40 sollte ebenfalls aus Gründen der Kompatibilität vom Empfänger angeboten werden.

Wenn ein Sender eine Nachricht verschlüsseln will, hat er zu entscheiden welcher Algorithmus zu verwenden ist. Um diese Entscheidung treffen zu können, kann der Sender auf verschiedene Informationen zurückgreifen. Zum Beispiel könnte er auf eine vom Empfänger empfangene Liste mit Leistungsmerkmalen oder auf Voreinstellungen vom Benutzer zurückgreifen. Um eine endgültige Entscheidung diesbezüglich treffen zu können, beschreibt S/MIME folgendes Verfahren um einen Algorithmus auszuwählen:

1. Wenn der Sender eine Liste von Leistungsmerkmalen bezüglich der zu verschlüsselnden Nachricht empfangen hat, soll er aus dieser Liste das erste und vom Empfänger am meisten bevorzugte Merkmal auswählen, welches der Sender unterstützen kann.
2. Falls
    - der Sender keine Kenntnisse über die unterstützten Algorithmen besitzt
    - aber schon mindestens eine verschlüsselte und unterschriebene Nachricht vom Empfänger erhalten hat
    - und die Unterschrift als vertrauenswürdig erkannt wurde

soll die zu versendete Nachricht die gleiche Verschlüsselung verwenden, wie die letzte verschlüsselte und signierte Nachricht vom Empfänger.

3. Falls:

   - der Sender keine Kenntnisse über die Leistungsmerkmale der Verschlüsselung besitzt
   - und der Sender über keine Kenntnisse betreffs der vom Empfänger verwendeten Version von S/MIME besitzt

   soll der Sender den stärkeren 3DES-Algorithmus für die Verschlüsselung verwenden, welcher auch von S/MIME in der Version 3 gefordert wird. Falls sich der Sender entschliesst, nicht 3DES zu verwenden, soll RC2/40 benutzt werden.

5.3.7.5 S/MIME-Nachrichten

S/MIME benutzt eine Reihe neuer MIME-Inhaltstypen, welche in [RFC1847] und [RFC3369] definiert sind und die zur Übersicht in Tabelle 10 zu finden sind. Die neuen Anwendungstypen verwenden PKCS.

PKCS bezieht sich auf eine Reihe von Spezifikationen der Public-Key-Kryptographie, welche von den RSA Laboratories veröffentlicht und für S/MIME verfügbar gemacht wurden.

Haupttyp	Unterart	S/MIME Parameter	Beschreibung
Multipart	Signed		Dieser Inhaltstyp besteht aus zwei Teilen: Ein Teil enthält die zu unterschreibende Nachricht und der andere enthält die eigentliche Unterschrift vom Typ Application/pkcs7-signature.
Application	pkcs7-mime	signedData	Eine unterschriebene S/MIME-Einheit
	pkcs7-mime	envelopedData	Eine verschlüsselte S/MIME-Einheit
	pkcs7-mime	degenerated signedData	Eine S/MIME-Einheit, welche nur Public-Key-Zertifikate enthält
	pkcs7-signature		Der Unterschriftsanteil einer Nachricht vom Typ multipart/signed besteht aus diesem Inhaltstyp. Dieser Typ enthält ein Objekt vom Typ signedData (siehe Abschnitt SignedData Content Type), mit der Spezialität, dass der Inhalt fehlt und nur die Signatur vorhanden ist.
	pkcs10-mime		Eine Anfrage zur Registrierung eines Zertifikates

Tabelle 10 S/MIME-Inhaltstypen nach [RFC2633]

In Abschnitt, Sichern einer MIME-Einheit werfen wir einen Blick auf das allgemeine Verfahren zur Vorbereitung einer S/MIME-Nachricht, um anschliessend in Abschnitt EnvelopedData, SignedData und Clear Signing jeden dieser Inhalstypen zu untersuchen.

Die Parameter name und filename

Um die Kompatibilität mit älteren Systemen sicher zu stellen, sollen Sender von Nachrichten vom Typ **`application/pkcs7-mime`** den optionalen Parameter **`name`** zum Header-Feld **`Content-Type`** des MIME-Header hinzufügen. Ebenso sollte im Header-Feld **`Content-Disposition`** der Parameter **`filename`** nicht fehlen. Falls diese Parameter hinzugefügt werden, sollen diese Parameter den Wert eines Dateinamens mit der dazugehörigen Erweiterung besitzen.

MIME-Typ	Endung des Dateinamens
Application/pkcs7-mime (signedData, envelopedData)	.p7m
Application/pkcs7-mime (degenerate signedData „certs-only“ Nachrichten)	.p7c
Application/pcks7-signature	.p7s

Tabelle 11 Werte für Parameter name und filename nach [RFC2633]

Als zusätzliche Beschränkung sollte der Dateiname auf 8 Zeichen beschränkt sein. Auf den Dateinamen folgt die aus drei Zeichen bestehende Dateierweiterung. Der Dateiname kann aus einer beliebigen Zeichenkette bestehen, allerdings sollte der Dateiname smime benutzt werden um anzuzeigen, dass die MIME-Einheit mit S/MIME verknüpft ist.

Es gibt zwei Gründe um einen Dateinamen anzugeben. Er ermöglicht die leichtere Handhabung von S/MIME-Objekten als Dateien auf einer Festplatte. Der Dateiname kann auch Informationen über den Inhaltstyp über Gateways hinaus übermitteln. Betrachten wir ein Beispiel mit einer MIME-Einheit vom Typ **`application/pkcs7-mime`**. Wenn die MIME-Einheit bei einem Gateway ankommt, der kein spezielles Wissen über S/MIME besitzt, dann wird das Gateway die MIME-Einheit auf den Inhaltstyp **`application/octet-stream`** setzen und den Inhalt als ein generischen Anhang behandeln, somit wird die Informationen über den eigentlichen Typ verloren gehen. Der vorgeschlagene Dateiname wird aber dennoch über das Gateway hinaus transportiert. Oft erlaubt dieser Dateiname beim Empfänger die richtige Anwendung auszuwählen, um den mitgeführten Anhang zu bearbeiten. Es ist anzumerken, dass dieser Mechanismus in vielen Umgebungen Nutzen bringen kann. Ordnungsgemäss muss eine S/MIME-Implementation die MIME-Typen gebrauchen und nicht von den Dateinamen abhängig sein.

Sichern einer MIME-Einheit

In S/MIME kann eine MIME-Einheit durch digitale Unterschriften, Verschlüsselung oder die Kombination der beiden Merkmale gesichert werden. Eine MIME-Einheit kann die komplette Nachricht (mit Ausnahme der RFC-822-Header) darstellen, oder beim MIME-Inhaltstyp Multipart einen oder mehrere Teile der Nachricht bilden. Die MIME-Einheit wird entsprechend den üblichen Regeln für die Vorbereitung einer MIME-Nachricht gebildet. Nach diesem Vorgang wird die MIME-Einheit zusammen mit einigen sicherheitsrelevanten Daten, wie beispielsweise Bezeichnung des Algorithmus und der Zertifikate, vom S/MIME verarbeitet, um damit ein so genanntes PKCS-Objekt zu erzeugen. Ein erstelltes PKCS-Objekt wird anschliessend als Nachrichtenteil behandelt und weiter in MIME verpackt, sprich mit den entsprechenden MIME-Headern versehen. Dieser Vorgang wird verständlicher, nachdem wir im folgenden Abschnitt bestimmte Objekte und Beispiele betrachtet haben.

Die Nachricht wird immer vor der Absendung in die kanonische Form umgewandelt. Es wird für den gegebenen Typ und Unterart jeweils das entsprechende kanonische Format als Nachrichteninhalt verwendet. In einer Nachricht vom Typ **`multipart`** wird das entsprechende kanonische Format für jeden Teil erzeugt.

Besondere Aufmerksamkeit verlangt der Gebrauch der Übertragungsverschlüsselung. Das Ergebnis der Anwendung des Sicherheitsalgorithmus wird in den meisten Fällen so aussehen, dass ein teilweise oder ganz aus willkürlichen Binärdaten bestehendes Objekt erzeugt wird. Dieses Objekt wird anschliessend in eine äussere MIME-Nachricht eingeschlossen, woraufhin die Übertragungskodierung erfolgen kann, im normal Fall **`base64`**. Im Fall einer **`multipart/signed`**-Nachricht, wird der Nachrichteninhalt durch den Sicherheitsprozess nicht verändert. Sofern dieser Nachrichteninhalt nicht aus **`7bit`** besteht, sollte er zur Übertragung mit **`base64`** oder **`quoted printable`** verschlüsselt werden. Damit wird die Gefahr beseitigt, dass der Nachrichteninhalt, welchem die Unterschrift hinzugefügt wurde, verändert wird.

Folgend werden wir auf jeden der Inhaltstypen von S/MIME eingehen und Ihn erläutern.

EnvelopedData

Die Unterart Application/pkcs7-mime wird mit einem eindeutigen **smime**-Parameter für eine der vier Kategorien der S/MIME-Verarbeitung benutzt.

Das Ergebnis der S/MIME-Verarbeitung wird Objekt genannt und mittels dem Format der Basic Encoding Rules (BER) dargestellt. Ein entsprechendes Objekt sollte in der äusseren MIME-Nachricht mit **base64** zur Übertragung kodiert werden. Zunächst betrachten wird envelopedData.

Für die Vorbereitung einer MIME-Einheit vom Typ envelopedData sind folgende Schritte erforderlich:

1. Für einen bestimmten, symmetrischen Verschlüsselungsalgorithmus (RC2/40 oder 3DES) muss ein zufälliger Sitzungsschlüssel erzeugt werden.
2. Für jeden Empfänger wird der Sitzungsschlüssel mit dessen öffentlichem RSA-Schlüssel verschlüsselt.
3. Für jeden Empfänger wird ein Block namens RecipientInfo vorbereitet. Dieser Block enthält:
    - das Zertifikat des Absenders
    - die Bezeichnung des verwendeten Algorithmus um den Sitzungsschlüssel zu verschlüsseln
    - der chiffrierte Sitzungschlüssel.
4. Verschlüsseln der Nachrichteninhalt mit dem Sitzungsschlüssel.

Die Blöcke RecipientInfo und der folgende, verschlüsselte Inhalt bilden die envelopedData. Diese Blöcke werden anschliessend gemeinsam mit **base64** kodiert. Eine Nachricht ohne den RFC-822-Header könnte wie folgt aussehen (Quelle: [RFC2633]):

```
Content-Type:     application/pkcs7-mime;     smime-type=enveloped-data;
name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```

Beim Empfänger wird zunächst die **base64**-Kodierung entfernt. Danach wird mit Hilfe des privaten Schlüssels des Empfängers der Sitzungsschlüssel entschlüsselt. Anschliessend wird der eigentliche Nachrichteninhalt mit dem Sitzungsschlüssel entschlüsselt.

SignedData

Die Unterart **Application/pkcs7-mime** mit dem Parameter **signedData** kann zusammen mit einem oder mehreren Unterzeichnern benutzt werden. Um die Verständlichkeit zu verbessern, beschränken wir unsere Beschreibung auf eine einzige, digitale Unterschrift. Folgende Schritte sind für die Vorbereitung einer MIME-Einheit vom Typ signedData erforderlich:

1. Auswahl eines geeigneten Algorithmus für den Message Digest (SHA oder MD5)
2. Vom zu unterschreibenden Inhalt wird der Message Digest oder die Hash-Funktion berechnet.
3. Der Message Digest wird mit dem privaten Schlüssel des Unterzeichners signiert.
4. Ein Block namens SignerInfo wird vorbereitet. Der Block SignerInfo enthält das Zertifikat für den privaten Schlüssel des Unterzeichners, einen Bezeichner des Message Digest-Algorithmus, einen Bezeichner für den Algorithmus, mit dem der Message Digest verschlüsselt wird, sowie den verschlüsselten Message Digest.

Die Einheit signedData besteht aus einer Reihe von Blöcken, darunter ein Bezeichner des Message-Digest-Algorithmus, die unterschriebene Nachricht, sowie die SignerInfo. Zusätzlich kann die Einheit

signedData einen Satz von Zertifikaten über öffentliche Schlüssel enthalten, mit denen sich eine Kette von einer anerkannten Zertifizierungsstelle zum Unterzeichner herstellen lässt. Diese Informationen werden danach mit **base64** kodiert. Eine Nachricht ohne den RFC-822-Header könnte wie folgt aussehen (Quelle: [RFC2633]):

```
Content-Type:       application/pkcs7-mime;       smime-type=signed-data;
name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj7
77n8HHGT9HG4VQpfyF467GhIGfHfYT6rfvbnj756tbBghyHhHUujhJhjH
HUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H7n8HHGghyHh
6YT64V0GhIGfHfQbnj75
```

Damit der Empfänger die Nachricht zurückgewinnen und die Unterschrift überprüfen kann, entfernt der Empfänger zuerst die **base64**-Kodierung. Anschliessend wird mit Hilfe des öffentlichen Schlüssels des Unterzeichners der Message Digest entschlüsselt. Der Empfänger berechnet selbstständig den Message Digest über die Nachricht und vergleicht diesen mit dem entschlüsselten Message Digest, um damit die Unterschrift zu überprüfen.

Clear Signing

Clear Signing erreicht man durch die Verwendung des Inhaltstyps **multipart** und der Unterart **signed**. Die Unterart ist in [RFC1847] definiert. Es wurde bereits schon erwähnt, dass dieser Signing-Prozess keine Veränderung der Nachricht beinhaltet, die unterschrieben werden soll, sodass die Versendung im Klartext erfolgt. Aus diesem Grund kann ein Empfänger, welcher über MIME, aber nicht über S/MIME verfügt, die ankommende Nachricht lesen. Die Signatur kann von diesem Empfänger, nicht geprüft werden. Eine Nachricht vom Typ **multipart/signed** besteht aus zwei Teilen. Für den ersten Teil der Nachricht kann jeder MIME-Typ gewählt werden, er muss aber so präpariert werden, dass er auf dem Übertragungsweg nicht verändert wird. Wenn der erste Teil der Nachricht nicht im 7-Bit-Format ist, dann muss man ihn mittels **base64** oder **quoted-printable** kodieren. Dieser Teil wird daraufhin genauso verarbeitet, wie **signedData**, es wird aber ein Objekt im Format **signedData** erzeugt, bei dem das Feld **eContent** leer ist. Es handelt sich bei diesem Objekt also um eine selbstständige Signatur. Diese Signatur wird anschliessend mit Hilfe von **base64** kodiert, um damit den zweiten Teil der Nachricht vom Typ **multipart/signed** zu bilden. Dieser zweite Teil besitzt den MIME-Typ **application** und die Unterart **pkcs7-signature**. Eine **multipart/signed** Nachricht ohne den RFC-822-Header könnte wie folgt aussehen (Quelle: [RFC2633]):

```
Content-Type: multipart/signed;
protocol="application/pkcs7-signature"
micalg=sha1; boundary=boundary42
--
boundary42
Content-Type: text/plain

This is a clear-signed message.

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42--
```

Der Protokoll-Parameter informiert, dass es sich bei der Nachricht um eine zweiteilige clear-signed-Einheit handelt. Der Parameter micalg kennzeichnet den Typ des verwendeten Message Digest an.

Die Unterschrift kann vom Empfänger überprüft werden, indem der Empfänger den Message Digest über den ersten Teil der Nachricht berechnet und diesen Wert mit dem Message Digest, welcher aus der Unterschrift im zweiten Teil zurückgewonnen wurde, vergleicht.

Registrierungsanfrage

Ein Benutzer der ein Public-Key-Zertifikat haben will, wendet sich normalerweise an eine Zertifizierungsstelle. Die S/MIME-Einheit **`application/pkcs10`** kann nun zur Übertragung einer solchen Registrierungsanfrage benutzt werden. Die Registrierungsanfrage enthält einen Block certificattionRequestInfo, gefolgt vom Bezeichner des verwendeten Algorithmus für den öffentlichen Schlüssel und von der Unterschrift des Blocks certificattionRequestInfo, welche mit dem privaten Schlüssel des Absenders erstellt wurde. Der in der Anfrage enthaltene Block certificattionRequestInfo enthält den öffentlichen Schlüssel des Benutzers in Form eines Bit-Strings und den Namen für das Subjekt des Zertifikates,. Das Zertifikat stellt die Einheit dar, dessen öffentlicher Schlüssel zertifiziert werden soll.

Certificates-Only-Nachricht

Als Antwort auf eine Registrierungsanfrage kann eine Nachricht versendet werden, welche nur Zertifikate oder eine Certificate Revocation List (CRL) enthält. Die Nachricht ist vom S/MIME-Typ **`application/pkcs7-mime`** mit dem S/MIME-Parameter **`degenerate`**. Die notwendigen Schritte für eine die Erstellung einer solchen Nachricht sind dieselben wie für das Erzeugen einer Nachricht vom Typ signedData, mit der Ausnahme, dass die Felder **`eContent`** und **`signerInfo`** leer sind.

5.3.8 Real-time Transport Protocol (RTP)

Der Abschnitt Real-time Transport Protocol (RTP) ist mehrheitlich aus der Projektarbeit PA2 Sna 03/2 [PA_SIP] von Daniel Kaufmann und Andreas Stricker entnommen.

SIP regelt nur den Verbindungsaufbau und handelt mit Hilfe von SDP mit der Gegenstelle ein geeignetes Transportprotokoll aus. Die eigentlichen Nutzdaten, bzw. Sprachdaten werden bei der SIP-Telefonie über das Real-Time-Protocol (RTP) transportiert.

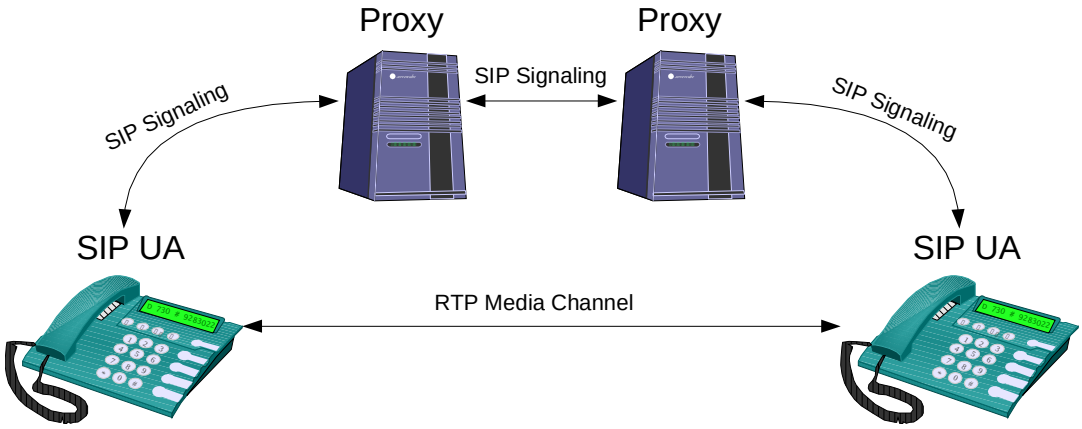


Abbildung 11 Bedeutung des Medien-Kanals

Das Real-time Transport Protocol (RTP) wird in [RFC1889] beschrieben. RTP bietet Funktionen für einen End-zu-End-Netzwerk-Transport, welche besonders geeignet sind für Anwendungen die real-time Daten, wie Audio, Video oder Daten für Simulationen, über multicast oder unicast Netzwerkdienste transportieren. Internet-Telefonie ist eine solche zeitkritische Anwendung. Steigt die Verzögerung der Sprachdaten auf ca. 250ms an, so wird diese Verzögerung schon als störend empfunden. Eine vernünftige Kommunikation ist ab einer Verzögerung von ca. 1s nicht mehr möglich. Weiter wirken sich Verzögerungsschwankungen (Jitter) als sehr störend auf die Übertragung aus. Das RTP-Protokoll [RFC1889] ist auf den Transport solcher kritischer Daten spezialisiert und bietet einen kleinen Overhead und Methoden zur Messung der Übertragungsqualität. In Verbindung mit der Internet-Telefonie wird das Real-time Transport Protokoll üblicherweise über UDP transportiert, kann aber auch auf anderen Transportprotokollen, wie zum Beispiel TCP, aufsetzten.

Die RTP-Übertragung setzt sich üblicherweise aus zwei Protokollen zusammen. Zum eigentlichen Nutzdatentransport wird das Real-time Transport Protocol (RTP) verwendet. RTP alleine bietet keine Reservierung von Ressourcen und auch garantiert auch keine quality-of-service für realtime Dienste. Zum Transport von Status-, Monitoring- und Qualitätsmeldungen wird auf das RTP Control Protocol (RTCP) zurückgegriffen. Der Datentransport über RTP ist somit um RTCP erweitert, um die Datenzustellung in grossen unicast und multicast Netzwerken in skalierbarer Weise zu überwachen und um zusätzliche minimale Kontroll- und Identifikations- Funktionalität zu bieten. So wie RTP, wurde auch RTCP designed um unabhängig von den darunter liegenden Transport und Netzwerk Schichten funktionieren zu können.

5.3.8.1 RTP-Paketaufbau

Das RTP-Paket hat das folgende Format:

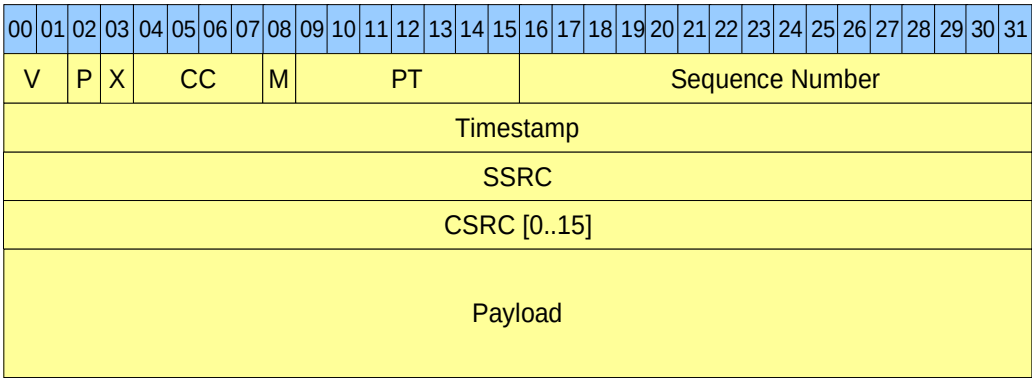


Abbildung 12 RTP-Header Version 2

Die ersten zwölf Oktette sind immer in jedem RTP-Paket vorhanden, während die Liste mit CSRC's nur vorhanden ist, wenn sie durch einen Mixer eingefügt wurde.

V, Version: RTP-Versions Nummer. Immer auf 2 gesetzt.

P, Padding: Wenn Padding Bit gesetzt, dann folgen noch weitere Padding-Bytes am Ende des Headers die nicht zum Payload gehören.

X, Extention: Ist das Header-extention Bit gesetzt, folgt genau eine Headererweiterung an den RTP-Header.

CC, CSRC Count: Gibt die Anzahl der CSRC-Bezeichner an, die nach dem fixen Headerteil folgen.

M, Marker: Die Interpretation des Marker-Bits ist abhängig von einem Profil. Es kann zum Beispiel zur Markierung eines Frames, innerhalb eines Paketstreams verwendet werden.

PT, Payload Type: Identifiziert das Format des RTP-Payloads. (Audio, Video, Codec etc.) und bestimmt wie die RTP-Payload von der Anwendung interpretiert werden muss.

Sequence Number: Eine Nummer, deren Startwert zufällig gewählt wird, die bei jedem RTP-Paket in einem Datenstrom inkrementiert wird. Die Sequence Number kann von einem Empfänger gebraucht werden, um verlorene Datenpakete zu erkennen und um die richtige Reihenfolge der Pakete wiederherzustellen.

Timestamp: Ein Zeitstempel, der zur Berechnung von Verzögerung und Verzögerungs-schwankungen benötigt wird. Das Format dieses Zeitstempels ist je nach Paketrate verschieden.

SSRC: Die Synchronistation-Source ist ein weiterer Zufallswert, der aber bei einer Übertragung stets gleich bleibt. Er dient zur Identifikation der Datenquelle. Die SSRC sollte eindeutig sein und bei einer Änderung der Datenquelle neu gewählt werden.

CSRC: Die Contributing Source ist eine Liste von 0 bis 15 CSRC Elementen, welche die beteiligten Quellen an einem Datenstrom identifiziert. Die Anzahl Elemente in dieser Liste wird durch das CC Feld gegeben.

Das Payload enthält die Daten. Das Format der Daten wird im Headerfeld PT definiert. Bei SIP Internet-Telefonie ist dies z.B. ein Audiostrom in G.711 uLaw codiert.

5.3.8.2 RTCP-Protokoll

Das RTP Control Protocol (RTCP) ist das Protokoll zur Kontrolle des RTP-Stroms. Es werden verschiedene Informationen des Übertragungskanals zwischen Sender und Empfänger transportiert. Die Paketrate dieser Kontroll-Pakete ist abhängig von der Nutzdatenrate. Ein RTCP-Paket besteht aus einem definierten Header (Abbildung 13) und einem oder mehreren Report-Blocks. Bei diesen Report-Blocks wird wiederum zwischen Sender-Report SR (Abbildung 14) und Receiver-Blocks RR (Abbildung 15) unterschieden, wobei der Sender-Report vom Sender an den Empfänger und der Receiver-Report vom Empfänger and den Sender geschickt wird.

RTCP-Header

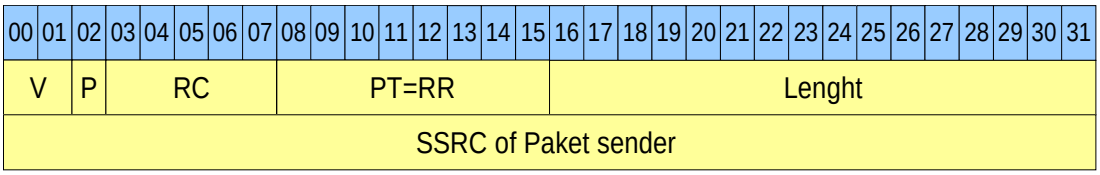


Abbildung 13 RTCP-Header

Der RTCP-Header ist der fixe Teil eines RTCP-Paketes und ist bei SR und RR immer gleich aufgebaut. Auf den RTCP-Header folgen die Report-Blocks.

V, Version:	Die RTPC-Versionsnummer ist, wie bei RTP immer auf 2 gesetzt.
P, Padding:	Wenn Padding Bit gesetzt ist, dann folgen noch weitere Padding Bytes am Ende des Headers die nicht zur Kontrollinformation gehören.
RC:	Der Receiver Report Count gibt an wie viel Report-Blocks an den Header anschliessen.
PT:	Der Packet Type definiert, ob es sich beim RTCP-Paket um ein Sender-Report (PT=200) oder um ein Receiver-Report (PT=201) handelt.
Lenght:	Die Länge des RTCP-Paketes inklusive Header und eventuelle Paddings.
SSRC:	Die Synchronistation-Source: Ein Zufallswert, der bei einer Übertragung stets gleich bleibt. Er dient zur Identifikation der Datenquelle.

Sender-Report

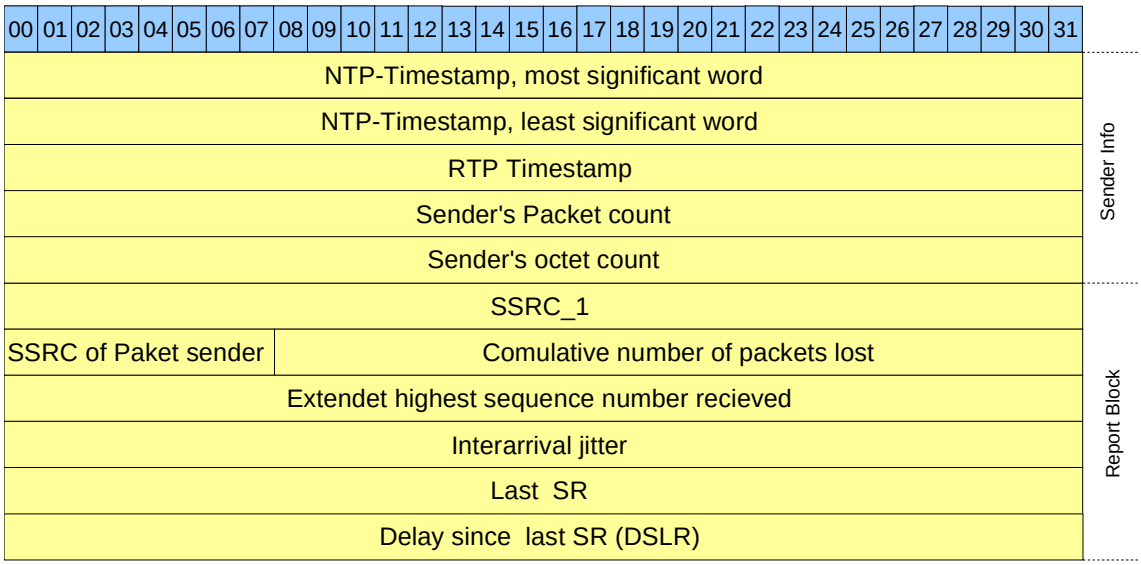


Abbildung 14 RTCP- Sender Report

Der Sender Report besteht aus einem Sender-Info-Teil und einem oder mehreren Report-Blocks, die an den RTCP-Header folgen.

Sender Info:

NTP-Timestamp: Die Wallclock Zeit, zu der die Nachricht abgesendet wurde.

RTP-Timestamp: Ein Zeitstempel, mit der gleichen Information wie der NTP-Timestamp, aber im Format des RTP-Zeitstempel.

Sender's Packet Count:
Ein Zähler der die bereits gesendeten RTP-Pakete zählt.

Sender's Octet Count:
Die Anzahl Daten-Oktetts die bereits gesendet wurden.

SSRC_n: Der Source Identifier bezeichnet die Quelle des Datenstroms, zu der die RTCP Nachricht gehört.

Fraction lost: Prozentualer Anteil RTP-Pakete, die seit dem letzten RTCP-Paket verloren gegangen sind.

Comulative Packetlost:
Die Gesamtanzahl der verloren gegangenen Pakete seit Bestehen der Verbindung.

Extd. Highest seq. Number:
Sequenznummer des letzten gesendeten RTP-Paketes.

Interarrivial Jitter: Statistische Varianz der Ankunftszeiten der RTP-Pakete, gemessen in Timestamp-Einheiten.

Last SR: Zeitstempel des letzten RTCP-Paketes (Sender Report).

Delay since last SR: Zeitspanne, die seit dem Versenden der letzten RTCP-Nachricht verstrichen ist.

Falls mehrere Datenströme über die gleiche Übertragungsstrecke laufen (z.B. Audio- und Video-Datenstrom) so folgen noch weitere Report-Blöcke. Je einer für jeden RTP-Datenstrom.

Receiver Report

Im Gegensatz zu dem Sender-Report besteht ein Receiver-Report nur aus dem Header und einem oder mehreren Report-Blocks. Ein Sender-Info bzw. Receiver-Info entfällt.

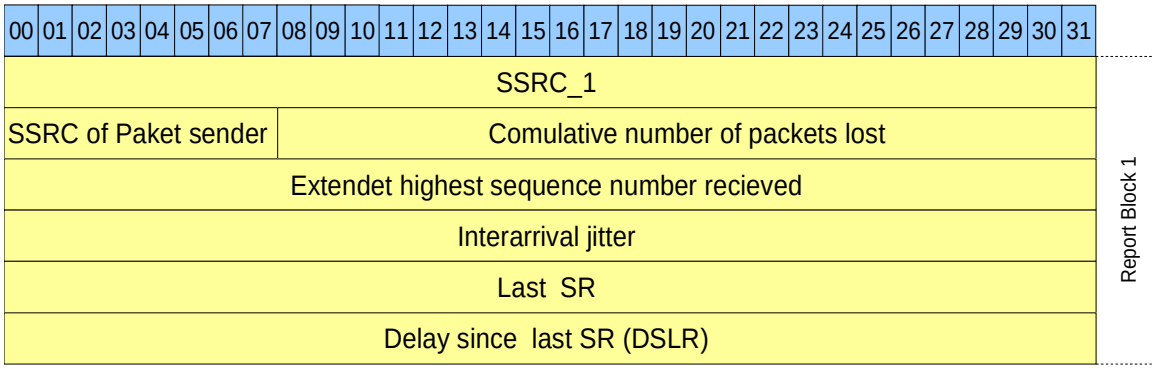


Abbildung 15 SRTCP- Receiver Report

SSRC_n: Der Source Identifier bezeichnet die Quelle des Datenstroms, zu der die RTCP Nachricht gehört.

Fraction lost: Prozentualer Anteil RTP-Pakete, die seit dem letzten RTCP-Paket verloren gegangen sind.

Comulative Packetlost: Die Gesamtanzahl der verloren gegangenen Pakete, seit Bestehen der Verbindung.

Extd. Highest seq. Number: Sequenznummer des letzten empfangenen RTP-Paketes.

Interarrivial jitter: Statistische Varianz der Ankunftszeiten der RTP-Pakete, gemessen in Timestamp-Einheiten.

Last SR: Zeitstempel des letzten empfangenen RTCP-Paketes (Sender Report).

Delay since last SR: Zeit seit dem Empfang des letzten RTCP-Paketes.

Weitere RTCP-Pakettypen

Es existieren weitere Möglichkeiten zur Übertragung von Quelleninformation über so genannte Source-Description RTCP-Pakete (SDES). Solch ein SDES Paket enthält einen Header, der die Anzahl und die Art der nachfolgenden Informationselemente beschreibt. Solche Elemente können je aus Username, E-Mail Adresse, Telefonnummer, Standortinformation, Applikationsinformation oder Endpunktinformation bestehen.

Ein weiteres RTCP-Paket ist die BYE-Meldung (Goodbye). Dieses regelt den Abbau der RTP-Verbindung. Eine BYE-Meldung besteht aus denselben Elementen wie ein RTCP-Header, wobei das Element PT mit dem Wert 203 den Abbau der Verbindung signalisiert. Es bestehen noch zusätzliche, optionale Felder, mit denen der Grund für den Verbindungsabbau angegeben werden kann.

5.3.9 Secure Real Time Protocol SRTP

5.3.9.1 Einführung

Wenn bei der SIP-Telefonie das RTP-Protokoll zum Transport der Sprachdaten verwendet wird, ist kein Sicherheitsmechanismus aktiv. Eine Sicherung des SIP-Protokolls hat keinen Einfluss auf die Nutzdaten. Eine einfache Abhilfe schafft das Secure Real-time Transport Protocol (SRTP), welches in [SRTP] beschrieben wird.

Das Dokument [SRTP] ist zurzeit noch ein 'working draft' der IETF und liegt mittlerweile in der zehnten Auflage vor. In den bisherigen Versionen gab es einige Änderungen am Draft und es ist nicht ausgeschlossen, dass noch weitere Änderungen folgen werden, bevor es geplant ist im Jahr 2005 SRTP als RFC der IETF zu verabschieden.

Das Transportprotokoll SRTP ist eigentlich ein Profil, welches RTP erweitert und deshalb die gleichen Eigenschaften wie RTP bietet, aber Mechanismen zur Verschlüsselung, sowie einen Integritäts- und Replay-Schutz mitbringt. Zur weiteren Eigenschaft von SRTP zählt der niedrige Berechnungsaufwand für die Verschlüsselung und die Integritätssicherung. Wie auch bei RTP besteht eine SRTP-Übertragung aus einem SRTP Paketstrom für die Nutzdaten und einem SRTCP-Paketstrom zur Überwachung des Datenkanals. SRTP eignet sich somit für die Übertragung von Sprachdaten bei der SIP-Telefonie sehr gut. Da es sich bei SRTP derzeit noch um einen working draft handelt ist es nicht verwunderlich, dass SRTP noch nicht bei der Vovida Vocal Implementation integriert ist.

5.3.9.2 Ziele und Merkmale

Mit den Sicherheitsmechanismen in SRTP will man bezüglich der Sicherheit folgende Ziele erreichen:

- Vertraulichkeit der RTP- und RTCP-Payload
- Integrität der ganzen RTP und RTCP Pakete, zusammen mit dem Replay-Schutz

All diese Sicherheitsmerkmale sind optional und unabhängig von einander. Die Ausnahme bildet hier der notwendige Integritätsschutz bei SRTCP, da sonst böswillige oder fehlerhafte RTCP Nachrichten die Verarbeitung des RTP-Streams stören könnten.

Neben den Zielen bezüglich der Sicherheit verfolgt SRTP noch weitere zusätzliche, funktionale Ziele. Kurz zusammengefasst lassen sich diese wie folgt beschreiben:

- Mit SRTP soll ein Framework realisiert werden, welches die Erweiterung mit neuen kryptographischen Algorithmen erlaubt.
- Der Verbrauch an Bandbreite soll klein gehalten werden können, zum Beispiel soll die Möglichkeit erhalten bleiben RTP-Header effizient komprimieren zu können.
- Der Berechnungsaufwand für die kryptographischen Sicherheitsmerkmale soll klein sein.
- Um die Sicherheitsmerkmale realisieren zu können soll der Codeumfang und Verbrauch an Speicher möglichst klein gehalten werden können. Zum Beispiel sollen die Listen für den Replay-Schutz und die Informationen für das Keymanagment wenig Codeumfang besitzen und Speicher erfordern.
- Die Expansion der Pakete soll durch hinzufügen der kryptographischen Merkmale klein gehalten werden, um damit die Anforderung an den sparsamen Gebrauch der Bandbreite nicht zu verletzten.
- Unabhängigkeit von den in RTP genutzten und zu Grunde liegenden Transport-, Netzwerk-Protokollen und den physikalischen Schichten. Im Besonderen muss SRTP eine grosse Toleranz gegenüber Paketverlust und der Übertragungsreihenfolge von den Paketen besitzen.

Mit diesen Eigenschaften soll erreicht werden, dass SRTP ein brauchbares Profil für RTP/RTCP in drahtgebundenen und drahtlosen Umgebungen darstellt.

Zu den bereits genannten Zielen, bietet SRTP zusätzliche Merkmale. Diese Merkmale sind eingeführt worden, um das Protokoll von den Lasten des Schlüsselmanagements zu befreien und um die Sicherheit

weiter zu erhöhen.

- Aus einem einzigen Master-Key können die nötigen Schlüssel gebildet werden, um die Merkmale Vertraulichkeit und Integrität für den SRTP-Stream und den korrespondierenden SRTCP-Stream zu bilden. Dieses Merkmal wird durch eine Funktion realisiert, mit welcher die nötigen Session-Schlüssel für die angewendeten Sicherheitsmechnismen aus dem Master-Key abgeleitet werden können. Diese Funktion wird auch als Key-Derivation[i] bezeichnet.
- Zusätzlich kann die Key-Derivation so konfiguriert werden, dass die nötigen Session-Keys periodisch erneuert werden. Durch diese periodische Erneuerung wird die Menge an Schlüsseltext beschränkt, die mit einem einzigen, konstanten Schlüssel erzeugt wird. Durch diese Beschränkung wird es einem Angreifer erschwert, den Schlüsseltext zu brechen.
- Ohne die Verwendung von Salting-Keys werden gleiche Klartexte auch zu gleichen Schlüsseltexten abgebildet. Diese Eigenschaft kann für verschiedene Angriffe verwendet werden. Durch die Verwendung von wechselnden Salting-Keys werden gleiche Klartexte in unterschiedliche Schlüsseltext abgebildet. Damit werden diese Attacken erschwert oder unmöglich.

5.3.9.3 SRTP Framework

Wie erwähnt ist SRTP als ein so genanntes Profil für RTP definiert, welches RTP um kryptographischen Merkmale erweitert. Konzeptionell kann SRTP als eine zusätzliche Schicht im Stack angesehen werden, welche sich zwischen der RTP- und der Transport-Schicht befindet. SRTP fängt Pakete von der RTP-Schicht ab, erweitert diese um die gewünschten, kryptographischen Merkmale und reicht sie an die Transport-Schicht weiter. Auf der Seite des Empfängers werden die Pakete, welche von der Transport-Schicht kommen, von der SRTP-Schicht abgefangen und bearbeitet, um sie an anschliessend an die RTP-Schicht zu übergeben.

Secure RTCP (SRTCP) bietet die gleichen Dienste bezüglich der Sicherheit für RTCP an. Die Authentifizierung von SRTCP-Nachrichten ist ein zwingendes Merkmal und schützt die Felder der RTCP-Pakete.

5.3.9.4 SRTP-Paketaufbau

Das SRTP-Paket ist in etwa gleich aufgebaut wie das RTP-Paket. Der wesentliche Unterschied besteht darin, dass der Payload verschlüsselt ist und an den Payload weitere Felder zur Schlüsselidentifikation (Digest) und Integritätssicherung folgen (Abbildung 16).

Die Felder, die bei SRTP identisch zu RTP sind, haben auch dieselbe Funktion. Es folgt ein optionales Feld für RTP-Headererweiterungen. Normalerweise gehört diese Information bei RTP zum Header, es macht aber durchaus Sinn diese Information durch Verschlüsselung zu schützen, da diese Felder schützenswerte Daten enthalten können.

i dt. Schlüsselbildung

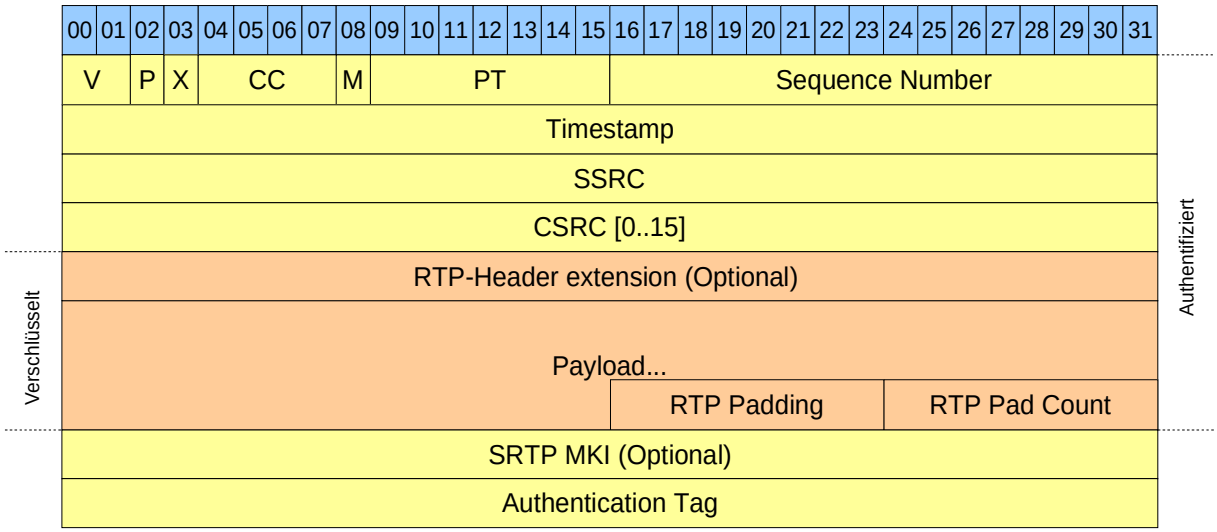


Abbildung 16 SRTP-Paketaufbau

An diese Headererweiterung folgen die eigentlichen Nutzdaten. An die Nutzdaten können anschliessend zwei Felder für RTP-Padding und RTP-Padding Count folgen.

Der durch die Verschlüsselung geschützte Teil besteht also aus den optionalen Headererweiterungen, der eigentlichen Payload und falls vorhanden aus dem RTP-Padding und dem RTP Pad Count. Kein im Standard definierter Verschlüsselung-Algorithmus benützt padding, somit besitzt die RTP- und die SRTP-Payload die gleiche Grösse. Neue Algorithmen für SRTP könnten eventuell padding benötigen und die SRTP-Payload vergrössern. RTP bietet dafür sein eigenes Padding-Format und falls benötigt sollte dieses RTP-Padding auch von den SRTP-Algorithmen gebraucht werden. Algorithmen können aber auch ihre eigene Padding-Methode benötigen und müssen dann den Umfang, das Format und die Bearbeitung ihres Paddings definieren. Es ist an dieser Stelle allerdings sehr wichtig zu erwähnen, dass Algorithmen mit Padding für Attacken verletzlicher sein können [SRTP]. Diese Gefahr wird noch grösser, wenn keine Authentifizierung der Nachrichten verwendet wird. Aus diesem Grund ist es wichtig, dass für jede neue Spezifikation über einen Verschlüsselungs-Algorithmus, der Gebrauch und die Anwendung von Padding durchdacht werden muss.

Dem verschlüsselten Teil des Paketes folgt ein optionales Feld zur Identifizierung des Master-Keys und ein empfohlenes Feld für eine Paket-Signatur (Authentication Tag). Die Felder MKI und Authentication Tag sind die einzigen, die in SRTP und nicht RTP definiert sind und folgenden Bedeutung besitzen:

- MKI (Master Key Idendifier): Dieses Feld ist optional und konfigurierbar in der Länge. Dieses Feld wird durch das Schlüsselmanagement definiert und gebraucht. Der MKI identifiziert den Master-Key, von dem die Session-Keys abgeleitet sind um das Paket zu verschlüsseln und/oder zu authentifizieren. Werden die Session-Keys periodisch erneuert, wird der MKI auch vom Schlüsselmanagement verwendet um diese Funktionalität zu realisieren.
- Authentication tag: Diese empfohlene Feld enthält die Daten für die Authentifizierung des Paketes. In die Authentifizierung fliessen der RTP-Header und die darauf folgenden, verschlüsselten Daten des SRTP-Paketes hinein. Somit muss beim Gebrauch von Verschlüsselung und Authentifizierung, zuerst die Chiffrierung der Daten vorgenommen werden um anschliessend den Wert für das authentication tag zu berechnen. Auf der Seite des Empfängers ist der Vorgang von Authentifizierung und Verschlüsselung in umgekehrter Reihenfolge durchzuführen. Das authentication tag bietet Authentifizierung des RTP-Headers und der Payload. Indirekt bietet dieses authentication tag somit Schutz gegen Replay-Attacken, da die Sequenznummer ebenfalls in die Berechnung für das authentication tag hineinfliesst. Das Feld MKI wird nicht geschützt, da dies keinen zusätzlichen Schutz bieten würde.

5.3.9.5 SRTCP-Protokoll

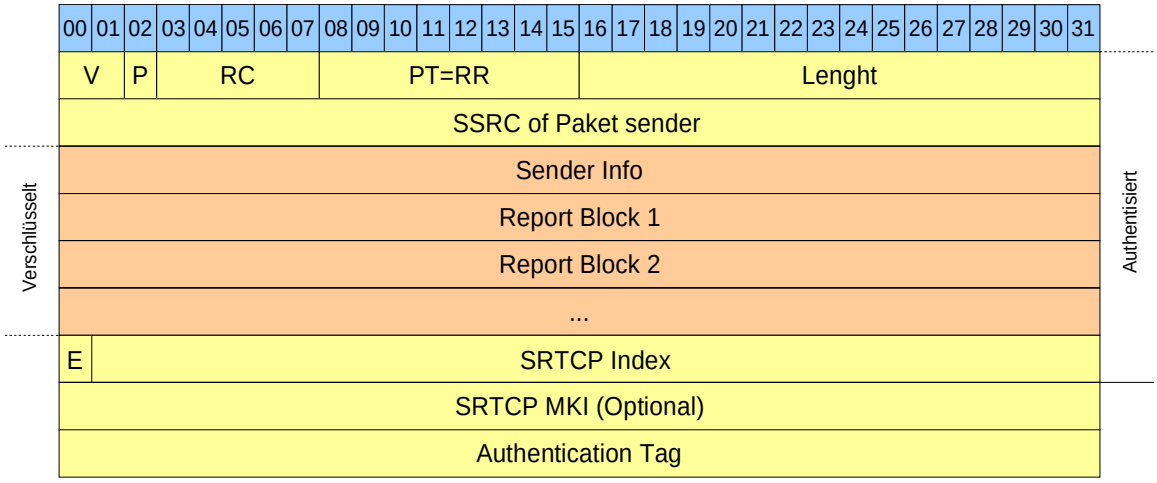


Abbildung 17 SRTCP-Paket

SRTCP übernimmt wie auch das RTCP-Protokoll die Funktion der Nutzdatenkanalüberwachung. Es existieren wie bei RTCP Sender- und Receiver-Reports. Nach einem RTCP-Header folgen die spezifischen Report-Blöcke. Die Felder, die bei SRTP identisch zu RTP sind, haben auch dieselbe Funktion.

SRTCP fügt mit dem E-Flag, dem SRTCP Index und dem Authentication Tag drei neue und erforderliche Felder hinzu, sowie mit SRTCP MKI ein optionales Feld. Die neuen Felder besitzen folgende Bedeutung:

- E-Flag: Dieses erforderliche „Encryption Bit“ (Feld E in Abbildung 17) wird als Ergänzung bei verschlüsseltem RTCP-Inhalt gesetzt.
- SRTCP Index: Weiter ist mit dem erforderlichen SRTCP Index ein Zähler eingefügt, der bei jedem SRTCP-Paket inkrementiert wird. Der Index ist in jedem Paket explizit enthalten. Im Gegensatz zum Ansatz mit einem impliziten Index, wie er in SRTP-Paketen verwendet wird.
- SRTCP MKI: Diese Feld ist optional und besitzt eine konfigurierbare Länge. Der MKI ist der Master Key Identifier und besitzt die gleiche Bedeutung wie das MKI-Feld im SRTP-Paket. Somit wird diese Feld vom Schlüsselmanagement verwendet, da mit dem MKI die Master-Key identifiziert wird und damit auch die davon abgeleiteten Session-Keys.
- Authentication Tag: Das Authentication Tag ist erforderlich und besitzt eine ebenfalls eine konfigurierbare Länge. Das Authentication Tag dient zur Sicherung der Integrität.

Der durch die Verschlüsselung geschützte Teil besteht aus dem Sender Info und den Report-Blöcken. Durch die Authentifizierung wird noch zusätzlich der RTCP-Header und das vom SRTCP hinzugefügte E-Flag, sowie der SRTCP-Index gesichert.

5.3.9.6 SRTP-Sicherheitsmechanismen

Es gibt sehr viele Algorithmen für die Verschlüsselung und Authentifizierung, die sich zum Gebrauch mit SRTP anbieten würden. In [SRTP] wird eine Auswahl an Algorithmen getroffen und in erforderliche, empfohlene und standardmässige Algorithmen eingeteilt.

Tabelle 12 bietet einen kurzen Überblick über die verschiedenen Transformationen und zeigt auch, welche von den Transformationen zwingend, optional und standardmässig sind. In den folgenden Abschnitten wird näher auf die verschiedenen Transformationen eingegangen.

	Zwingend	Optional	Standard
Verschlüsselung	AES-CM, NULL	AES-f8	AES-CM
Nachrichten-Integrität	HMAC-SHA1	-	HMAC-SHA1
Schlüsselbildung	AES-CM	-	AES-CM

Tabelle 12 Zwingende-, Optionale- und Standard-Transformationen für SRTP und SRTCP

Schlüsselbildung

Das SRTP-Protokoll benötigt einen einzelnen, symmetrischen Master-Key. Die Verteilung und die Erneuerung dieses Master-Keys ist nicht in [SRTP] spezifiziert. Möglichkeiten zur Verteilung dieses Master-Keys sind MIKEY, beschrieben in 5.3.10 MIKEY oder SDP.

Aus dem Master-Key, dem Paketindex und einem eventuell vorhandenem Master-Key-Salt werden insgesamt sechs verschiedene Schlüssel generiert: Für SRTP und SRTCP jeweils der Session-Encryption-Key, der Session-Authentication-Key und der Session-Salt-Key. Der Session Encryption Key ist ein Schlüssel der für die jeweilige aktive Übertragung gültig ist und mit dem die Nutzdaten verschlüsselt werden. Um den Schlüsselgenerator des Schlüsselstroms für die Datenverschlüsselung zu initialisieren wird der Session-Salt-Key verwendet. Der Algorithmus zur Bildung dieser Schlüssel ist in den SRTP-Spezifikationen nachzulesen [SRTP].

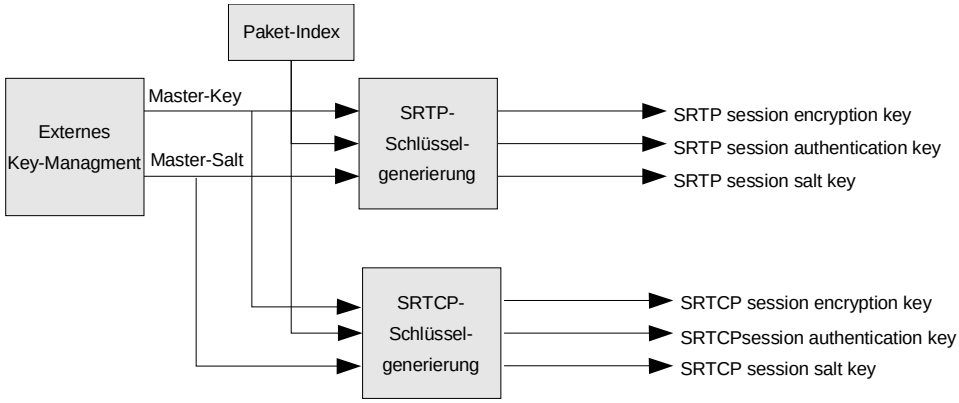


Abbildung 18 Schlüsselbildung

Um diese sechs verschiedenen Schlüssel zu generieren, wird im Draft [SRTP] auf den Advanced Encryption Standard (AES) im Counter-Mode (AES-CM) zurückgegriffen. Der Algorithmus wird eigentlich für die Verschlüsselung der Nutzdaten verwendet und wird im Zusammenhang mit SRTP auch für die Generierung der nötigen Schlüssel verwendet.

Verschlüsselung

Die Verschlüsselung erfolgt nun mit dem schnellen und modernen Advanced Encryption Standard (AES) im Counter-(AES-CM) bzw. F8-Mode (AES-F8). Der AES-Verschlüsselungs-Algorithmus arbeitet in diesen zwei genannten Modi auf der Basis der Bit-weisen XOR Verknüpfung von Schlüssel und Klartext. Für detailliertere Informationen über den AES-Verschlüsselungsalgorithmus siehe: „Federal Processing Standards Publication 197“ [FIPS-197].

Der AES-Verschlüsselungsalgorithmus verarbeitet einen Klartext-Block in einen verschlüsselten Block gleicher Länge, was die Paketgrösse des SRTP somit nicht beeinflusst. Es kann auch ein NULL-Algorithmus gewählt werden, wobei dann die Klartext-Daten unverändert als Schlüsseltext kopiert werden und somit die Nachricht nicht verschlüsselt wird.

Die Verbindungsinformationen beim SRTCP-Protokoll wird nach dem gleichen Prinzip verschlüsselt.

Zur Entschlüsselung kann der ganze Vorgang umgekehrt werden. Der Durchsatz, der Verschlüsselung

bzw. Entschlüsselung ist bei dem verwendeten AES-Verfahren sehr hoch und eignet sich somit sehr gut zur verschlüsselten Übertragung von zeitkritischen Sprachdaten.

Integritätssicherung

Um sicher zu gehen, dass der Inhalt der Nachricht im SRTP-Payload während der Übertragung nicht verändert wurde, gibt es bei SRTP, bzw. SRTCP die Möglichkeit den Inhalt zu signieren. Die Signatur wird mit dem HMAC-SHA1 Verfahren gebildet. Es wird zuerst ein Message Digest der Daten mittels SHA1 gebildet, welcher immer noch im „Klartext" ist. Somit könnte eine unverschlüsselte Nachricht von einem Angreifer verändert und der Message Digest, mit dem bekannten Message Digest-Algorithmus neu berechnet werden. Um den gefälschten Message Digest zu entdecken, muss der Benutzer den korrekten Message Digest bereits kennen. Um dies zu umgehen wird dieser noch per HMAC-Verfahren gesichert. Zum Verschlüsseln des Message Digest wird der Session-Authentcation-Key verwendet. Das Resultat besitzt eine Länge von 80-Bit. Wie in den Abschnitten 5.3.9.4 und 5.3.9.5 erwähnt, ist das Authentication-Tag in der Länge konfigurierbar. Es könnten somit die vollen 80-Bit in das Authentication-Tag eingefügt werden und dies wird auch sehr empfohlen. Dennoch kann es Anwendungen geben, bei denen ein kürzere Länge des Authentication-Tag vollkommen ausreichend ist. Man kann sich zum Beispiel eine Telefon-Anwendung vorstellen, die den G.729 Audio-Codec mit einem Paketisierungsintervall von 20 Millisekunden verwendet und durch ein Authentication-Tag von 32-Bit geschützt ist. Mit der Länge des Authentication-Tag von 32-Bit beträgt die Wahrscheinlichkeit für jedes Paket, dass es erfolgreich verfälscht werden kann, eins zu $2^{32.}$ Somit kann ein Angreifer während einer Periode von 994 Tagen im Durchschnitt nur 20 Millisekunden kontrollieren. Somit ist für viele Anwendung ein Authentication-Tag von 32-Bit vollkommen ausreichend und die 80-Bit können auf die nötigen 32-Bit gekürzt werden.

Replay-Schutz

Sicherer Replay-Schutz ist nur möglich, wenn Integritätssicherung vorhanden ist. Es wird sehr empfohlen für SRTP und SRTCP den Replay-Schutz zu benützen, da die Integritätssicherung alleine keinen Replay-Schutz sicherstellen kann.

Ein Paket wird „replayed", wenn es von einem Angreifer zwischen gespeichert wurde und zu einem späteren Zeitpunkt erneut ins Netzwerk eingespielt wird. Wenn die Authentifizierung von Nachrichten angeboten wird, bietet SRTP den Replay-Schutz gegen solche Attacken durch Replay-Listen an. Jeder SRTP-Empfänger unterhält eine Replay-Liste, welche konzeptionell alle Indizes der Pakete enthält, die empfangen und authentifiziert wurden. Für diesen Zweck muss der Index eines Paketes unter anderem aus der Sequenznummer des Paketes und dem rollover counter (ein Zähler) berechnet werden. In der Praxis kann die Liste den Ansatz des „sliding window" benützen , so dass eine feste Grösse von Speicher für die Realisation des Replay-Schutzes ausreicht.

Der Empfänger betrachtet den Index eines empfangenen Paketes gegen die Replay-Liste und das sliding window. Es sollen nur Pakete akzeptiert werden, deren Index dem sliding window voraus ist oder innerhalb des sliding window liegt, aber noch nicht empfangen wurden.

Nachdem des Paket akzeptiert wurde, wird es authentifiziert, falls nötig muss zuerst das sliding window weiter geschoben werden und dann die Replay-Liste aktualisiert werden.

Die Replay-Liste kann sehr effizient implementiert werden, indem ein Bitmap verwendet wird, um die Pakete zu kennzeichnen die schon bereits empfangen wurden.

Re-Keying

Beim Vorgang des Re-Keying wird nach verstrichener Zeit oder nach einer bestimmten verschlüsselten Datenmenge ein neuer Master-Key bestimmt und verwendet. Dieser Vorgang dient vor allem zur Steigerung der Sicherheit, falls ein Master-Key kompromittiert wird, ist nur der Teil der Kommunikation betroffen, welche den betreffenden Master-Key verwendet hat. Alle anderen Daten sind weiterhin geschützt und für den Angreifer nicht einsehbar. In SRTP sind Mechanismen für das Merkmal Re-Keying vorgesehen. Die nötigen Interaktionen müssen allerdings vom verwendeten Key-Management ,wie zum

Beispiel MIKEY, übernommen werden oder können durch einen optionalen Mechanismus in [SRTP] realisiert werden.

5.3.10 MIKEY

5.3.10.1 Überblick

Diese umfassende Lösung für die Schlüsselverteilung bei Multimedia-Systemen wird auch zukünftig noch ein Draft bleiben. Aufgrund des Umfangs haben wir beschlossen dieses Thema aussen vor zu lassen und uns ganz auf die andere Lösung mit S/MIME und SDP zu konzentrieren. Trotzdem wollen wir hier einen Blick auf MIKEY wagen und die wichtigsten Features aufzeigen.

Seit der letzten Arbeit hat der Draft einen Sprung von der Version 06 auf 07 [MIKEY] vollzogen. Die Grundlagen sind jedoch gleich geblieben.

Die Vorgaben, welche MIKEY zu erfüllen hat sind nicht ganz einfach: Es sollen Schlüssel in Echtzeit-Applikationen verteilen und verwalten, insbesondere geeignet sein für SIP und SRTP Sessions mit dem Transport-Protokoll SRTP. Dadurch muss es in Uni- wie auch Multicast-Systemen einsetzbar sein. Da es sich bei den Zielgeräten auch um eingebettete und kleine Systeme auf Telefonen handelt, muss sowohl der Ressourcenbedarf als auch der Bandbreitenbedarf möglichst minimal gehalten werden. MIKEY ist darauf ausgelegt, End-zu-End-Sicherheit zu garantieren. Schlussendlich scheint man aus den Fehlern bei IKE [IKE] gelernt zu haben und will das Protokoll möglichst einfach halten.

MIKEY ist ein Teil der [GKMARCH] Architektur. Daher stammen einige der verwendeten Begriffe aus [GKMARCH] und ebenso wird in der Spezifikation häufig darauf verwiesen.

5.3.10.2 System-Übersicht

Zuerst folgt eine Übersicht der im Folgenden verwendeten Begriffe:

CS	Crypto Session
CSB	Crypto Session Bundle
TEK	Traffic-Encrypting Key
TGK	TEK Generation Key
SA	Security Association

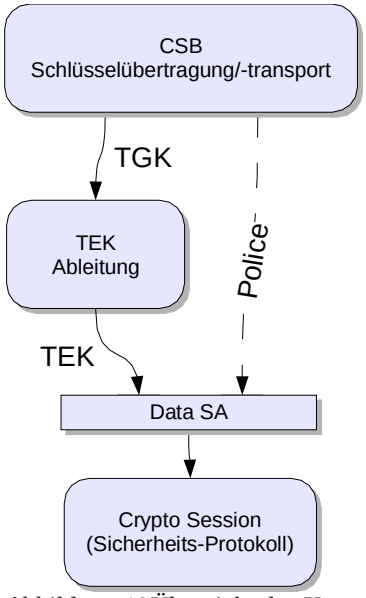


Abbildung 19Übersicht der Key Management-Prozedur

Ein Ziel von MIKEY ist das Erzeugen eines sogenanntes Data security protocol SA[i]. Eine Gruppe von Sicherheits-Parameter und TGK werden über das CSB ausgehandelt. Von den TGK werden für jede Sitzung TEK abgeleitet und bilden zusammen mit der Police die Data SA. Diese Data SA bildet die notwendigen Schlüssel und Parameter, welche für die Crypto Session[ii] über das Sicherheits-Protokoll gebraucht werden. (Siehe Abbildung 19)

Im Falle von SRTP wird aus dem so erhaltenen TEK nochmals eine Gruppe Schlüssel abgeleitet (Siehe Secure Real Time Protocol SRTP).

Zukünftig können die TEK erneuert werden, indem der ganze Vorgang nochmals abläuft. Daneben kann man sich auch auf ein paar wenige Parameter beschränken, die neu ausgetauscht werden.

i Ich versuche Data SA mit Sicherheits-Verebung zu übersetzten, d.h. eine zusammengehörende Sicherheitsgruppe, welche während einer Multimedia-Sitzung die Teilnehmer teilen.

ii Sprich: Übertragung von vertraulichen Nutzdaten

5.3.10.3 Schlüsselverteilung und -transport, CSB

Für die Schlüsselverteilung stehen drei verschiedene Varianten zur Verfügung:

- Voraus ausgetauschte Schlüssel, Pre-shared keys (PSK)
- Public-Key Verschlüsselung (PK)
- Diffie-Hellman Schlüsselaustausch (DH)

In allen drei Fällen wird neben dem TGK eine mindestens 128Bit grosse Zufallzahl und ein Zeitstempel übertragen. Bis auf die Diffie-Hellman Methode werden die TGK Schlüssel sicher verschlüsselt mit der „Push"-Methode übertragen. Im Falle der Diffie-Hellman Methode wird der TGK von den ausgetauschten Diffie-Hellman-Werten abgeleitet.

Jedes dieser Verfahren bietet natürlich sowohl Vor- als auch Nachteile:

- Pre-shared keys

  + Sehr effizient, nur symmetrische Kryptographie notwendig

  + Einfach

  - Skaliert sehr schlecht

- Public-Key encryption

  + sehr gut Skalierbar

  - Ressourcen-aufwändig

  - benötigt eine Publik-Key Infrastruktur

- Diffie-Hellman

  + ermöglicht Perfect-Forward-Secrecy

  + sehr flexibel

  - sehr Ressourcen-aufwändig, auch bei der Bandbreite

  - Lässt sich nur mit zwei Partnern einsetzen (und nicht mit Gruppen)

Es ist vorgesehen, dass nach einmaligem Schlüsselaustausch, ein symmetrischer Schlüssel zwischengespeichert wird, so dass die folgenden Schlüssel-Verteilungen mit weniger Aufwand ablaufen können.

Jeder Teilnehmer kann sowohl Server, als auch Client sein und es kommt durchaus auch vor, dass er beides gleichzeitig ist.

5.3.10.4 Kryptographische Algorithmen und Verfahren

MIKEY verwendet als kryptografische Algorithmen die „üblichen Verdächtigen":

Authentifizierung
: Hier wird das HMAC-Verfahren eingesetzt. Es können alle bei [HMAC] erwähnten Hash-Funktionen angewendent werden, Standard ist jedoch SHA-1.

Pseudo-Random-Funktion (PRF)
: Hier wird eine mit einem Key versehene PRF-Funktion im MIKEY-Draft beschrieben, welche auf einer Hash-Funktion basiert (wieder mit SHA-1 als Standard).

Transport-Verschlüsselung für die Schlüssel-Daten
: Für diese Anwendung wird AES im Counter-Mode eingesetzt, mit den gleichen Einstellungen wie bei SRTP (siehe 5.3.9 Secure Real Time Protocol SRTP). Das ist der Standard und muss vorhanden sein. Daneben werden weitere Verfahren aufgezählt, welche vorkommen können.

Unter anderem kann der Schlüssel auch nicht verschlüsselt werden, um doppelten Aufwand zu vermeiden, wenn das darunter liegende Protokoll bereits verschlüsselt. Das ist z.B. der Fall, wenn MIKEY über SIP/SDP getunnelt wird und das SDP mit S/MIME verschlüsselt ist.

MAC, Message-Digest
: Um die Integrität zu überprüfen wird wieder HMAC mit SHA-1 als Standard verwendet.

Public-Key Verfahren und Zertifikate
: Dies ist vom verwendeten Zertifikat abhängig

Zertifikate
: Hier wird nur die Verfahrensweise erklärt, implizit wird auf X.509 verwiesen. Als CLR wird vorgeschlagen, ein Protokoll wie OCSP zu verwenden.

Authorization
: Für peer-to-peer Konfigurationen gibt es nur die Möglichkeit mit Pre-shared Keys oder gegenseitig Zertifikaten zu akzeptieren.
Sonst wird eine hierarchische Architektur bevorzugt, wobei alle Teilnehmer einem CA-Zertifikat vertrauen müssen.

5.3.10.5 Meldungs-Aufbau

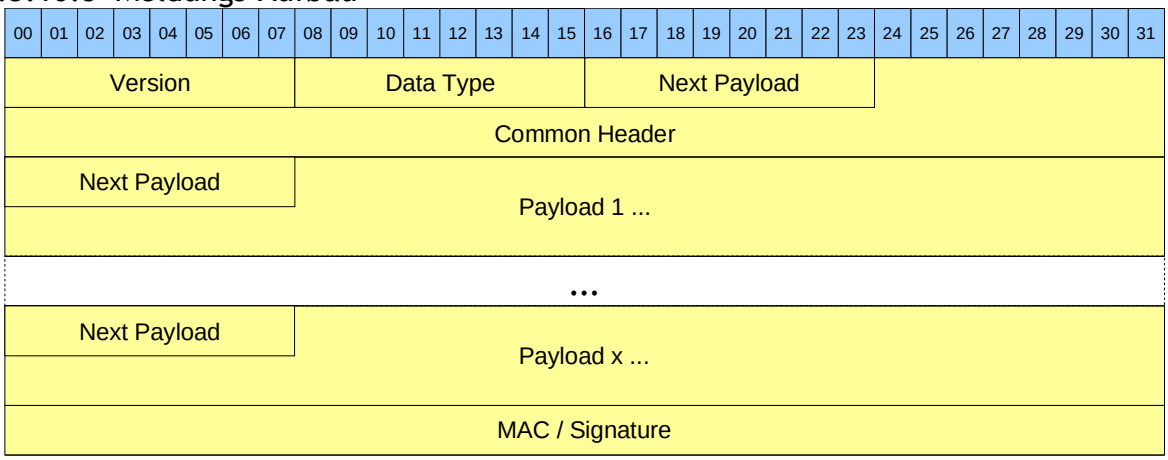


Abbildung 20 MIKEY Paketformat

Aufbau

Ein MIKEY-Paket wird immer durch einen gemeinsamen Header eingeleitet. Der folgende Inhalt wird durch ein Feld „Next Payload“ im vorangehenden Feld bestimmt. Beispiele für mögliche Inhalte sind: DH-Wert, Signatur, Sicherheits-Protokoll, usw.

Die Inhalte (Payloads) sind Byte-ausgerichtet und nicht 32-Bit-ausgerichtet.

Der MAC wird über den gesamten Inhalt, inklusive Header, aber ohne MAC/Signatur gebildet und anschliessend der MAC und die Signatur als letztes Feld hinzugefügt.

Hinweise zum Interpretieren empfangener Nachrichten

Da es sich um eine Echtzeit-Anwendung handelt, sind ein paar Überlegungen zum parsen der Meldung notwendig. Um unnötige Arbeit zu verhindern, können ein paar Optimierungen getroffen werden. So können Meldungen, bei denen der Timestamp zu alt ist, bereits aufgrund dieser Tatsache verworfen werden. Ebenso können Meldungen, dessen ID im Replay-Buffer enthalten ist, verworfen werden.

Common Header

Im Feld „Version“ steht die Version von MIKEY. Momentan gibt es nur die Version „1“. Das nächste Feld „Data-Type“ bezieht sich auf die Art der Meldung. Für jede Art des Schlüssel-austausches sind zwei verschiedene Typen vorgesehen (hin und zurück) und für Fehlermeldung ein weiterer Typ. Das folgende

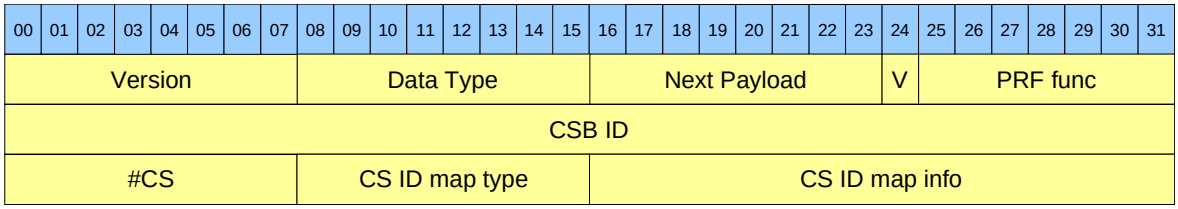


Abbildung 21 MIKEY Common-Header

Feld „Payload“, welches auch in jedem Payload vorhanden ist, zeigt wie erwähnt den nächsten Payload Typ. Momentan sind das 22 verschiedene Payloads. Das ein-Bit-Feld „V“ ist dafür da, mitzuteilen, ob eine Verifikations-Meldung erwartet wird oder nicht.Die „CSB ID“ sollte zufällig gewählt werden und für die aktuelle Kommunikation eindeutig sein. Eine Antwort trägt die gleiche „CSB ID“ wie die Anfrage. „#CS“ gibt die aktuelle Anzahl Crypto-Sessions an. Die „CS ID map type“ spezifiziert eindeutig die Methode um die Crypto-Session an das Sicherheits-Protokoll zu binden wobei die „CS ID map info“ die entsprechende Crypto-Session bezeichnet.

5.3.10.6 Integration im SDP und SIP

MIKEY ist dafür ausgelegt, innerhalb von SDP eingebeteten zu werden. Das Protokoll wird dabei immer Base64 codiert.

Wie genau sich MIKEY in SDP integrieren lässt, ist in KMASDP beschrieben. Leider ist das Dokument über die üblichen Quellen nicht mehr auffindbar. Vermutlich wird daran noch gearbeitet.

Im Falle, dass MIKEY in SIP/SDP übertragen wird, muss es dem Offer-/Answer-Model[i] von SIP folgen.

5.3.10.7 Einsatz in Gruppen

MIKEY lässt sich nicht nur bei peer-to-peer Verbindungen einsetzen, sondern ist auch dafür ausgelegt, um in Multicast-Umgebungen und Gruppen eingesetzt zu werden. Dabei kann sowohl eine zentrale Schlüsselverteilung als auch eine jeder-zu-jeder Struktur eingesetzt werden.

Soll dabei die gleichen TGK/TEK verwendet werden, so ist es notwendig, dass alle die selbe CS-ID verwenden.

i Angebot/Antwort-Model

6 Konzeptvarianten und Konzeptentscheid

Inhaltsverzeichnis

6.1 SIP Sicherheitsmechanismen

Der Abschnitt Session Initiation Protocol (SIP) ist mehrheitlich aus der Projektarbeit PA2 Sna 03/2 [PA_SIP] von Daniel Kaufmann und Andreas Stricker entnommen.

In diesem Kapitel werden die Sicherheitsmechanismen, wie sie in [RFC3261] vorgesehen sind, aufgeführt und beschrieben.

6.1.1 Basic Authentification

Um es vorweg zu nehmen: Basic Authentification ist in SIP 2 [RFC3261] nicht mehr erlaubt.

Basic Authentification war vermutlich aus dem einzigen Grund in SIP 1 [RFC2543] enthalten, weil Basic Authentification auch in HTTP/1.0 enthalten ist (genauer: [RFC2069]). Damit war die Kompatibilität in der Authentifizierung erreicht.

Mit SIP 2 [RFC3261] wird die Authentifikation nach [RFC2617] wie für HTTP/1.1 gehandhabt, inklusive der Abwärtskompatibilität zu [RFC2069]. Die Basic Authentification ist in SIP 2 jedoch nicht mehr erlaubt.

Basic Authentification hat die gleichen Schwächen wie von HTTP her bekannt: Das Passwort wird im Klartext über die Leitung gesendet.

6.1.2 Digest Authentification

Die Digest Authentification funktioniert gleich wie in HTTP/1.1 [RFC2617]. Gegenüber der Basic Authentification hat die Digest Authentification den Vorteil, dass das Passwort nicht im Klartext über die Leitung gesendet wird.

Der Client versucht zuerst einen normalen Request (**`REGISTER`**, **`INVITE`**) abzusetzen. Dieser wird vom Proxy durch ein **`407 Proxy Authentification Required`** beantwortet. Gleichzeitig enthält die Statusmeldung den Header **`Proxy-Authenticate`**, welcher den gewünschten Digest Algorithmus, eine **`nonce`**, eine **`domain`** sowie ein **`realm`** enthält. Wichtig ist hierbei die **`nonce`**, ein zufälliger Wert, der Replay-Attacken verhindert. Diese Werte werden mittels einer Digest-Funktion (MD5, SHA-1) mit dem Passwort vermischt. Dieser Wert wird dann in einem neuen **`INVITE`**/**`REGISTER`**-Request, zum Proxy-Server zurückgeschickt. Ein potentieller Angreifer kennt nun alles bis auf das Passwort. Das Passwort selber lässt sich nicht mit vernünftigem Aufwand aus dem Digest berechnen, sofern das Passwort genügend stark gegenüber Wörterbuch- und Bruteforce-Attacken ausgelegt ist.

Die Schwächen dieses Authentifikations-Verfahren liegen darin, dass einerseits das Passwort genügend stark sein muss und andererseits verhindert es nicht alle Attacken, da nur der **`INVITE`**/**`REGISTER`**-Request geschützt ist, die restlichen Pakete jedoch nicht.

Neben der Authentifikation gegenüber einem Proxy oder einem Registrations-Server ist auch die Möglichkeit vorhanden, dass sich zwei Clients identifizieren oder dass sich ein Proxy gegenüber dem Client authentifiziert. Die dafür notwendige Statusmeldung lautet **`401 Unauthorized`**.

6.1.2.1 Beispiel einer Digest Authentification

Im Folgenden ist das Beispiel aus 5.3.3.3 mit zusätzlicher Digest Authentification zu sehen. Die Nachrichten nach dem **`INVITE`** sind nicht mehr aufgeführt, da sich diese nicht mehr von dem oben aufgeführten Meldungsaustausch unterscheiden.

```
INVITE sip:6666@dskt6621.zhwin.ch SIP/2.0
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 INVITE
Max-Forwards: 70
Subject: VovidaINVITE
Contact: <sip:4444@160.85.170.139:5060>
User-Agent: Vovia-SIP-SoftPhone/0.1  (www.vovida.org)
Content-Type: application/sdp
Content-Length: 220
Session Description Protocol not shown here
```

Es handelt sich um eine normale **INVITE**-Methode, die hier noch keine Authentifikations-Elemente enthält. Anstelle des **100 Trying** erscheint nun die folgende Statusmeldung vom Proxy:

```
SIP/2.0 407 Proxy Authentication Required
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=3b6c2a3f
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 INVITE
Proxy-Authenticate: Digest algorithm=MD5,
          domain="160.85.162.81",
          nonce="1058800787",
          realm="160.85.162.81"
Content-Length: 0
```

Mit der Statusmeldung **407 Proxy Authentification Required** schickt der Server auch gleich den Header **Proxy-Authenticate**. Dieser beinhaltet das **realm**, d.h. der gültige Bereich, die **domain** und – wichtig für den Schutz gegen Replay-Attacken – eine **nonce**. Zudem ist der Digest Algorithmus spezifiziert.

```
ACK sip:6666@dskt6621.zhwin.ch SIP/2.0
Via: SIP/2.0/UDP 160.85.170.139:5060
To: 6666<sip:6666@dskt6621.zhwin.ch>;tag=3b6c2a3f
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 1 ACK
Max-Forwards: 70
Content-Length: 0
```

Der UA bestätigt die vorangegangene Statusmeldung zuerst einmal mit einem **ACK**.

```
INVITE sip:6666@dskt6621.zhwin.ch SIP/2.0
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 2 INVITE
Proxy-Authorization: Digest algorithm=MD5,
          nonce="1058800787",
          realm="160.85.162.81",
          response="142311a910a4d57ba49afdbe5646768c",
          uri="sip:6666@dskt6621.zhwin.ch",
          username="4444"
Max-Forwards: 70
Subject: VovidaINVITE
Contact: <sip:4444@160.85.170.139:5060>
User-Agent: Vovia-SIP-SoftPhone/0.1  (www.vovida.org)
Content-Type: application/sdp
Content-Length: 220
Session Description Protocol not shown here
```

Kurz darauf folgt vom UA eine Wiederholung des **INVITE**, diesmal zusätzlich mit dem gewünschten **Proxy-Authorization** Header. Dieser enthält die vom Proxy vorgegebenen Werte **nonce** und **realm**, die

URI des Anzurufenden und den **username** des UA. Der Digest dieser Werte und dem geheimen Passwort enthält die **response**.

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 160.85.170.139:5060;branch=z9hG4bK4129d28b8904
To: 6666<sip:6666@dskt6621.zhwin.ch>
From: <sip:4444@160.85.170.139>;tag=daa21162
Call-ID: 392c3f2b568e92a8eb37d448886edd1a@160.85.170.139
CSeq: 2 INVITE
Content-Length: 0
```

Diesmal erscheint das gewünschte **100 Trying**. Damit geht die SIP-Session wie bei 5.3.3.3 weiter. Die folgenden Pakete sind nicht mehr durch eine Authentifikation geschützt.

6.1.3 PGP

In [RFC2543], der ersten Version von SIP, wurde PGP vorgesehen, um die Header-Felder und den Body der SIP-Meldung zu verschlüsseln. In der aktuellen Version 2 von SIP wird PGP nicht mehr unterstützt. Stattdessen steht S/MIME als Alternative zur Verfügung.

6.1.4 S/MIME

In die Fussstapfen von PGP tritt mit SIP Version 2 die MIME-Erweiterung S/MIME [RFC2633]. SIP unterstützt von Haus aus MIME-Erweiterungen. So wird z.B. SDP als MIME-Attachment mitgeführt. S/MIME kann diese Erweiterungen schützen, indem es die Integrität sicherstellt und/oder verschlüsselt. Es ist vorgesehen, dass SIP selbst in S/MIME verwendet wird, um damit SIP über SIP zu tunneln. Der Vorteil liegt darin, dass so Modifikationen im Header entdeckt werden können.

S/MIME in SIP ermöglicht End-zu-End Identifikation und Verschlüsselung und kann so z.B. kompromittierten Proxy-Servern entgegenwirken.

Mit selbst unterschriebenen Zertifikaten ist S/MIME anfällig auf MiM-Attacken: Die einzige Möglichkeit eine solche Attacke zu erkennen bedingt, dass die Gesprächspartner den Hashwert ihrer Public-Keys validiert haben. Im Gegensatz zu anderen Protokollen, wo dieses Problem ebenfalls besteht (SSL/TLS, SSH), ist die dort übliche Praxis bei SIP nicht sinnvoll bzw. machbar: Verifikation des Hash-Wertes über das Telefon. Es bleibt somit nur der Tausch in einem vorausgehenden Treffen.

S/MIME entfaltet dort seine Stärke, wo eine Public Key Infrastruktur vorhanden ist.

Tabelle 13 zeigt die wichtigsten MIME-Typen, wie sie in SIP verwendet werden. Abhängig von dem Sicherheitsbefinden des Users eröffnet sich eine ganze Palette an Möglichkeiten:

- Verschlüsselung des MIME-Attachment:
  Bei Verwendung sicherer Übertragungsprotokolle für den Medienkanal, wie z.B. SRTP, kann der dafür notwendige Schlüssel so gesichert, innerhalb des SDP übertragen werden (z.B. **k=clear:key**).
- Signierung mittels **multipart/signed**:
  Das MIME-Attachment wird zwar offen übertragen, dennoch stellt die Signatur sicher, dass dieses nicht verändert wurde. Diese Variante ermöglicht es auch nicht S/MIME-fähigen Clients das MIME-Attachment zu lesen.
- Tunneln des SIP und Signierung:
  Ebenfalls mit dem Contentype **multipart/signed** wird der SIP Header ein zweites Mal mitgeführt und signiert. Die Spezifikation sieht vor, wenn möglich, auch das MIME-Attachment (üblicherweise SDP) hier mit zu signieren. Ebenso wird empfohlen den Header **Date** in beiden SIP Headern zu verwenden um z.B. Replay-Attacken vorzubeugen. Header, die nicht von Proxy-Server verändert werden, können nun verglichen werden, um eine Attacke zu erkennen.
- Tunneln des SIP und Verschlüsseln:
  Das Verschlüsseln des SIP-Headers macht in wenigen Fällen Sinn: Über einen Anomyzer kann der

Anrufer für Drittpersonen unerkannt den Anzurufenden kontaktieren. Im sichtbaren Header steht dann nur eine Adresse wie z.B. **sip:anonymous@anonymizer.org**. Der **From** Header der den eigentlichen Anrufer identifiziert, liegt verschlüsselt innerhalb der zweiten SIP-Meldung.

Um die oben genannten Möglichkeiten besser zu verstehen, zeigen wir nun anhand einiger Beispiele aus dem [RFC3261], wie die entsprechenden Pakete aussehen.

MIME-Type	Beschreibung
application/sdp	SDP unverschlüsselt
application/pkcs7-mime	Verschlüsseltes MIME-Attachment
application/pkcs7-signature	Signatur-Teil für ein MIME-Attachment
multipart/signed	Mehrteilige S/MIME Signatur
multipart/mixed	Für Meldungen die auch ungesicherte Teile enthalten
message/sip	SIP over SIP, für verschlüsselte SIP-Meldungen in S/MIME

Tabelle 13 Einige MIME-Typen die üblicherweise in SIP mit S/MIME verwendet werden

6.1.4.1 SDP verschlüsseln

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
   name=smime.p7m
Content-Disposition: attachment; filename=smime.p7m
   handling=required

**************************************************
* Content-Type: application/sdp                  *
*                                                *
* v=0                                            *
* o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com *
* s=-                                            *
* t=0 0                                          *
* c=IN IP4 pc33.atlanta.com                      *
* m=audio 3456 RTP/AVP 0 1 3 99                  *
* a=rtpmap:0 PCMU/8000                           *
* k=clear:f18535407369cb0aa6f34009cfc35d54       *
**************************************************
```

In diesem Beispiel wird das SDP MIME-Attachment verschlüsselt. Der zugehörige S/MIME-Typ ist **application/pkcs7-mime**. Im verschlüsselten Teil wird der übliche MIME-Typ **application/sdp** verwendet.

Beachte: Ein SIP-Client ohne S/MIME Unterstützung ist hier nicht in der Lage das Gespräch einzuleiten und wird ein **493 Undecipherable** oder ein **415 Unsupported Media Type** zurückmelden. Um ein Downgrade Attack zu verhindern, soll der User darauf aufmerksam gemacht werden, dass die Fortsetzung des Gesprächs von nun an unverschlüsselt abläuft.

Dieses Beispiel aus [RFC3261] wurde durch den SDP Parameter „k“ erweitert, der einen Schlüssel für den Medien-Kanal bereitstellt.

6.1.4.2 SIP Tunneln und Signieren

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: multipart/signed;
  protocol="application/pkcs7-signature";
  micalg=sha1; boundary=boundary42
Content-Length: 568

--boundary42
Content-Type: message/sip

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <bob@biloxi.com>
From: Alice <alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 pc33.atlanta.com
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;
   handling=required

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42--
```

In diesem Beispiel wird die SIP Meldung inklusive des SDP Attachment mit S/MIME getunnelt. Es handelt sich hier um eine reine Signierung mittels **multipart/signed**. Dabei ist, wie in der Spezifikation empfohlen, der Header **Date** in beiden Teilen präsent und weist je dieselbe Zeit auf (die Spezifikation sieht hier eine Toleranz vor).

Im ersten Teil, der durch das erste Vorkommen von **--boundary42** eingeleitet wird, ist die SIP- und SDP-Meldung enthalten. Der hierfür notwendige MIME-Typ lautet **message/sip**.

Nach dem zweiten Vorkommen von **--boundary42** folgt die Signatur. Hierfür ist der MIME-Type **application/pkcs7-signature** vorgesehen.

6.1.4.3 Zusatzbemerkung

Der Standard sieht vor, dass SIP-Meldungen inkl. MIME-Attachments nicht grösser als die MTU werden. Ist die SIP Meldung grösser, so soll statt UDP, TCP verwendet werden, um eine Fragmentierung zu verhindern. Beim Einsatz von S/MIME können entsprechend grosse Pakete auftreten.

6.1.5 TLS (SSL)

Im Gegensatz zu S/MIME bietet TLS [RFC2246] eine Hop-by-Hop Sicherheit an: Jeder Proxy muss das Vertrauen des Anwenders geniessen. Wird ein Proxy kompromittiert, ist die Sicherheit verloren.

Dennoch bietet SSL/TLS einen guten Schutz vor vielen Attacken. Die Integration von SSL/TLS ist relativ einfach, und so wundert es nicht, dass dies viele SIP-Server und -Clients unterstützen.

Ein weiterer Nachteil kommt daher, dass SSL/TLS verbindungsorientiert arbeitet und aus diesem Grund nicht mit UDP funktioniert. SIP ist zwar sowohl bei UDP als auch bei TCP zu Hause; leider führt TCP zu langen Timeouts und für den Verbindungsaufbau müssen zuerst zwei Pakete verschickt werden.

Damit die Pakete auch den Rückweg wieder verschlüsselt passieren, wird mit dem Header **`Recorde-Route`** und **`Via`** der Weg durch die Proxys festgelegt.

Es kann durchaus sein, dass zwischen einzelnen Hops keine Verschlüsselung eingesetzt wird. Der Empfänger kann das durch eine Analyse der **`Via`** und **`To`** Header feststellen.

Analog zu den Protokollen, die mit SSL/TLS harmonieren ist auch für SIP eine „S"-Variante für die URI vorgesehen. Die SIPS-URI weist auf eine mit SSL/TLS verschlüsselte Verbindungsaufnahme hin. Siehe dazu auch Kapitel 5.3.3.3.

6.2 MIKEY

MIKEY wurde bereits im Kapitel 5.3.10 vorgestellt.

6.3 IPsec

Einen Punkt, den wir[i] an der Projektarbeit [PA_SIP] bereits getestet haben, ist die Verwendung von Ipsec, um den Kommunikations-Kanal abzudichten. Wenn genügend Bandbreite vorhanden ist[ii], eignet sich IPsec sehr gut für die Absicherung des Kommunkations-Kanals.

IPsec ist stark verbreitet, dadurch ist diese Lösung überlegenswert. Wenn eine bestehende CA für S/MIME existiert, kann dadurch die CA für IPsec entfallen, sofern man sich nur auf die Anwendung für VoIP beschränkt. Über SIP/SDP kann ein symmetrischer Schlüssel für eine dynamische IPsec-Verbindung ausgetauscht werden. Schlussendlich muss der Client dafür eine Schnittstelle bieten. Leider muss das standardisiert werden, damit es auch zwischen verschiedenen Clients funktioniert.

i Andreas Stricker und Daniel Kaufmann aus [PA_SIP]
ii IPsec hat eine etwas grössere Anforderung an Bandbreite als SRTP

6.4 Stacks

6.4.1 SIP-Stack

Hersteller/Projekt	reSIProcate	oSIP	Vovida/Vocal	dissipate
Version / RFC	SIP 2 / RFC 3261	SIP 2 / RFC3261	SIP 1 / RFC 2543	SIP 1 / RFC 2543
S/MIME	Ja	Nein	Nein	Nein
Programmiersprache	C++	C	C++	C++
Projektstatus / -aktivität	Stark aktives Projekt	zögerlicher Fortschritt	Die Entwicklung geht weiter voran	kein Fortschritt
Verwendende Anwendungen / Verfügbare Clients	SipIMP (Test-Applikation)	Jusua, Linphone	SIPSet	Kphone
Codeumfang	ca. 60'000 Zeilen	ca. 28'500 Zeilen	ca. 71'000 Zeilen	ca. 11'000 Zeilen
Link	resiprocate.org	osip.org	vovida.org	Kein offizieller Link vorhanden.

Tabelle 14 Untersuchte SIP-Stacks

Es gibt mehrere Projekte die einen SIP Stack zur Verfügung stellen. In der Tabelle 14 sind die verschiedenen Möglichkeiten aufgelistet und Vergleichspunkte aufgelistet. Die Entscheidung, welchen Stack zu verwenden, beruht schlussendlich auf diesen Kriterien. Der Stack sollte auf der SIP Version 2 [RFC3261] aufbauen und nicht auf Version 1 [RFC2543]. Damit blieben bereits nur noch zwei Projekte übrig. Unter diesen beiden Möglichkeiten sprach eindeutig mehr für den Stack von reSIProcate. So sollte es ein Stack sein, der in C++ und nicht in C geschrieben ist. Auch der Fortschritt des Projektes sprach eindeutig für diesen Stack. Während bei oSIP unklar war, wie stark weiterentwickelt wird, war bei reSIProcate gleich ersichtlich, dass ständig weiterentwickelt und getestet wird. Dieser Umstand war ein wichtiges Entscheidungskriterium und sollte sich auch als sehr vorteilhaft herausstellen. Es gab täglich Fragen, Antworten und Informationen über die Maillingliste zum aktuellen Projekt. Es wurde auch immer umgehend auf diverse Anfragen reagiert, das Problem verfolgt und meist gelöst. Ein weiterer Vorteil war, dass bereits S/MIME implementiert ist und daher Funktionen zur Verfügung stehen. Leider mussten wir am Ende unserer Tests feststellen, dass es momentan noch ein Problem mit dem Parser im Umgang mit S/MIME gibt.

6.4.2 SRTP-Stack

Wie bereits erwähnt, wird SRTP in [SRTP] definiert und es handelt sich hierbei um einen 'Internet Draft' der IETF. Da es sich bei internet draft's vor allem um Publikationen handelt die sich erst in der Entwicklung befinden und dadurch noch Änderungen unterworfen sein können, existiert nur eine Implementation die als Referenz dienen kann.

Auch für SRTP gibt es keine grosse Auswahl an Open Source Implementationen, wie dies zum Beispiel bei SIP der Fall ist. Bei unserer Suche nach einem SRTP Open Source Stack mussten wir zur Kenntnis nehmen, dass gerade mal ein frei verfügbarer SRTP-Stack existiert. Es handelt sich hierbei um den Stack des Projektes libsrtp unter sourceforge.net, welcher in Kapitel 7.1.2 SRTP-Stack libsrtp genauer beschrieben wird.

6.4.3 SIP-Client

Client	Kphone	linphone	SipIMP	SIPSet
Version	SIP 1	compliance with RFC3261 (SIP2)	SIP 2	SIP 1
Programmiersprache	C++	C	C++	C++ / GUI in Java
Codecs	G.711	G.711, GSM, LPC10-15		G.711
Projektstatus / -aktivität	Starke Aktivitäten des Projektes	Nicht genau bekannt.	Test-Applikation für den Stack reSIProcate	Nicht genau bekannt.
Besonderheit	Video calls			
Verwendete Stacks	dissipate	oSIP	reSIProcate	Vovida / Vocal
S/MIME	Nein	Nein	Nein	Nein
Link:	wirlab.net/kphone	linphone.org	resiprocate.org	vovida.org

Tabelle 15 Untersuchte SIP-Clients

Es gibt diverse SIP-Clients auf dem Markt, die bereits ziemlich stabil sind. Die bereits bestehenden Clients haben einen solch grossen Umfang, dass zum Teil mehr abgedeckt wird, als es für die Aufgabenstellung dieser Arbeit gefordert war. Die Aufstellung zeigt aber auch, dass bisher kein Client S/MIME implementiert.

Weiter baut der einzige Client (kPhone), bei dem regelmässig Aktivitäten und Weiterentwicklungen zu erkennen sind, nur auf der SIP Version 1 auf. Bei den anderen Projekten sind nur geringe oder keine weiteren Änderungen zu erwarten. Leider ist meistens der Source Code nicht oder nur sehr dürftig kommentiert, so dass es sehr mühsam ist sich im Code zu orientieren. Dies ist ein weiterer Grund, weshalb wir uns schlussendlich dazu entschlossen haben, einen eigenen Client zu entwickeln. Diese Entscheidung machte es möglich, nur die benötigten Komponenten einzubinden. Weiter musste keine zusätzliche Zeit aufgewendet werden, um den Client zu interpretieren und nach eigenen Bedürfnissen zu erweitern.

6.5 Fazit

Im Vergleich der SIP-Sicherheitsmechanismen ist deutlich ersichtlich, dass S/MIME viele Vorteile gegenüber den anderen Varianten besitzt. S/MIME ist flexibel, weit verbreitet, bietet mehrere Möglichkeiten um die Forderungen an die Sicherheit zu erfüllen und bietet eine End-zu-End-Sicherheit an.

Es gibt mehrere frei verfügbare SIP-Stacks. Die SIP-Stacks unterscheiden sich sehr stark im Funktionsumfang und auch darin, mit welchem Einsatz an der weiterentwickelt wird. Bei der Gegenüberstellung steht der SIP-Stack reSIProcate ganz klar als Sieger fest. Der Stack verfügt schon über S/MIME-Unterstützung, ist in C++ geschrieben und wird ständig weiterentwickelt und vorangetrieben.

Das Secure Real-time Transport Protocol (SRTP) befindet sich erst in der Entwicklung, dennoch gibt es schon einen frei verfügbaren Stack, der SRTP implementiert. Das letzte offizielle Release liegt bereits über ein ganzes Jahr zurück und entspricht nicht mehr ganz der aktuellen Spezifikation. Die Arbeit am Stack geht aber weiter voran und ein neue Version sollte in nicht mehr in allzu weiter Ferne liegen.
Für SRTP wird ein symmetrischer Schlüssel benötigt. MIKEY wurde für diesen Zweck betrachtet, aber aus folgenden Gründen nicht gewählt:

- aufwändige Implementation
- frühes Entwicklungsstadium

Um nun dennoch in den Besitz eines symmetrischen Schlüssels zu kommen, bietet sich die Nutzung von SDP an und wurde von uns auch ausgewählt.

Neben den Stacks mussten wir auch eine Entscheidung bezüglich des Clients treffen. Wie bei den SIP-Stacks hat man eine grosse Auswahl. Die meisten Clients sind stabil und auch reich an Funktionen. Die Unterstützung für S/MIME ist in keinem Client integriert.

7 Realisierung

Inhaltsverzeichnis

7.1 Bibliotheken

7.1.1 SIP-Stack reSIProcate

7.1.1.1 Übersicht

Eine vom Vovida SIP-Stack abgesplittete Gruppe von Entwicklern hat das Projekt reSIProcate ins Leben gerufen. Einige der Entwickler sind im Standard-Gremium an der Entwicklung der SIP-Spezifikation beteiligt, andere an der Entwicklung von weiteren Stacks (Vocal und kommerzielle Stacks). Sie geben an, ihr Ziel bei reSIProcate sei ein moderner SIP/2.0 [RFC3261] Stack, der vieles besser macht, als die bisherigen Stacks.

Der Stack ist voll kompatibel zum [RFC3261]. Unterstützung für weitere RFC's ist im Gange und teilweise schon erledigt (wie z.B. [RFC3263]). An Plattformen unterstützt er diverse Unix-Varianten und Windows. Der Stack steht unter der BSD-ähnlichen Lizenz von Vovida. Da er mittlerweile keinen Code von Vovida mehr enthält, läuft zurzeit eine Diskussion, ob nicht zur BSD-Lizenz gewechselt werden soll. Der Stack ist in C++ geschrieben.

Momentan ist der Stack trotz beachtlichem Umfang immer noch in der frühen Entwicklungsphase und hat soeben Release 0.3.0 erreicht.

7.1.1.2 Bezugsmöglichkeit

Der Stack ist unter einer eigenen Domain im Internet zu finden, die Entwicklung läuft aber über Sourceforge. Deshalb führen folgende, die beiden folgenden Domains zum Ziel:

`http://www.resiprocate.org`

`http://resiprocate.sourceforge.net`

Neben den sporadisch erscheinenden Releases im tar.gz-Format sind die Quelltexte auch über das Sourceforge-CVS zu beziehen:

`pserver:anonymous@cvs.sourceforge.net:/cvsroot/resiprocate`

7.1.1.3 Aufbau

Um den Stack zu verstehen, muss momentan der Source-Code studiert werden. Kommentar ist nur an besonders wichtigen Stellen vorhanden und die Dokumentation ist sehr kurz gefasst und Lückenhaft. Aus diesem Grund sind die mitgelieferten Beispiele[i] sehr hilfreich um die Funktion zu verstehen.

Transport-Protokolle

Der Stack horcht an den hinzugefügten und **`Transports`** genannten Schnittstellen. Momentan verfügbar sind TCP, UDP und TLS. Empfängt der Stack eine Meldung, wird diese soweit nötig interpretiert und im Eingangsbuffer abgelegt.

i SIPImp und Limpc sind einfache Instant-Messenger, die Tests sind zusätzlich auch hilfreich

Lazy Parser

Um den Verarbeitungs-Aufwand so gering wie möglich zu halten, verwendet der Stack einen Lazy Parser: Nur wenn der Wert eines Header-Feld benötigt wird, wird dieser auch geparsed.

Um die einzelnen Header auseinander zu halten, sucht der Parser die Zeilenenden und zerlegt die Meldung an diesen Stellen. Bei der Suche nach einem bestimmten Header muss er nur den Header-Typ am Anfang der Zeile ermitteln, der Rest bleibt unbearbeitet, bis darauf zugegriffen wird.

Benutzerschnittstelle

Der Benutzer des Stack arbeitet hauptsächlich mit zwei Klassen des Stacks: **`SipStack`** und **`SipMessage`**. Die zweite werde ich im folgenden Abschnitt erläutern.

Der **`SipStack`** wird instanziert, wobei hinter der Kulisse der gesammte Stack initialisiert wird. Allerdings ist er noch nicht in der Lage Daten zu senden oder zu empfangen: Dazu muss mindestens ein **`Transport`** hinzugefügt werden.

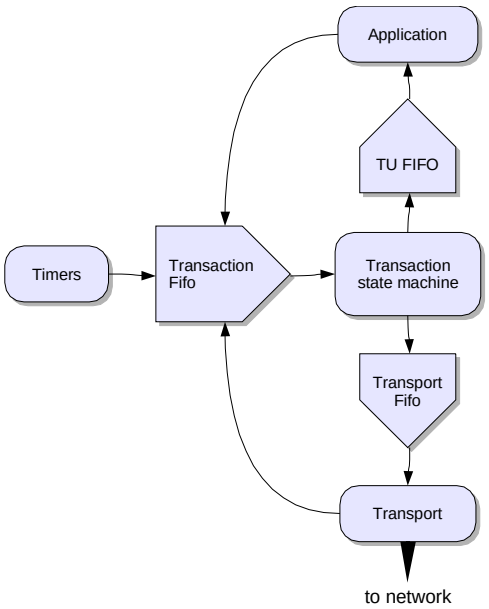


Abbildung 22 Struktur des reSIProcate Stacks

Von jetzt an können Meldungen über die Funktion **`SipStack::send()`** versendet werden.

Wenn der Stack zerstört wird (üblicherweise am Ende des Programms), so sind zwei Punkte zu beachten: Vor dem Aufruf des Destruktors sollte dem Stack das baldige Ende über den Aufruf der Funktion **`SipStack::shutdown()`** mitgeteilt werden. Von jetzt an nimmt er keine weiteren Meldungen mehr an. Wenn der Stack intern aufgeräumt hat, schickt er eine Meldung **`ShutdownMessage`** ab. Nach dem Empfang dieser Meldung kann der Konstruktor aufgerufen werden.

SipMessage

Die gesendeten und die empfangenen Meldungen werden mit der Klasse **`SipMessage`** gehandhabt.

Zugriff auf die einzelnen Felder bietet die Funktion **`SipMessage::Header()`**. Die Funktion ist mehrfach überladen, für jeden Header-Type, RequestLine und auch StatusLine.

An den MIME-Inhalt kommt man über **`SipMessage::getContents()`** und setzt den Inhalt über **`SipMessage::setContents()`**. Ob ein Header überhaupt existiert, kann mit **`SipMessage::exists()`** abgefragt werden. Um eine empfangene Meldung grob einzuteilen, kann mit **`SipMessage::isRequest()`** und **`SipMessage::isResponse()`** abgefragt werden ob es sich um eine Anfrage oder um eine Antwort handelt.

Hilfs-Klassen

Die Generierung von Meldungen erleichtert die **`Helper`**-Klasse, indem sie eine Factory für einige, wichtige Meldungs-Typen bietet.

Für einen typischen Dialog nimmt die Klasse **`Dialog`** viel Arbeit ab, indem sie sich die Session-Informationen merkt und daraus die entsprechenden Antworten generieren kann.

SIP-URI werden mit der **`Uri`**-Klasse bearbeitet. Diese liefert einerseits die einzelnen Teile der URI, wie Schema (**`sip`** oder **`sips`**), User, Host und Port, und andererseits ist sie auch in der Lage eine als String übergebene URI zu parsen.

Security

Die Klasse Security übernimmt fast alle Aufgaben, die für die Sicherheit-Features notwendig sind. Intern

greift diese Klasse auf OpenSSL-Funktionen zu. Die einzelnen Features dieser Klasse:

- Laden und speichern von Zertifikaten, Private- und Public-Keys in verschiedenen Formaten (PEM, PKCS7, PKCS12)
- Verschlüsseln und Entschlüsseln von S/MIME signierten MIME-Attachments
- Überprüfen der digitalen Signatur von S/MIME-Attachments
- Bereitstellen der Zertifikate für die TLS-Verbindung

SDP

Für die Behandlung von SDP in MIME-Anhängen bietet der Stack die Klasse **`SdpContents`**, welche einen vollwertigen SDP-Parser beinhaltet. Diese Klasse integriert eine Klasse **`SdpContents::Session`**, welche wiederum Klassen für jedes SDP-Feld beinhaltet. Ein Beispiel: Über **`SdpContents::Session::Medium`** erhält man Zugriff auf das Übertragungs-Medium.

7.1.2 SRTP-Stack libsrtp

7.1.2.1 Informationen

Es wurde bereits erwähnt, dass nur ein freiverfügbarer SRTP-Stack existiert. Es handelt sich hierbei um den Stack des Projektes libsrtp Dieser Stack ist unter sourceforge.net erhältlich und kann unter folgendem Link bezogen werden:

`http://sourceforge.net/projects/srtp/`

Zum Zeitpunkt der Diplomarbeit liegt die Implementation, welche in der Programmiersprache C geschrieben ist, in der Version 1.0.6 vor und befindet sich in einem frühen Stadium der Entwicklung. Die Version 1.0.6 wurde am 21.Juli 2002 veröffentlicht und ist konform zur 4. Auflage des Draft's [SRTP]. Da sich aber sehr viel am Draft geändert hat, ist der Stack zur aktuellen Version nicht mehr konform.

Der Leistungsumfang des SRTP-Stacks und der Fortschritt des Projekts hält sich in Grenzen. Zum jetzigen Zeitpunkt kümmert sich gerade ein Entwickler um das Projekt. Es handelt sich hierbei um Herrn David A. McGrew, welcher auch an der Spezifizierung von [SRTP] beteiligt ist und für Cisco arbeitet. Der Stack wird durch David A. McGrew bei Cisco entwickelt und ist auch mit einer entsprechenden Open Source Linzenz und dem Copyright von Cisco Systems versehen. Mit diesem Entwickler sind wir bezüglich einer aktuelleren Version des Stacks in Kontakt getreten. Nach den Aussagen des Entwicklers existiert eine neue Version mit bedeutenden Verbesserungen und Änderungen, welche auch den aktuellen Draft unterstützt. Diese Version ist aber für eine offizielle Veröffentlichung noch nicht bereit, da auf den Plattformen Linux/x86 und Solaris Probleme auftreten.

Wie bereits angesprochen, hält sich der Funktionsumfang des SRTP-Stacks in Grenzen. Doch ist bereits die bestehende Funktionalität dokumentiert. Mit dem Stack werden zusätzlich einige, kleine Applikationen zum Testen des Stacks mitgeführt, welche für das Verständnis und die richtige Verwendung sehr hilfreich sind. Der Stack 1.0.6 umfasst schon folgende Funktionen:

- AES-128 in counter mode (AES-CM)
- NULL cipher
- tmmh Version 2 (Algorithmus für die Authentifizierung)
- Re-Play protection

Folgende Funktionen sind noch nicht implementiert, sind aber nach den Angaben des Entwicklers bereits in der noch nicht veröffentlichten Version enthalten oder für eine spätere Version vorgesehen:

- AES-128 in f8 mode (AES-f8)
- HMAC-SHA-1

- Re-keying mit Hilfe des Merkmals der Schlüsselbildung
- SRTCP

7.1.2.2 Aufbau

Der Stack ist in der Programmiersprache C geschrieben und ist somit nicht objektorientiert. Durch den kleineren Umfang als beim reSIProcate-Stack, kann viel schnell ein Überblick gewonnen werden.

Es ist wichtig zu wissen, dass für die Nutzung des Stacks noch relativ viel Verwaltungsaufwand selber geleistet werden muss. So muss zum Beispiel selber ein Socket erstellt und gebunden werden; diese Aufgabe wird nicht automatisch vom Stack übernommen.

Strukturen

Mit Hilfe von den C-Strukturen wird die Arbeit mit dem Stack erleichtert. Zu den wichtigsten Strukturen gehören dabei die folgenden:

`srtp_ctx_t`
: Diese Struktur enthält alle Informationen, die für eine SRTP-Session benötigt werden und bildet somit den Kontext der SRTP-Verbindung. In dieser Struktur sind die verwendeten Algorithmen für die Verschlüsselung und Authentifizierung enthalten. Weiter wird hier spezifiziert, welche Sicherheitsmerkmale angewendet werden sollen.

`rtp_receiver_t`
: Enthält alle wichtigen Informationen für den Empfänger. In dieser Struktur sind der Socket, die oben genannte Struktur **`srtp_ctx_t`** und auch die nötigen Informationen für den erfolgreichen Empfang enthalten. Zu den Informationen gehören hierbei die lokale Adresse, der lokale Port und das verwendete Netzwerk- und Transport-Protokoll.

`rtp_sender_t`
: Diese Struktur enthält alle wichtigen Informationen für den Sender und ähnelt sehr stark der Struktur **`rtp_reciever_t`**. **`rtp_sender_t`** enthält ebenfalls einen Socket, die bereits genannte Struktur **`srtp_ctx_t`** und weitere Informationen für das erfolgreiche Versenden von Daten. Diese Informationen enthalten die Adresse und den Port des Empfängers, sowie das benutzte Netzwerk- und Transport-Protokoll.

Funktionen

Die Funktionen im SRTP-Stack realisieren die eigentliche Funktionalität des Stacks. Der Stack besitzt sehr viele interne Funktionen. Für den Gebrauch des Stacks sind für uns nur eine kleine Auswahl von diesen Funktionen wichtig.

`rtp_receiver_init`
: Wie es der Name der Funktion schon andeutet, wird mit Hilfe dieser Funktion die Struktur rtp_receiver_t initialisiert.

`srtp_receiver_init`
: Mit Hilfe dieser Funktion wird die Initialisierung der Struktur **`rtp_receiver_t`** fortgesetzt. Die nötigen Informationen für die weitere Initialisierung umfassen die gewünschten Sicherheitsmerkmale und einen Hex-String mit 60 Hex-Zeichen Die 60 Hex-Zeichen entsprechen dabei 240 Bit. Diese 240 Bit entsprechen dem Master-Key und dem Master-Salt. 128 Bit sind dabei für den Master-Key vorgesehen und die restlichen 112 Bit werden für das Master-Salt verwendet.

`rtp_sender_init`
: Diese Funktion initialisiert die Struktur **`rtp_sender_t`**.

srtp_sender_init

Mit Hilfe dieser Funktion wird die Initialisierung der Struktur rtp_sender_t fortgesetzt und benötigt die gleichen Informationen wie die Funktion srtp_receiver_init.

rtp_recvfrom:

Mit dieser Funktion werden die eigentlichen Daten empfangen. Die Daten werden automatisch beim Empfang entschlüsselt und überprüft.

srtp_sendto:

Diese Funktion ermöglicht es die Daten zu versenden. Die gewünschten Sicherheitsmerkmale werden automatisch auf die Daten angewendet.

7.1.3 Posix-Threads pthread

7.1.3.1 Informationen

Im Posix-Standard ist diese Art von Threads beschrieben. Diese bieten ein gut dokumentiertes API und es handelt sich um Kernel-Threads[i]. Ein weiterer Vorteil der Posix-Threads ist die gute Portierbarkeit.

Wir verwenden pthreads und den pthread-Mutex-Mechanismus um den gegenseitigen Zugriff auf gemeinsame Speicherbereiche zu verhindern.

Die pthread-Bibliothek sollte bei jeder GNU/Linux-Distribution vorhanden sein.

7.1.4 C++ pthread Wrapper SafePt

7.1.4.1 Informationen

Übersicht

Bei dieser Bibliothek handelt es sich um einen sehr einfach gehaltenen C++-Wrapper um die pthread Bibliothek. Der Einsatz von pthreads in einem C++-Programm wird dadurch sehr einfach.

Bezugsmöglichkeit

SafePt ist unter der unten aufgeführten Addresse zu finden. Neben dem Tarball besteht auch die Möglichkeit, die aktuelle Version aus dem CVS zu beziehen.

http://pmade.org/software/safept/

7.1.4.2 Anwendung

Ein Thread kann über ein Template erzeugt werden und die **Mutex**-Klasse sowie die darin enthaltene **Mutex::Lock-**Klasse erlauben das Locking mit einer Zeile, freigegeben wird automatisch im Destruktor. Ebenso einfach lassen sich auch die pthread Conditions einsetzen. Verwiesen wird hier auf die Beispiele und die API-Dokumentation, die sehr gelungen sind.

7.1.5 OpenSSL

7.1.5.1 Informationen

Übersicht

OpenSSL ist bei vielen Open Source Software-Paketen in Verwendung. Neben der Haupt-anwendung, welche schon der Name suggeriert - SSL- und TLS-Verschlüsselung - enthält es unzählige, kryptographische Funktionen für verschiedene Anwendungszwecke. Zum Beispiel kennt es auch

i Unter Linux handelt es sich um sogenannte Light-Weight-Prozesse, da Linux keine schlanken Threads kennt

S/MIME. Neben der Bibliothek wird auch ein Kommando-Zeilen-Tool mitgeliefert, über welches fast jede Funktion erreichbar ist. So ist es möglich, über dieses Frontend eine eigene CA zu verwalten.

Bezugsmöglichkeiten

Openssl hat eine eigene Projekt-Homepage die unter folgender URL erreichbar ist:

`http://www.openssl.org/`

7.2 Implementierung

7.2.1 Threads

Unser SIPSec-Client besitzt mehrere Aufgaben, welche quasi parallel ablaufen müssen. Für diesen Zweck bietet sich die Verwendung von Threads an. Verwendet wird der in Kapitel 7.1.4 C++ pthread Wrapper SafePt beschriebenen Wrapper um die Bibliothek 7.1.3 Posix-Threads pthread einfacher nutzen zu können.

Im Ganzen besitzt der Client vier unabhängige Threads, die Ihre Aufgaben parallel wahrnehmen können. Der Client setzt sich aus dem Main-Thread, dem SIP-Thread, dem Sender- und Receiver-Thread zusammen. Neben dem Vorteil, dass mehrere Aufgaben quasi gleichzeitig ablaufen, wird durch die Aufteilung der Funktionalität auf die vier Threads, auch die Struktur des Clients sehr vereinfacht.

Die Threads werden als Klassen definiert, welche den Funktionsaufruf-Operator deklarieren. Der Funktionsaufruf-Operator stellt praktisch die Main-Methode für den jeweiligen Thread dar.

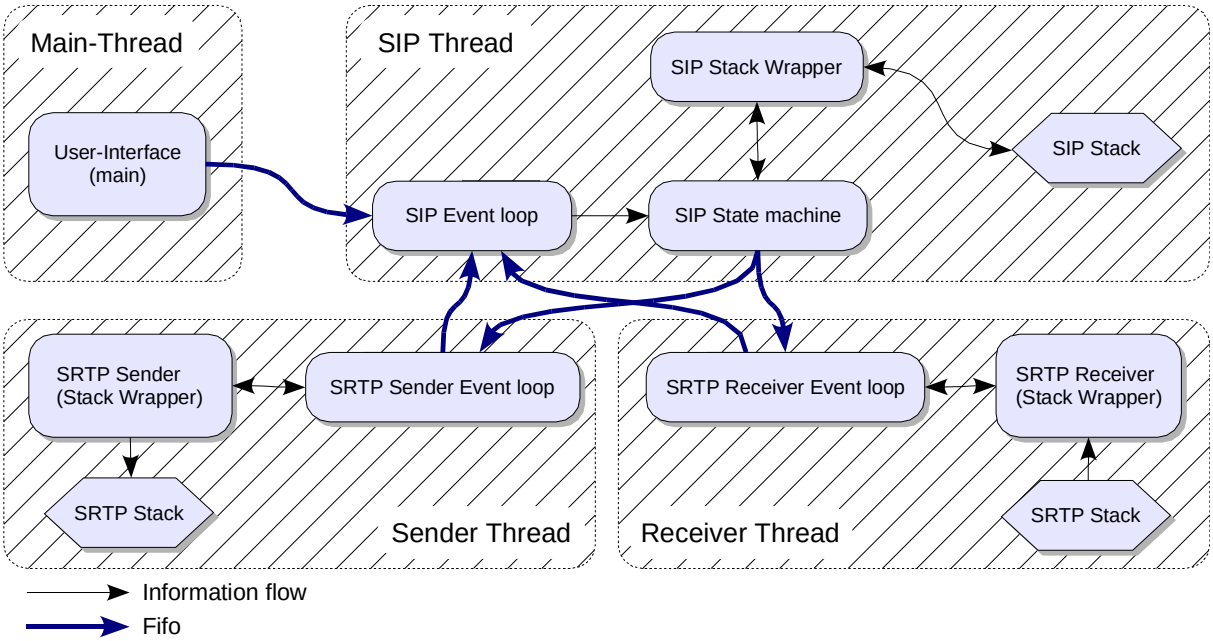


Abbildung 23 Architektur des SIPSec-Clients

7.2.1.1 Main Thread

Die Main-Funktion stellt einen eigenen Thread dar Im Main-Thread läuft das User-Interface realisiert, welches die Ausgabe der Informationen für den Benutzer darstellt und Befehle für die Steuerung des Clients entgegen nimmt. Hier werden die weiteren Threads erzeugt und gestartet.

7.2.1.2 SIP-Thread

Dieser Thread beinhaltet die SIP-Zustands-Maschine (FSM) und nutzt mit Hilfe des SIP-Wrappers die Funktionalität des SIP-Stacks reSIProcate. Nach der Initialisierung des Threads geht er in einen Event-Loop, in welchem er auf Events der anderen Threads wartet. Empfängt er ein Event über einen Fifo, wird dieses Event zur Verarbeitung an die SIP-FSM weitergereicht.

7.2.1.3 Sender-Thread

Mit Hilfe des Sender-Threads werden die Daten versendet. Für diesen Zweck nutzt er eine Instanz der Klasse **`SRTPSender`** und damit indirekt den SRTP-Stack. Die erste Aufgabe besteht darin den Sender beziehungsweise den SRTP-Stack zu initialisieren. Der Thread erwartet darum ein entsprechendes Event vom SIP-Thread. Nach der Initialisierung wartet der Thread bis zu seinem Ende auf Events, die Daten

zum Versenden enthalten.

7.2.1.4 Receiver-Thread

Der Aufbau des Receiver-Threads ist relativ einfach. Er enthält ein Objekt der Klasse **`SRTPReceiver`**, über welches er die Funktionalität des SRTP-Stacks nutzt. Vor dem Empfangen von Nachrichten muss der Receiver mit den nötigen Parametern initialisiert werden. Für diesen Zweck wartet der Receiver-Thread auf die Informationen vom SIP-Thread. Ist der Receiver einmal initialisiert, verbringt der Receiver-Thread seine Zeit damit, Nachrichten vom Receiver zu empfangen und zum SIP-Thread weiterzuleiten. Der Thread ruft zum Empfangen die blockierende Methode **`receive`** auf.

7.2.1.5 Fifo

Die Kommunikation zwischen den Threads wird mit Hilfe von Events realisiert, welche über die **`Fifo`**s versendet und empfangen werden. Bei der Kommunikation zwischen Threads gelten spezielle Anforderungen an die Synchronisation und den gegenseitigen Ausschluss, welche durch die Container aus der C++ Standard-Library nicht erfüllt werden können. Aus diesem Grund musste ein eigener Fifo entwickelt werden, welcher diese Anforderungen erfüllen kann.

Die erste Anforderung besteht darin, dass ein Thread, der ein Event empfangen will und einen leeren **`Fifo`** vorfindet, sofort blockiert. Die Blockierung soll beim ersten eingefügten Event wieder aufgehoben werden.

Zweitens dürfen sich nie zwei Threads gleichzeitig auf den Fifo zugreifen und den Inhalt der gekapselten Queue bearbeiten. Verändert ein erster Thread den Inhalt des Fifos, müssen alle weiteren Threads vom Versuch, auf den **`Fifo`** zuzugreifen, abgehalten werden. Verlässt der erste Thread den Fifo wieder, dürfen die blockierten Threads weiterarbeiten.

Der **`Fifo`** kapselt dabei intern eine Queue aus der C++ Standard-Bibliothek und steuert den Zugriff auf diese Queue. Um den Zugriff der Threads zu steuern, werden die Mutex- und Kondition-Mechanismen von SafePt verwendet und schützen somit die interne Queue.

7.2.2 Zustandsmaschine

7.2.2.1 Prinzip

Für die Implementation des SIP-Clientswurde eine allgemein gestaltetes Framework für eine Zustandsmaschine entwickelt.

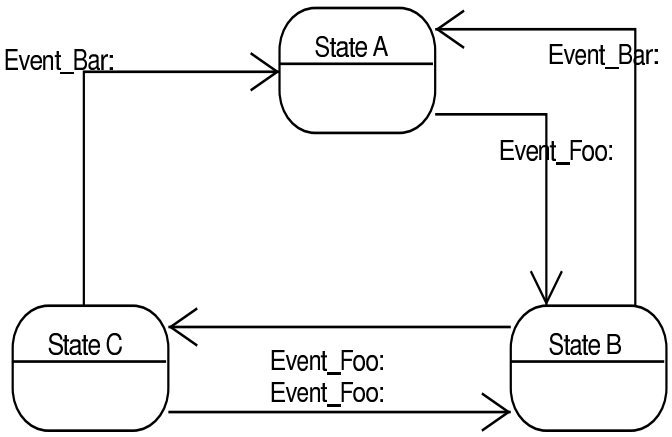


Abbildung 24 Beispiel einer einfachen Zustandsmaschine

Für jeden Zustand wird eine eigene Klasse instanziert. Jeder Übergang verfügt über eine Operation. Jede Operationen ist als eigene Klasse implementiert. Anlass für den Wechsel des Zustandes ist das Eintreffen eines Ereignisses (Events). Diese Events werden von einer übergeordneten **`Event`**-Klasse abgeleitet. Die Events werden nur durch den Typ unterschieden, denn die virtuelle Methode **`getEventType()`** zurückliefert.

Wenn ein Event ausgelöst wird und im momentanen State abgehandlet wird, so wird die **`process()`** Funktion vom zuständigen Operator aufgerufen. Falls das auftretende Ereignis nicht im momentanen Zustand deklariert ist, so wird es ignoriert. In der Funktion **`process()`** werden die notwendigen Operationen abgehandelt und ausgeführt.

Für das bessere Verständnis soll die Abbildung 24 und das anschliessende Beispiel helfen, welches drei Zustände (A, B und C) mit zwei möglichen Events (Foo und Bar) beinhaltet. Implementiert wird alles,

indem man die drei Zustandsklassen, die zwei Operatorklassen und ein generelle Klasse die alle Events definiert, ableitet.

Erzeugen der neuen State's und Operation's:

```
State *A = collectState(new StateA(this));
State *B = collectState(new StateB(this));
State *C = collectState(new StateC(this));
Operation *opMuahh = collectOp(new OperationMuahh());
Operation *opDoh = collectOp(new OperationDoh());
```

Den Zuständen werden Event-Handler hinzugefügt: Die deklarieren genau, auf welches Event reagiert, welche Operation ausgeführt und in welchen State gewechselt werden soll.

```
A->addEventHandler(Event_Foo, *opMuahh, *B);
B->addEventHandler(Event_Bar, *opDoh, *A);
B->addEventHandler(Event_Foo, *opMuahh, *C);
C->addEventHandler(Event_Foo, *opMuahh, *B);
C->addEventHandler(Event_Bar, *opDoh, *A);
```

Wenn man sich nun besipielsweise im Zustand A befindet und sich ein **`Event_Foo`** ereignet, so wird die **`OperationMuahh()`** ausgeführt und anschliessend geht die Maschine in Zustand B über. Sollte im Zustand A ein **`Event_Bar`** ausgelöst werden, so wird dieses nicht abgehandelt und man verbleibt im aktuellen Zustand.

7.2.2.2 Implementierung

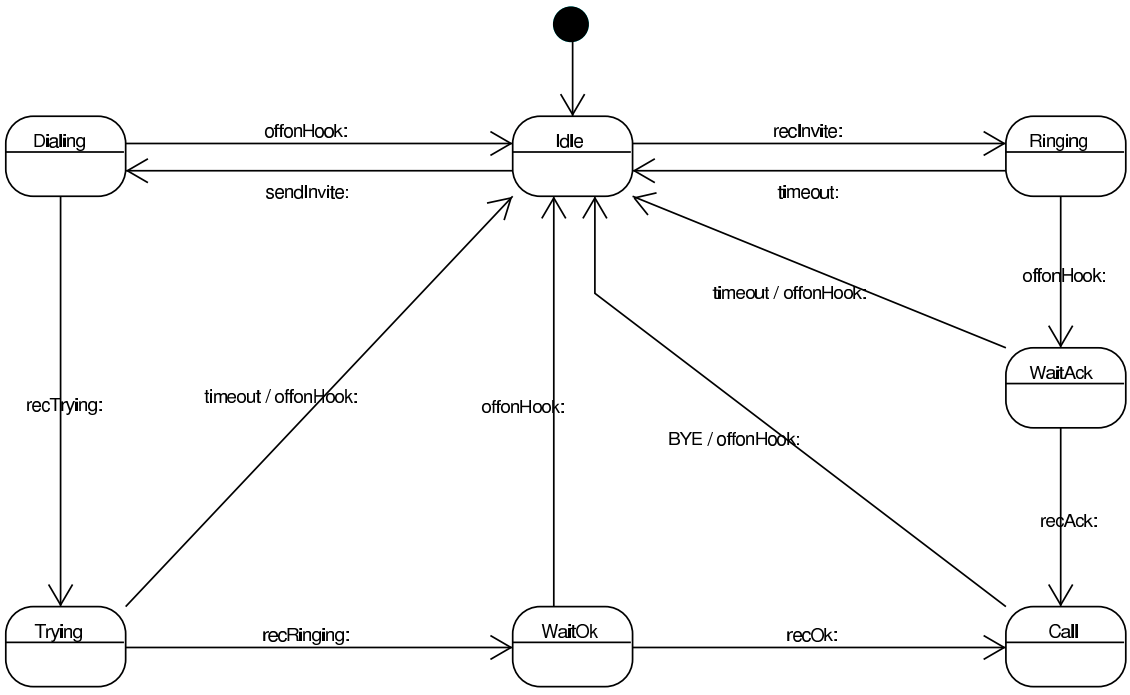


Abbildung 25 SIP Zustandsmaschine

Für unseren SIP-Clients mussten wir mehrere Zuständen definieren die für einen kompletten Verbindungsaufbau benötigt werden.

```
State *Idle     = collectState(new DebugState(this,"Idle"));
State *Ringing  = collectState(new DebugState(this,"Ringing"));
State *Call     = collectState(new DebugState(this,"Call"));
State *Dialing  = collectState(new DebugState(this,"Dialing"));
State *Trying   = collectState(new DebugState(this,"Trying"));
State *WaitOk   = collectState(new DebugState(this,"WaitOk"));
State *WaitAck  = collectState(new DebugState(this,"WaitAck"));

Operation *OpSendInvite = collectOp(new OperationSendInvite(this));
Operation *OpRecInvite  = collectOp(new OperationRecInvite(this));
Operation *OpRecTrying  = collectOp(new OperationRecTrying());
Operation *OpRecRinging = collectOp(new OperationRecRinging());
Operation *OpOffHook    = collectOp(new OperationOffHook(this));
Operation *OpOnHook     = collectOp(new OperationOnHook());
Operation *OpTimeout    = collectOp(new GenericOperation());
Operation *OpRecBye     = collectOp(new OperationRecBye(this));
Operation *OpRecOk      = collectOp(new OperationRecOk(this));
Operation *OpRecAck     = collectOp(new GenericOperation());
```

Der Idle Zustand wird am Anfang gesetzt, so dass wir uns nach der Initialisierung in diesem befinden und auf die Benutzereingaben richtig reagieren können.

Jedem State muss noch ein EventHandler mitgeben werden, der genau angibt, welche Operation aufgerufen wird und welches der nächste Zustand ist.

```
Idle->addEventHandler (Event_SendInvite, *OpSendInvite, *Dialing);
Idle->addEventHandler (Event_RecInvite, *OpRecInvite, *Ringing);

Dialing->addEventHandler (Event_RecTrying, *OpRecTrying, *Trying);
Dialing->addEventHandler (Event_OffOnHook, *OpOnHook, *Idle);
Dialing->addEventHandler (Event_RecRinging, *OpRecRinging,*WaitOk);

Trying->addEventHandler (Event_RecRinging, *OpRecRinging, *WaitOk);
Trying->addEventHandler (Event_Timeout, *OpTimeout, *Idle);
Trying->addEventHandler (Event_OffOnHook, *OpOnHook, *Idle);

WaitOk->addEventHandler (Event_RecOk, *OpRecOk, *Call);
WaitOk->addEventHandler (Event_OffOnHook, *OpOnHook, *Idle);

WaitAck->addEventHandler (Event_OffOnHook, *OpOnHook, *Idle);
WaitAck->addEventHandler (Event_Timeout, *OpTimeout, *Idle);
WaitAck->addEventHandler (Event_RecAck, *OpRecAck, *Call);

Ringing->addEventHandler (Event_OffOnHook, *OpOffHook, *WaitAck);
Ringing->addEventHandler (Event_Timeout, *OpTimeout, *Idle);

Call->addEventHandler (Event_RecBye, *OpRecBye, *Idle);
Call->addEventHandler (Event_OffOnHook, *OpOnHook, *Idle);
```

7.2.2.3 Zustand Idle

Dieser Zustand wartet auf ein **INVITE** oder kann selber ein solches auslösen. Hier entscheidet sich aufgrund des ersten Events, ob der Client die Zustandsmaschine als Empfänger oder Sender durchläuft, bis er sich schlussendlich wieder im Zustand „Call“ ist.

SendInvite

Wird ein **SendInvite** ausgelöst, so wird die Operation SendInvite aufgerufen, welche wiederum eine Funktion des SipWrappers startet, der die zu sendende Nachricht für die Anfrage zusammenstellt. Um ein INVITE auch senden zu können, braucht es die Zieladresse und Port, welche im User Interface eingegeben wird. Anschliessend werden die lokalen Daten, wie der SIP Port und Host, ermittelt und der Nachricht angefügt. Anhand dieser Daten kann der SIP Header zusammengestellt werden.

```
INVITE sip:gislean1@dskt6813.zhwin.ch:5060 SIP/2.0
To: <sip: gislean1@dskt6813.zhwin.ch>
From: <sip:gislean1@DSKT6813.zhwin.ch:2221>;tag=0909070d
Via: SIP/2.0/UDP ;branch=z9hG4bK-c87542-837977068-1--c87542-;rport
Call-ID: 373af13120a2370e
CSeq: 1 INVITE
Contact: <sip:gislean1@DSKT6813.zhwin.ch:2222>
Max-Forwards: 70
Content-Length: 0
```

Nun werden die Daten ermittelt, welche für den SDP Inhalt benötigt werden. Hier ist vor allem die lokale IP-Adresse wichtig, damit der Empfänger eine Zieladresse erhält. Weiter wird ein zufälliger Sitzungsschlüssel generiert, der ebenfalls beim Erstellen des SDP eingebunden wird. Nun kann dieser minimal benötigte SDP-Content der Nachricht angefügt und dann an das Ziel verschickt werden. Die anderen benötigten Daten werden bereits beim Initialisieren des SDP erstellt. Die Bedeutung der einzelnen Komponenten ist im Kapitel Session Description Protocol (SDP) erklärt.

```
To: <sip:loretman@DSKT6814.zhwin.ch:5060>
From: <sip:gislean1@DSKT6813.zhwin.ch:2221>;tag=0909070d
Via: SIP/2.0/UDP ;branch=z9hG4bK-c87542-837977068-1--c87542-;rport
Call-ID: 373af13120a2370e
CSeq: 1 INVITE
Contact: <sip:gislean1@DSKT6813.zhwin.ch:2222>
Max-Forwards: 70
Content-Type: application/sdp
User-Agent: SIPSEC
Content-Length: 238

v=0
o=- 3906564628 3906564628 IN IP4 160.85.170.136
s=DA SIP Security 2003
c=IN IP4 160.85.170.136
t=0 0
k=clear:40c4068eaecae6f6444547130962d1faf56a1e4b02b0b02a788bea9aa12b
m=audio 6666 RTP/AVP 0
a=ptime:20
a=rtpmap:0 PCMU/8000
```

Wenn nun das **INVITE** an den Empfänger abgeschickt wurde, so wird zum Zustand Dialing gewechselt und auf eine Antwort gewartet.

RecInvite

Wenn man sich im Zustand Idle befindet und eine Nachricht eintrifft, so wird dies vom SIPWrapper erkannt und dieser entscheidet aufgrund der Kennung (**ACK, BYE, CANCEL** oder **INVITE**) wie er weiterfahren soll. Der SIP Stack sendet nach Erhalt eines **INVITE** ein Trying an den Sender, so dass wir nur noch ein **180 Ringing** zurückschicken müssen, damit dieser seinen Zustand weiterführen kann.

Aus der Nachricht werden die wichtigsten Daten extrahiert. Hierzu gehören der Sitzungsschlüssel, der Remote Port und der Remote Host. In der Operation RecInvite wird der SRTP Kanal bereits jetzt initialisiert, dies beinhaltet sowohl den Empfänger als auch den Sender des SRTP. Man spricht hier von early media.

Als nächstes wird zum Zustand Ringing gewechselt.

7.2.2.4 Zustand Dialing

In diesem Zustand befindet sich der Sender nach dem Auslösen der **INVITE** Nachricht. Wenn nun eine Antwort zurückkommt, wird diese analysiert und verarbeitet. Wenn die erhaltene Antwort ein Trying enthält, so wird der Zustand ins Trying übergeführt. Bei einem erhaltenen Ringing wird zum Zustand WaitOk gewechselt, vorher liest der Empfänger aber noch die erforderlichen Daten aus der Antwort heraus. Das Ringing muss hier zusätzlich abgefangen werden, da es möglich ist, dass die Trying-Antwort nicht ankommt oder verloren geht. Das letzte zu behandelnde Ereignis ist, wenn sich die Gegenstelle lange nicht meldet und somit keine Antwort beim Sender eintrifft. Es sollte nun ein Timeout ausgelöst

werden.

7.2.2.5 Zustand Trying

Für den weiteren Verbindungsaufbau wird nur das Ringing behandelt. Kommt eine solche Nachricht an, so wird der Zustand gewechselt und man findet sich im Zustand WaitOk wieder. Vorher wird in der Operation RecRinging noch der SRTP-Kanal initialisiert. Sollte die Gegenstelle ein OffOnHook oder Timeout schicken, so wird die ganze Zustandsmaschine in den Zustand Idle zurückgesetzt.

7.2.2.6 Zustand Ringing

In diesem Zustand wird darauf gewartet, dass ein offHook Ereignis ausgelöst wird. Ist dies der Fall, wird die Funktion **`acceptCall()`** des SIP Wrappers aufgerufen, der eine Nachricht **`200 Ok`** an den Sender geschickt. Ist dies abgeschlossen wird zum Zustand WaitAck umgeschaltet. Auch hier ist es möglich, dass ein Timeout stattfindet und man dann in den Anfangszustand zurückkehrt.

7.2.2.7 Zustand WaitOk

Wenn beim Sender eine Bestätigung mit der Kennung 200 eintrifft, so muss nur noch eine Acknowledge zurückgesendet werden. Der Sender wechselt nach dem Erhalt des 200 OK in den Call Zustand. Auch hier muss es möglich sein, den Verbindungsaufbau abzubrechen, wobei ebenfalls ein onHook Ereignis ausreichen muss.

7.2.2.8 Zustand WaitAck

Der Empfänger hat seine Bereitschaft abgeschickt und wartet nun auf das Acknowledge des Senders. Bei Erhalt dieser Nachricht kann auch der Empfänger in den Zustand Call wechseln, womit sich nun beide in diesem Zustand befinden sollten. Der Empfänger kann den Verbindungsaufbau selbst abbrechen, indem er ein onHook Ereignis auslöst. Es ist aber auch möglich, dass dies der Sender gemacht hat. Dann muss die Verbindung über ein Timeout geschlossen und alles in den Anfangszustand versetzt werden.

7.2.2.9 Zustand Call

Wenn sich die Clients hier befinden, so hat der komplette Verbindungsaufbau bereits stattgefunden und es besteht ein Kanal, über welchen Daten ausgetauscht werden können. Diese Verbindung kann mit einem **`BYE`** geschlossen werden.

7.2.3 SIP-Stack Interface

7.2.3.1 Übersicht

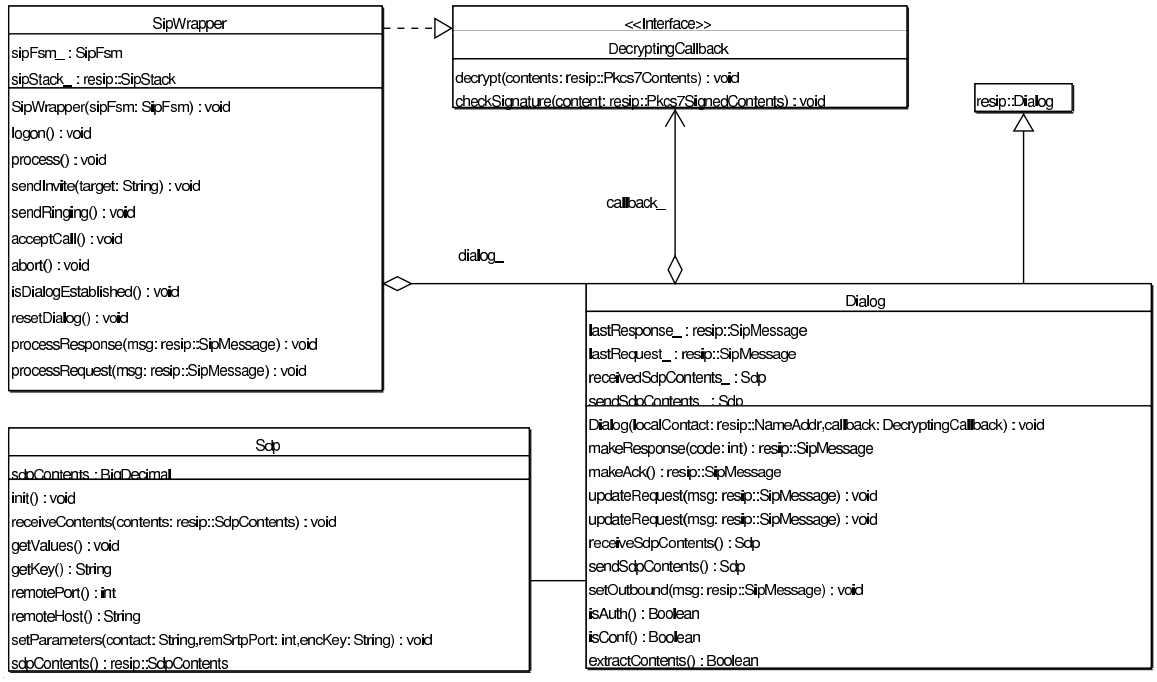


Abbildung 26 Klassenübersicht des SIP-Stack Interface

Wir haben eine strenge Trennung zwischen der Zustands-Maschine und dem SIP-Stack vorgenommen. Dadurch sind nur die Klassen **`SipWrapper`**, **`Dialog`** und **`Sdp`** vom SIP-Stack abhängig. Sollte der Wunsch aufkommen, den Stack auszutauschen, so sind nur diese drei Klassen betroffen.

Die Klasse SipWrapper stellt das Interface für die Zustands-Maschine zur Verfügung. Die Zustands-Maschine braucht nur noch die entsprechenden Vorgänge einzuleiten, um die internen Vorgänge kümmert sich der **`SipWrapper`**.

Das Verfolgen eines Anrufes übernimmt die von **`resip::Dialog`** abgeleitete Klasse **`Dialog`**. Sie speichert die für einen Anruf notwendigen Informationen und kann daraus neue Meldungen generieren.

Für die Behandlung der SDP-Kommunikation ist die Klasse **`Sdp`** zuständig. Sie kann neue SDP-Meldungen generieren und aus eintreffenden Meldungen die benötigten Daten auslesen.

7.2.3.2 Starten und Beenden des SIP-Thread

Der Startvorgang ist nicht kompliziert, jedoch etwas verzweigt, so dass er hier kurz erklärt wird um den Quelltext besser zu verstehen (Abbildung 27).

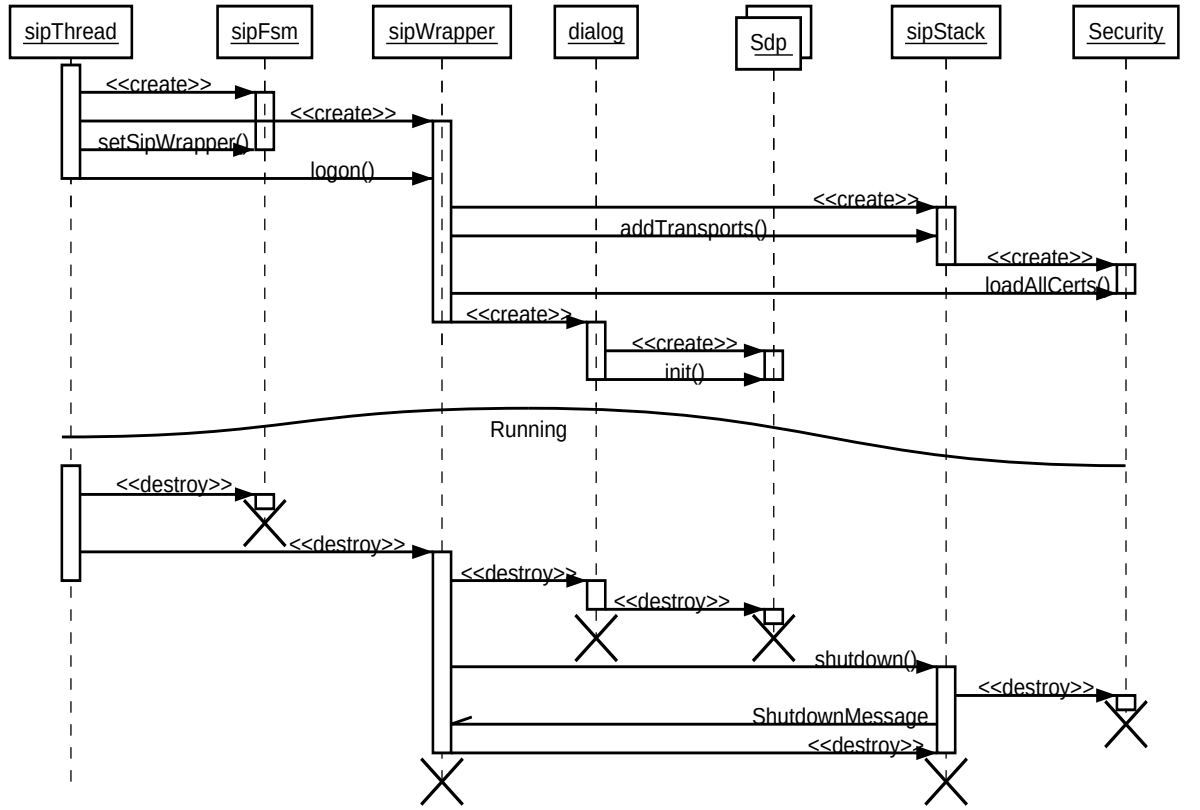


Abbildung 27 Starten und Beenden des SIP-Thread

Die Instanz der **SipFsmThread**-Klasse wird als eigener Thread gestartet. Die run-Methode (**operator()**) initialisiert zuerst die Zustandsmaschine. Wie diese aufgebaut ist, wurde in 7.2.2 erläutert. Daraufhin kann der **SipWrapper** instanziert werden, wobei er die Instanz der **SipFsm**-Klasse kennen muss. Da einige Operationen auf den **SipWrapper** zugreifen müssen, braucht der soeben erstelle **SipWrapper** der **SipFsm** über die Methode **SipFsm::setSipWrapper()** mitgeteilt werden. Nun kann der **SipWrapper** fertig initialisiert werden, dazu dient die Methode **SipWrapper::logon()**.

Der gestartete **SipWrapper** initialisiert nun den Rest. Insbesondere ist das die Klasse **SipStack** des reSIProcate Stacks. Neben dem Aufruf des Konstruktors müssen auch die benötigten **Transports** hinzugefügt werden, momentan UDP und TCP. Dem Konstruktor wird mitgeteilt, ob der Stack multithreaded oder nicht multithreaded laufen soll[i]. Weiter kann auch ein Objekt der **Security**-Klasse übergeben werden, was insbesondere dann notwendig ist, wenn von TLS Gebrauch gemacht wird. Wir überlassen das Konstruieren dem Stack.

Nun werden mit der Methode **Security::loadAllCerts()** die Zertifikate für S/MIME[ii] geladen.

Das Beenden verläuft ebenso unspektakulär, einzig muss darauf geachtet werden, dass vor dem Aufruf des **SipStack**-Destructors das baldige Ende dem Stack mit **SipStack::shutdown()** mitgeteilt wird. Sobald dieser die Aufräum-Arbeiten erledigt hat, schickt er eine **ShutdownMessage**, worauf der Destruktor aufgerufen werden kann.

7.2.3.3 Senden einer INVITE-Meldung

Vom Benutzer kommt eine Aufforderung, dass ein bestimmter User unter seiner SIP-URI angerufen werden soll. Die zu diesem Event gehörende Operation **OperationSendInvite** ruft danach die Methode **SipWrapper::sendInvite()** mit der gewünschten URI als Parameter auf (Abbildung 28). Diese Methode verwendet die Hilfsfunktion **Dialog::makeInitialInvite()** um eine **SipMessage** mit

i Die Version mit mehreren Threads ist momentan noch im Test und hat bei uns regelmässig zu Fehlern geführt.
ii Und natürlich auch TLS, falls das eingesetzt werden sollte

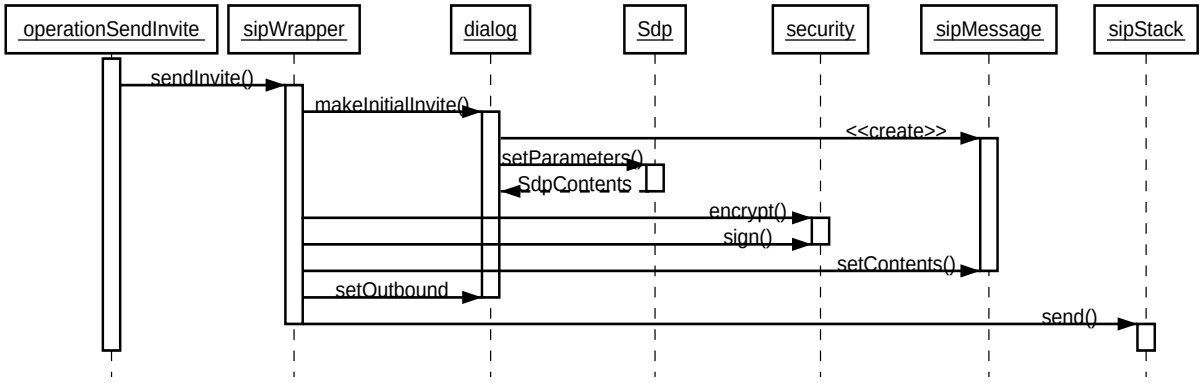


Abbildung 28 Versenden einer INVITE-Message

dazugehörigem **SdpContents** zu erzeugen. Die **SdpContents** werden durch Zuhilfenahme der Klasse **Security** zuerst verschlüsselt und dann signiert. Zum Schluss werden die so erhaltenen **Pkcs7SignedContents** der **SipMessage** hinzugefügt. Die **SipMessage** wird noch um die letzten Parameter mit **Dialog::setOutbound()** ergänzt (UA-Kennung und Transport-Parameter) und wird dann über **SipStack::send()** versendet.

7.2.3.4 Empfangen einer INVITE-Meldung

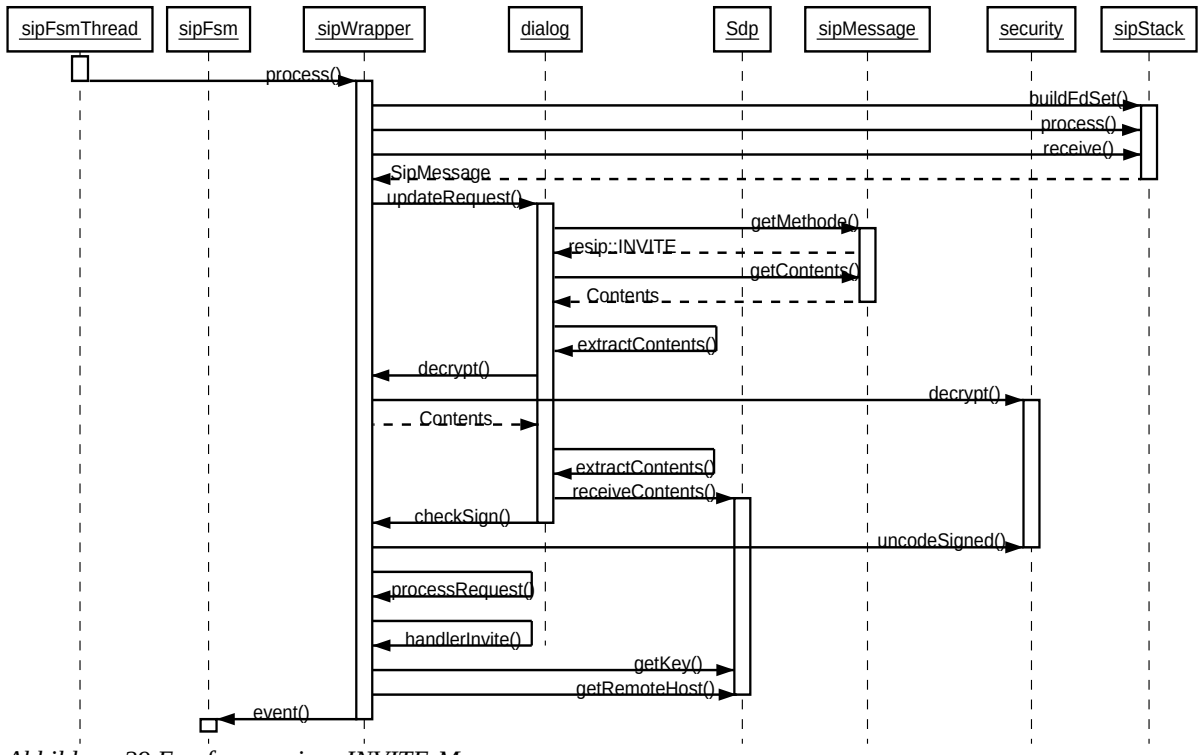


Abbildung 29 Empfangen einer INVITE-Message

Der Empfang einer Meldung ist viel komplexer als das Senden. Wie in Abbildung 29 ersichtlich, wird vom **SipFsmThread** die Methode **SipWrapper::process()** regelmässig aufgerufen (polling). Diese Methode lässt sich zuerst ein File-Descriptoren-Set von **SipStack::buildFdSet()** zusammenbauen. Dieses ist notwendig, um danach mit dem **FdSet** als Argument die Methode **SipStack::process()** aufzurufen. In dieser Methode findet im Stack die Verarbeitung von eingetroffenen Meldungen statt. Nun kann mit einem Aufruf der Methode **SipStack::receive()** geprüft werden, ob eine Meldung empfangen wurde.

Im Falle das eine Meldung empfangen wurde, liefert die Methode **process()** einen Zeiger auf die entsprechende **SipMessage** zurück. Falls keine Meldung empfangen wurde, wird **NULL** zurückgegeben.

Anhand der Methode **SipMessage::isRequest()** wird nun festgestellt, dass es sich um einen Request handelt. Damit der **Dialog** intern seinen Zustand anpassen kann, wird die erhaltene **SipMessage** dem Dialog mit **Dialog::updateRequest()** übergeben. Dieser ermittelt die Art der Meldung mit der Hilfe von **SipMessage::getMethode()** und stellt fest, dass es sich um einen **INVITE** handelt. Da ein **INVITE** SDP enthalten kann, holt er sich den Anhang der Meldung über **SipMessage::getContents()**. Der Inhalt wird extrahiert, wie das genau funktioniert ist im nächsten Abschnitt beschrieben.

Wenn das extrahieren des Anhangs funktioniert hat und die SDP-Meldung entschlüsselt werden konnte, ist **Dialog** mit allen benötigten Informationen ausgestattet und die Funktion kehrt zurück.

Nun wird die Methode **SipWrapper::processRequest()** mit der SipMessage als Parameter aufgerufen. Auch diese Methode prüft, was für eine Art von Meldung empfangen wurde und stellt ebenfalls fest, dass es sich um eine **INVITE**-Methode handelt. Nun wird ein **EventRecInvite** instanziert. Für diese Methode wurde ein **SipWrapper::handlerINVITE()** definiert, der dafür zuständig ist, den **EventRecInvite** mit den benötigten Informationen wie Host, Port und Passwort abzufüllen.

Das so erstellte Event wird nun über **SipFsm->event()** an die Zustands-Maschine verschickt, welche aufgrund dieses Events die notwendigen Operationen, wie das Versenden einer **180 Ringing** Meldung, ausführt und in den nächsten Zustand Ringing wechselt.

7.2.3.5 Extrahieren von MIME-Daten

Das Extrahieren von MIME-Daten ist etwas speziell gelöst, so dass ein paar Worte darüber notwendig sind.

Eine SIP-Meldung kann keine, einen oder mehrere MIME-Anhänge enthalten. In einem verschlüsselten S/MIME-Anhang (PKCS7) ist wiederum ein MIME-Anhang enthalten.

Wir haben das dadurch gelöst, indem wir die Methode **Dialog::extractContents()** so aufgebaut haben, dass sie sich rekursiv selber aufrufen kann.

Ihr wird als Parameter der Anhang übergeben. Bei diesem versucht sie den Typ des Inhalts festzustellen. Wird SDP erkannt, lässt sie den Inhalt von **Sdp::receiveContents()** auslesen und kehrt zurück. Wird jedoch **MultipartMixed** oder **MultipartSigned** als Typ erkannt, so ruft sie sich für jeden Teil (Part) nochmals auf. Beim Typ **Pkcs7** ruft sie die Funktion **decrypt()** des **DecryptionCallback**-Interface auf um den Inhalt zu entschlüsseln und ruft sich dann nochmals selber mit dem entschlüsselten Inhalt auf. Wird der Typ **Pkcs7Signed** erkannt, wird die Methode **DecryptionCallback::checkSignature()** aufgerufen, um die Signature überprüfen zu lassen. Ob die Signatur gültig ist, interessiert sie nicht. Wird ein anderer oder unbekannter Typ des MIME-Anhang entdeckt, wird ein Fehler gemeldet, bricht jedoch nicht ab.

7.2.3.6 Weitere Vorgänge zum Versenden und Empfangen von Meldungen

Die vorhergehenden Beispiele zum Versenden und Empfangen von Meldungen lassen sich gut auf die weiteren Ereignisse und Vorgänge übertragen. Diese funktionieren in weiten Teilen analog.

7.2.4 SRTP-Wrapper

In unserer Diplomarbeit verwendeten wir den SRTP-Stack vom Projekt libsrtp. Dieser Stack ist der einzige, frei verfügbare Stack, der das SRTP-Protokoll implementiert und ist in der Programmiersprache C geschrieben. Wir haben uns aus mehreren Gründen entschieden, einen „Wrapper“ für diesen Stack zu erstellen. Zu den wichtigsten Gründen gehören:

- Durch einen objektorientierten Wrapper verbergen wir, dass der Stack in C geschrieben ist. Mit Hilfe des Wrappers kann auf die Funktionalität des Stacks über ein Objekt zugegriffen werden. Die nötigen Funktionen und Strukturen werden durch den Wrapper versteckt und die Nutzung des Stacks erleichtert.
- Die Implementation des Projektes libsrtp ist der einzig frei verfügbare SRTP-Stack. Es ist aber sicher

denkbar, dass weitere Implementationen erstellt werden, sobald der Draft [SRTP] zum RFC wird. Der Client nutzt die Funktionalität des aktuellen Stacks nur über den Wrapper und enthält sonst keinen spezifischen Code um auf den Stack zu zugreifen. Durch unseren Wrapper ist es einfacher, den aktuellen Stack gegen einen neuen auszutauschen. Die Änderungen müssen nur im Wrapper stattfinden und der Client muss nicht angepasst werden.

- Für die Nutzung des Stacks mussten noch viele Verwaltungsaufgaben selber geleistet werden. Als Beispiel musste der Socket von uns selber erstellt und gebunden werden, diese Aufgabe wird nicht automatisch vom Stack übernommen. Die Aufgaben für die Verwaltung des Stacks fallen bei uns in den Aufgabenbereich des Wrappers und wurden darum von uns in den Wrapper verschoben.

Natürlich wurde beim Design des Wrappers darauf geachtet die Möglichkeiten der objektorientierten Programmierung zu nutzen. Daher wurde das Merkmal der Vererbung genutzt. Die Klasse SRTPWrapper ist die Oberklasse von SRTPReceiver und SRTPSender und vereinigt alle gemeinsamen Attribute und Operationen. Die Unterklasse SRTPSender und SRTPReceiver sind von SRTPWrapper abgeleitet und implementieren die spezifische Logik für den Sender und den Receiver.

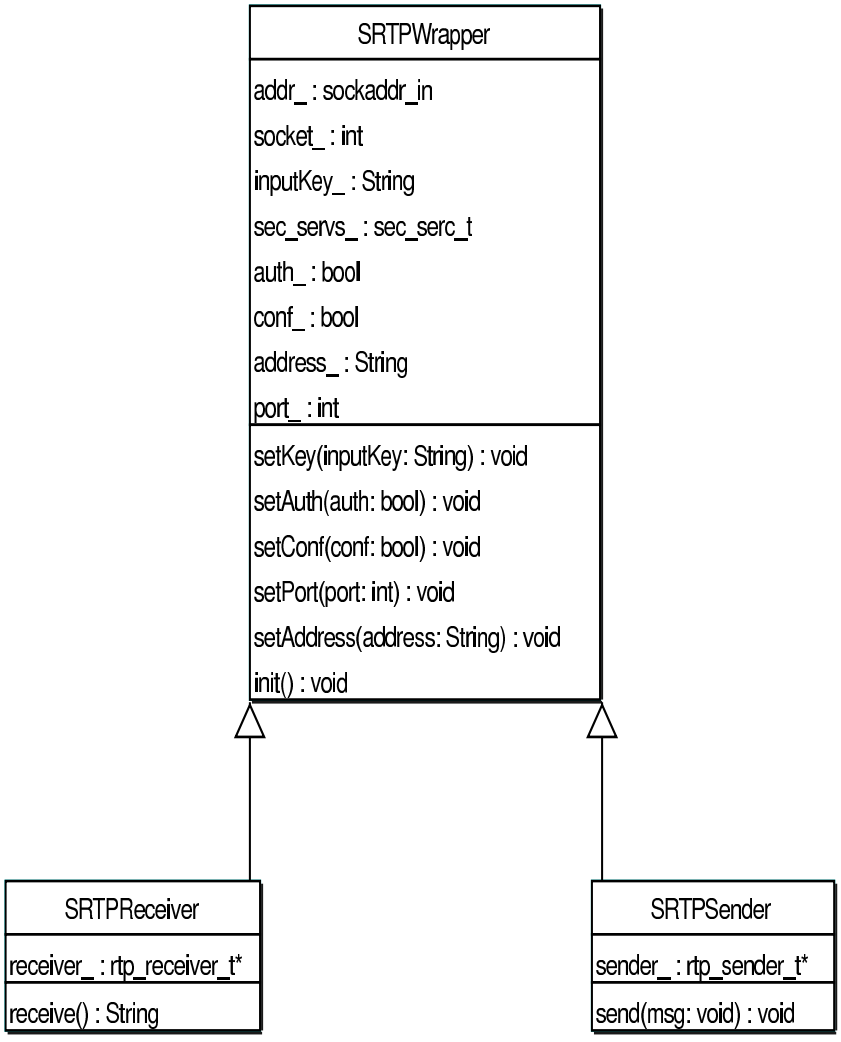


Abbildung 30 Klassendiagram SRTP-Wrapper

7.2.5 Mini-Programm zur Demonstration von S/MIME

Während der Entwicklung haben wir zweimal Fehler im Stack aufgedeckt. Um dem Entwickler-Team die Fehlersuche zu erleichtern, wurde ein minimales Programm geschrieben, welches die Fehler reproduziert. Gleichzeitig ist dieses Programm auch eine gute Demonstration, welche Schritte notwendig sind, um eine SIP-Meldung mit S/MIME verschlüsseltem SDP-Anhang zu erzeugen, zu versenden und wieder zu empfangen. Das Programm ist in 12.7 Mini-Programm abgedruckt.

7.2.6 Diverses

7.2.6.1 Strings

Die String-Template-Klasse aus der STL ist nicht Thread-Safe. Sie verwendet Copy-on-Write, d.h. der

interne Buffer es wird erst dann kopiert, wenn schreibend darauf zugegriffen wird. Das führt zu Problemen beim Kopieren über Thread-Grenzen hinweg.

Um diesem Problem aus dem Weg zu gehen, wird eine selbst geschriebene String-Klasse verwendet. Wir haben nur die Funktionen, welche notwendig sind, geschrieben. Diese stimmen mit dem Interface, wie auch mit dem Verhalten der **`std::string`**-Klasse aus der STL überein. Der interne Buffer wird bei einem Kopiervorgang sofort kopiert. Die Einbussen an Performance können sind kein Problem, da nicht mit grossen Zeichenketten gearbeitet wird.

7.2.6.2 Smart Pointers

Teilweise wird es schwierig festzulegen wer, wann für die Zerstörung der Objekte im Freispeicher (Heap) zuständig ist. Entweder wird vergessen, ein Objekt zu zerstören, was zu Speicher-Leaks führt oder ein Objekt wird zweimal zerstört, was durch ein Segementation Fault quittiert wird.

Eine Lösung bieten Smart-Pointers: Smart Pointers zählen die Anzahl an Referenzen auf ein Objekt und zerstören dieses, sobald keine Referenz mehr darauf zeigt.

Smart Pointers haben auch Nachteile, die sie nicht universell einsetzbar machen. Verweisen zwei Objekte (oder mehrere) gegenseitig aufeinander, so blockiert dieser Zyklus den Automatismus. In diesem Fall hilft nur das manuelle Aufbrechen des Zyklus oder besser: Vermeiden eines Zyklus von Anfang an.

In C++ lassen sich solche Smart-Pointers einfach durch Überladen von Operatoren erreichen:

- Basis ist ein Template, das mit dem gewünschten Typ des Zeigers instanziert wird
- Ein statischer Container speichert die Zeiger und die Anzahl Referenzen pro Zeiger
- Jeder Konstruktor erhöht die Anzahl des beim Konstruktor übergebenen Pointers um eins, auch der Copy-Constructor
- Der Destructor dekrementiert die Referenz-Anzahl um eins
- Der Zuweisungs-Operator dekrementiert die Referenz-Anzahl des alten Inhalts und erhöht die des neuen.

7.2.7 Random-Number-Generator

Während der Diplomarbeit musste von mehreren Orten auf ein Random-Number-Generator zurückgegriffen werden. So ist man beim Aufbau einer gesicherten SRTP-Verbindung auf einen zufälligen, symmetrischen Schlüssel angewiesen[i]. Aus diesem Grund programmierten wir in der Klasse **`Random`** unseren eigenen Generator, welcher die nötigen Werte als Hex-String oder auch unsigned long erzeugt. Innerhalb diese Klasse greifen wir auf die Funktionalität von OpenSSL 7.1.5 zurück und verwenden die Funktion **`RAND_bytes`**[ii].

OpenSSL greift für diesen Zweck standardmässig auf **`/dev/urandom`** zu. Seit Version OpenSSL 0.9.7 wird auf **`/dev/random`** zugegriffen, falls **`/dev/urandom`** nicht verfügbar ist.

7.3 Zielumgebung

7.3.1 Plattform

7.3.1.1 Betriebsystem

Als Zielplattform wurde GNU/Linux ausgewählt. Alle eingesetzen Bibliotheken sind für Unix konzipiert und als Open Source verfügbar. Zudem sind sie auch zwischen verschiedenen Systemen portierbar. Mit der Wahl von GNU/Linux wird der Weg für zukünftige Portierungen nicht verbaut.

i Auch für die Generierung der SDP SessionID wird ein zufälliger Wert benötigt.
ii Ein Buffer mit der gewünschten Menge an Zufallswerten wird gefüllt.

Mit geringem Portierungsaufwand sollte der Client auch auf anderen Unix-Systemen zum Laufen zu bringen sein. Mit etwas grösserem Aufwand auch unter Windows.

Zum Beispiel hatten wir schon Erfolg unseren Client unter MacOS X Jaguar zum Laufen zu bringen.

7.3.1.2 Entwicklungsumgebung

Ein weiterer Grund für die Wahl des Betriebsystem ist die grosse Verfügbarkeit von komfortablen Entwicklungs-Werkzeugen. Unser Client ist nicht an eine bestimme Entwicklungs-Umgebung gebunden, es kann eine beliebige C++/Make Umgebung verwendet werden. Wir haben Kdevelopp, Anjuta und Vim+Bash verwendet.

7.3.2 Entwicklungs-Tools

7.3.2.1 Autoconf und Automake

Sobald das Projekt grösser wird, muss man sich Gedanken über den Build-Prozess machen. Das Editieren von einzelnen Makefiles wird langsam mühsam, insbesondere, wenn verschiedene Abhängigkeiten erfüllt werden müssen. Zudem soll das Build-System auch in anderen Umgebungen funktionieren (und seien es nur unterschiedliche Pfade).

Anfangs wurde das von Kdevelop erzeugte Automake- und Autoconf-System verwendet, welches aber schnell zu umständlich wurde, da es primär auf KDE-Anwendungen zugeschnitten ist.

Mit dem Kauf von [GNUAUTO] waren wir dann in der Lage, selber ein solches System aufzubauen. Hat man einmal die grosse Anfangshürde genommen, will man das System nicht missen. Grundsätzlich muss nur ein Entwickler des Teams etwas vom Build-System verstehen, ein wenig Verständnis schadet aber auch den anderen Entwicklern nicht. Im Folgenden seien die Zusammenhänge der einzelnen Files erklärt.

Grundlagen

Anstatt sich um immer wiederkehrende Aufgaben zu kümmern, nimmt einem das Automake/Autoconf-System die Arbeit ab und beschränkt sich auf grundlegende Angaben der Quell- und Ziel-Dateien und der verwendeten Bibliotheken.

Der Entwickler muss sich prinzipiell nur um folgende Dateien kümmern:

`configure.in`
: Das ist die zentrale Steuerdatei, auf deren Angaben alle Tools zugreifen. Hier werden grundsätzliche Angaben zum Projekt gemacht und der Pfad zu den einzelnen **`Makefile.am`** eingetragen. Zudem werden hier sämtliche Tests auf Plattform-spezifische Details eingetragen.

`Makefile.am`
: Für jedes Modul der Anwendung wir ein **`Makefile.am`** erstellt. Dieses Makefile listet die Quellen auf und beschreibt zu welchem Ziel sie gehören. Aus diesen Angaben wird später automatisch das Ziel erzeugt, wobei die Abhängigkeiten automatisch aufgelöst werden. Für spezielle Zwecke können auch normale **`make`**-Ziele eingefügt werden.

`acinclude.m4`
: Diese Datei ist nicht unbedingt notwendig, sollte aber verwendet werden um eigene M4-Makros zu speichern. In der **`configure.in`** sollen nur Makros aufgerufen und nicht definiert werden.

Aus diesen Dateien wird später das gesammte Set an Dateien generiert, das dem Anwender von nun an den bekannten Dreisatz: **`./configure && make && make install`** zur Verfügung stellt.

Doch dazwischen laufen verschiedene Vorgänge ab.

Erstellen des Buildsystem

Zuerst müssen die **Makefile.in** und diverse weitere Dateien, inklusive dem **configure**-Script selber erzeugt werden, welche **configure** später in **Makefile**s transformieren wird. **configure** ist ein sehr portables Bash-Script, das in jeder Bash-Implementierung läuft. Der Anwender ist daher nicht auf die aktuellen Autotools[i] und GNU M4 angewiesen, um das Programmpaket zu übersetzen.

In der Abbildung 31 ist ersichtlich, welche Dateien aus welcher Quelle erstellt werden. Um diesen Vorgang zu automatisieren, existiert ein Shell-Script **bootstrap**. Jetzt reicht es, dieses Script einfach aufzurufen.

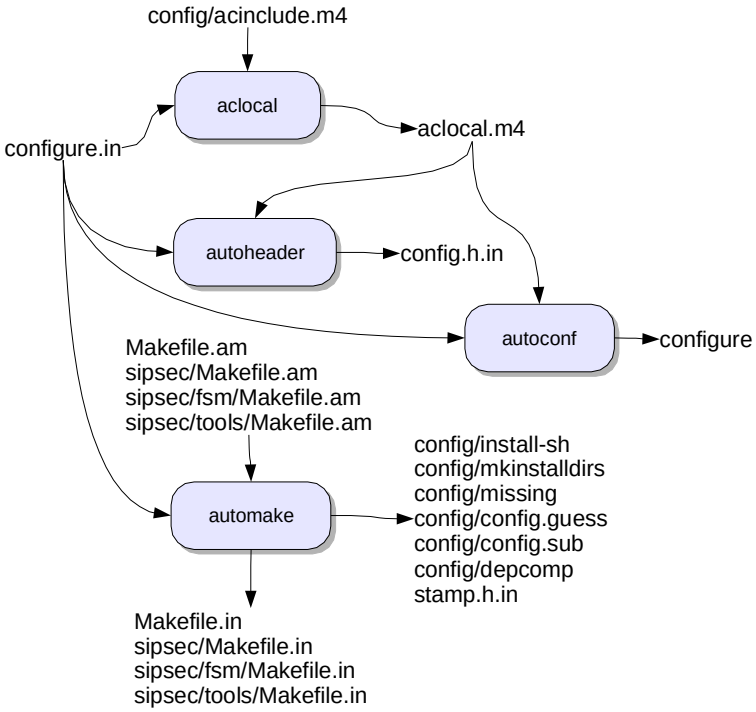


Abbildung 31 Erstellen der Build-Umgebung

Der configure-Vorgang

Jetzt wurden alle Dateien für das Build-System erstellt, welche schlussendlich in der Distribution der Software landen. So kann der Anwender die Software installieren. Auch der Entwickler muss jetzt so vorgehen, wie der Anwender um die Software zu kompilieren.

```
./configure --with-srtp-lib=/path/to/libsrtp.a \
            --with-srtp-inc=/path/to/srtp/includes
```

erstellt neben einigen Hilfsdateien die **Makefile**s. Es ist sehr gut möglich, dass **configure** in der oben gezeigten Fassung mit einer Fehlermeldung abbricht. Mit grosser Wahrscheinlichkeit hat er dann eine der benötigten Bibliotheken nicht gefunden. Abhilfe gibt die explizite Angabe des Pfades zur gewünschten Bibliothek. Die genauen Optionen erläutert ein Aufruf von:

```
configure -help
```

Abbildung 32 zeigt den Zusammenhang der einzelnen Quell-Dateien und wie diese zu den gewünschen Ausgangsdateien transformiert werden.

Die **config.cache**-Datei wird mit dem ersten Durchlauf von **./configure** angelegt und dient dazu, spätere **./configure**-Läufe zu beschleunigen, indem bereits gefundene Einstellungen im Cache gespeichert werden.

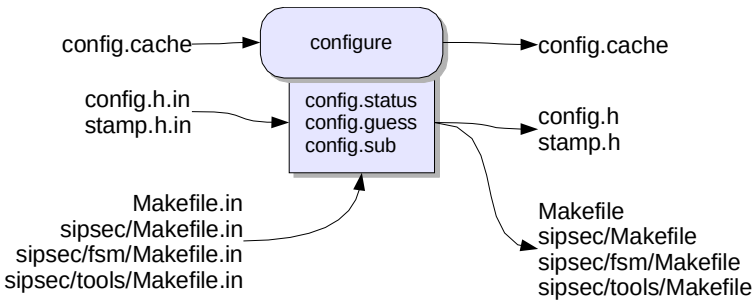


Abbildung 32 Der ./configure-Vorgang

i **automake**, **autoconf** und verschiedene weitere Tools wie **autoheader**, **aclocal** und **autoscan**

Der make-Vorgang

Der make-Vorgang unterscheidet sich nicht mehr davon, als wenn die Makefiles von Hand angelegt worden sind. Make erstellt die gewünschten Ziele und compiliert die Quelldateien mit dem durch **`configure`** erkannten Compiler. Dieser Vorgang ist in Abbildung 33 illustriert.

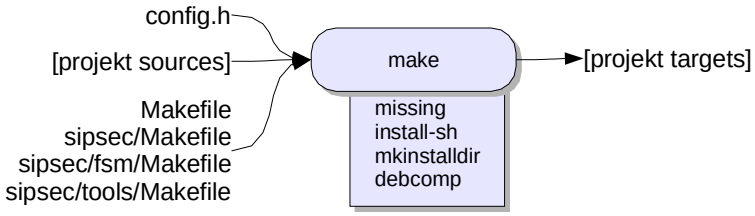


Abbildung 33 Der Make-Vorgang

`make install` ist zwar möglich, aber in der jetzigen Phase der Entwicklung noch nicht zu empfehlen.

7.3.2.2 TinyCA

Informationen

Mit dem Programm TinyCA steht ein einfaches, grafisches Frontend für die Verwaltung einer Certification Authority (CA) zur Verfügung. TinyCA ist in Perl/Gtk geschrieben und arbeitet als ein Frontend für Openssl, welches bereits schon in Kapitel 7.1.5 OpenSSL beschrieben ist.

Funktionen:

Das Programm TinyCA unterstützt in der Version 0.5.2 (beta) folgende Funktionen:

- Beliebig viele CAs
- Erstellung und Wiederruf von X.509 und S/MIME Zertifikaten
- PKCS#10 Requests können importiert und signiert werden
- Server-Zertifikate
    - Zertifikate können in folgende Formate exportiert werden: PEM, DER, TXT, ZIP und PCKS#12
    - Die Zertifikate können z.B. für folgende Server eingesetzt werden: Apache, Postfix, OpenLDAP, Cyrus und FreeS/WAN
- Benutzer-Zertifikate
    - Zertifikate können in folgende Formate exportiert werden: PEM, DER, TXT, ZIP und PKCS#12
    - Die Zertifikate könne z.B. für folgenden Programme genutzt werden: Netscape, Internet Explorer, Outlook (Express), Konqueror, Opera, kmail, mutt und FreeS/WAN
- Widerrufslisten
    - CRLs können in folgende Formate exportiert werden: PEM, DER und TXT
- Unterstützte Sprachen
    - Englisch
    - Deutsch

System-Voraussetzung:

TinyCA hat folgende Abhängigkeiten:

- OpenSSL Version
- Gtk-Perl

Bezug:

TinyCA kann unter folgendem Link bezogen werden:

http://tinyca.sm-zone.net

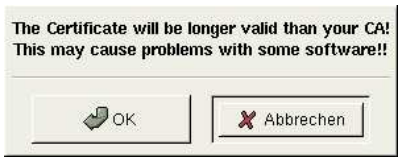
Auf der Internetseite sind verschiedene Varianten aufgelistet. Es kann als Package für Debian, RedHat oder SuSE direkt bezogen werden. Weiter kann man TinyCA auch als Tarball, als Source-Code oder via CVS herunterladen.

Anwendung

Um einen sicheren Verbindungsaufbau mit S/MIME zu ermöglichen wird ein Verfahren mit öffentlichen Schlüsseln verwendet. Um vollständiges Vertrauen in einen öffentlichen Schlüssel gewinnen zu können, muss die Verbindung vom öffentlichen Schlüssel zu seinem Benutzer sichergestellt werden. Diese Verbindung wird mit Hilfe eines Zertifikates hergestellt. Das Zertifikat bindet einen Namen eines Rechners oder einer Person zu einem dazugehörigen, öffentlichen Schlüssel[i].

TinyCA wurde benutzt, um unsere CA zu realisieren und zu verwalten. Um unsere sichere Verbindung zu testen, müssen wir im Besitz des Root-Zertifikates und der Zertifikate für die einzelnen Clients sein. Mit TinyCA ist es nun einfach eine CA zu erstellen und die nötigen Client-Zertifikate zu erzeugen. Die grafische Oberfläche von TinyCA ist sehr übersichtlich und intuitiv.

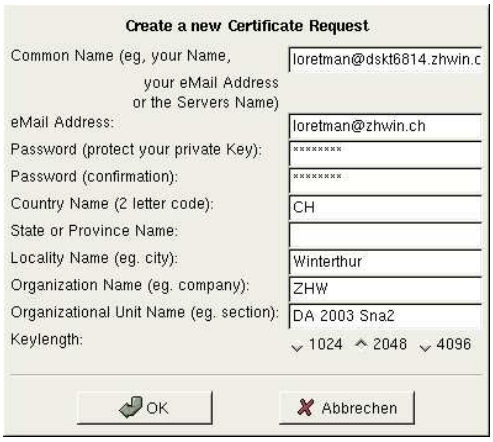
Das Einzige was an TinyCA etwas Kopfzerbrechen bescheren kann, ist das Feld um eine neue CA zu erstellen. Es ist zweimal das Feld **eMail Address** aufgelistet und es kann eigentlich nur vermutet werden, welche Bezeichnung dieses Feld richtigerweise besitzen müsste. Durch den im Feld enthalten Standard-Wert von 3650, kommt man schnell auf die eigentliche Bedeutung dieses Feldes. Es handelt sich bei diesem Feld um die Zeitspanne für die Gültigkeit des Root-Zertifikates. Für unsere Zwecke ist die Gültigkeit von 366 Tagen vollkommen ausreichend und wurde darum auf diesen Wert beschränkt. Hier ist darauf zu achten, dass das Root-Zertifikat eine längere Gültigkeitsdauer besitzen sollte, als die später ausgestellten Client-Zertifikate. Wird dies nicht beachtet, kann es bei der Nutzung der Zertifikate mit mancher Software zu Problemen kommen. TinyCA weist auf diesen Umstand hin.



Ein weiterer Punkt, der im Zusammenhang mit unserem Verwendungszweck beachtet werden soll, besteht bei der Wahl des **Common Name** bei der Anfrage nach einem neuen Client-Zertifikat. Der **Common Name** erscheint im Subjekt-Feld des Zertifikates und kann zum Beispiel der Name des Benutzers, die E-Mail-Adresse des Benutzers oder der Name des Servers sein. In unserem Fall eignet sich die SIP-Uri des Benutzers, welche aus dem Usernamen und dem Rechnernamen besteht.

Nachdem alle nötigen Client-Zertifikate erstellt worden sind, kann man diese und das Root-Zertifikat in ein gewünschtes Format exportieren. Wir wählten für unseren Zweck das Format PKCS#12. Im Objekt nach dem PKCS#12 Format sind das öffentliche Zertifikat und der private Schlüssel des Benutzers enthalten. Zusätzlich kann dieses Format optional auch das CA-Zertifikat beinhalten. Diese Option wird beim Exportieren ausgewählt.

i Dieser wird von einer dritten, vertrauenswürdigen Instanz unterschrieben.



Für unseren Client müssen wir nun den privaten Schlüssel des Benutzers, sowie das entsprechende, öffentliche Zertifikat und das CA-Zertifikat extrahieren. Leider bietet TinyCA diese Funktionalität nicht an. Aus diesem Grund muss man auf die Konsole zurückgreifen und direkt mit OpenSSL arbeiten.

Folgender Befehl lieferet den privaten Schlüssel des Benutzers:

```
openssl pkcs12 -in filename.p12 -out id_key.pem -nocerts
```

Dieser Befehl liefert das öffentlich Zertifikat:

```
openssl pkcs12 -in filename.p12 -out id.pem -nodes -nokeys
```

An das CA-Zertifikat kommt man schliesslich durch den Befehl:

```
openssl pkcs12 -in filename.p12 -out root.pem -nodes -cacerts -nokeys
```

Um diese Funktionen durchzuführen wird man von OpenSSL aufgefordert, ein Passwort für die Durchführung der Befehle einzugeben, es handelt sich dabei jeweils um das Passwort des Benutzers mit dem der private Schlüssel gesichert wird.

7.3.2.3 Concurrent Versions System (CVS)

Während unserer Diplomarbeit arbeiteten wir gemeinsam am gleichen Projekt. Um uns nicht immer genau absprechen zu müssen, wer wann, an welcher Datei arbeitet, griffen wir auf CVS zurück, dass unsere Arbeit stark vereinfachte. In der Folge sollen die Vorteile und die Funktionsweise von CVS kurz erläutert werden.

CVS ist eine Art Versionsverwaltungssystem, dass vor allem für zwei Aufgaben eingesetzt wird: Zusammenarbeit und Historienverwaltung. Die Historienverwaltung ist notwendig um den momentanten Zustand des Programmes mit einer alten Version zu vergleichen. Dies kann vor allem dann nützlich sein, wenn man plötzlich eine Version hat, die Fehler aufweist und zu einer kompilierbaren und lauffähigen Version zurückkehren will. Um diesen alten Zustand wieder herzustellen, ist mit CVS kein Problem. Es gibt Befehle wie: „Gib mir den Quelltext wie er vor 1 Woche war“ oder „als wir die letzte öffentliche Version freigegeben haben“. Diese Möglichkeiten der schnellen Wiederherstellung der Funktionalität macht CVS äusserst atraktiv.

Die zweite, sehr nützliche Funktion kommt zum Tragen, wenn mehrere Personen zusammen an einem Projekt arbeiten. CVS übernimmt die Aufgabe der gegenseitigen Koordination der Entwickler, indem es die Integration der Veränderungen übernimmt und allfällige Konflikte im Auge behält. So muss der Entwickler die Datei, an welcher er arbeiten will, nicht für andere sperren, sondern hat stets freien Zugriff auf alle Projektdateien. Dieser Prozess benutzt das Kopieren-Modifizieren-Zusammenfassen-Modell das wie folgt funktioniert:

- Entwickler A fordert eine Kopie des kompletten Verzeichnisbaumes (der alle Dateien des Projektes enthält) an. Dies wird auch „Check out“ einer Repository genannt.
- Nun kann der Entwickler frei an dieser Kopie arbeiten und die Dateien verändern. Gleichzeitig ist es

möglich, dass andere Entwickler weiterarbeiten. Da alle Kopien lokal sind und dadurch unabhängig sind, gibt es keine Konflikte.

- Entwickler A ist fertig und sendet seine Änderungen mit einer „Log-Nachricht" als Kommentar an den CVS-Server. Dies wird mit dem „**commit**" bewerkstelligt. Der Server lässt diese Änderungen in die Hauptkopie einfliessen, wo sie nun aufgezeichnet sind.
- Nun können andere Entwickler den Server abfragen (**update**) um herauszufinden ob die Hauptkopie seit ihrer letzten Kopie verändert wurde. Wurden Dateien verändert, so wird die lokale Kopie aktualisiert.
- Bei CVS sind alle Entwickler gleich. Es ist somit die Entscheidung der Entwickler, wann ein **commit** oder **update** vollzogen wird. Üblich ist vor grösseren Veränderungen eine Aktualisierung zu starten und die Veränderungen erst auf den Server zu laden, wenn diese vollständig und getestet sind. Dadurch ist sicher gestellt, dass die Hauptkopie immer in einem lauffähigen Zustand bleibt.
- Falls zwei Entwickler am gleichen Quelltext Veränderungen vornehmen und diese auf den Server laden, so entsteht ein Konflikt, welcher vom CVS bemerkt wird. Schickt Entwickler B nach A seine Änderungen an den Server, so bekommt dieser eine Rückmeldung, dass ein Konflikt stattgefunden hat. Zusätzlich werden im betroffenen File klar sichtbare Konfliktmarken gesetzt. Entwickler B muss sich nun das Ganze nochmals ansehen und eine neue Version abschicken, die den Konflikt löst. CVS informiert also über Konflikte, lösen müssen diese aber die Entwickler.

Wichtigste Befehle

Um ein Projekt von einem Server zu holen, müssen zuerst die Umgebungsvariablen richtig gesetzt werden. Dies macht man am besten in der bash mit den Befehlen:

Wenn CVS-Repository zum Beispiel auf **ksitb.zhwin.ch:/home/cvsrep** liegt, lauten die export-Befehle:

```
export CVS_RSH=ssh
export CVSROOT=:ext:user@ksitb.zhwin.ch:/home/cvsrep
```

cvs checkout <Projektname>: Eine Arbeitskope vom Archiv anfordern.

```
cvs checkout sipsec
```

cvs add <file>: Datei zum aktuellen Projekt hinzufügen, hier ist darauf zu achten, dass der komplette Pfad angegeben wird. Um die Datei wirklich auf den Server zu laden, muss zuerst ein **commit** durchgeführt werden. So ist sie erst im lokalen Verzeichnisbaum hinzugefügt. Wenn man eine binäre Datei hinzufügen will, so sind die Parameter **kb** mitzugeben

```
cvs add main.cpp
cvs add -kb <Pfad zur Datei>
```

cvs commit { -m „Kommentar" } { file }: Ohne zusätzliche Parameter werden alle Dateien die lokal aktueller sind, auf den Server geladen. Mittels „-m" kann direkt ein Kommentar mitgegeben werden, der in der Log-Nachricht gespeichert wird. Falls ein oder mehrere Files beim **commit** angegeben werden, so werden nur diese auf dem Server aktualisiert.

```
cvs commit -m „added the user interface to the main loop" main.cpp
```

cvs update: Aktualisieren von Veränderungen, welche durch andere Entwickler vorgenommen und auf den Server geladen wurden.

```
cvs update
```

Zeigt auch die lokalen Dateien an, welche noch auf den Server geladen werden müssen. Die Rückmeldung

der Veränderungen und Unterschiede wird mittels Buchstaben bewerkstelligt.

cvs server: Updating <Verzeichnis>: Dieses Verzeichnis wird gerade aktualisiert.

?: Datei ist lokal im Verzeichnisbaum vorhanden, aber nicht Bestandteil des CVS Archivs.

M: Datei ist lokal verändert worden und momentan aktueller als auf dem CVS Server. Kann mittels commit zum Server geladen werden.

U: Das File wurde zum ersten Mal zur lokalen Arbeitskopie hinzugefügt.

P: Datei wurde in der lokalen Arbeitskopie aktualisiert.

C: Konflikt vorhanden. Bevor diese Datei „commited" werden kann, muss der Konflikt durch den Entwickler gelöst werden.

Beispiel einer **update**-Ausgabe:

```
? sipsec/uml/exKlassendiagramm.gif
cvs server: Updating .
M configure.in
cvs server: Updating sipsec
cvs server: Updating sipsec/sip
U sipsec/sip/testsdp.cpp
cvs server: Updating sipsec/tools
P sipsec/tools/Makefile.am
cvs server: Updating sipsec/srtp
C sipsec/srtp/testsrtp.cpp
```

7.3.2.4 Doxygen

Doxygen ist ein Dokumentationssystem für C++, C, Java und IDL (Corba). Es kann eine online Dokumentation (HTML) und ein offline Referenzhandbuch (in Latex) aus den dokumentierten Quelldateien erstellen. Die Dokumentation wird direkt aus dem Sources extrahiert. Dadurch wird es stark vereinfacht, die Dokumentation konsistent mit dem Source Code zu halten. Die Ausgabe kann auch sehr hilfreich sein, um einen Überblick über die ganze Struktur zu erlangen. So können Relationen zwischen den verschiedenen Elementen angezeigt werden. Hierzu gehören Vererbungsdiagramme und auch Abhängigkeitsgraphen, welche automatisch generiert werden.

Um ein Handbuch für ein Projekt zu generieren, muss man der Quelltext mit speziellen Dokumentationsblöcken versehen werden. Diese haben ein genau definiertes Format, damit diese auch interpretiert und dargestellt werden. Für jedes Codeelement gibt es zwei Typen zur Beschreibung, ein **brief** und eine detaillierte Beschreibung. Wie die Namen schon aussagen, so ist **brief** eine Kurzbeschreibung, meistens ein Satz, während die detaillierte Beschreibung länger ist und viel stärker ins Detail geht. Um diese Blöcke zu kennzeichnen gibt es verschiedene Stile, wir verwendeten den JavaDoc Stil. Weitere Möglichkeiten, wie den Qt Stil, sind im Manual von Doxygen beschrieben, welches auf der Homepage (www.doxygen.org) gefunden werden kann.

```
/**
 * \brief (Kurzbeschreibung)
 * ...   detaillierte Beschreibung ...
*/
```

Es ist möglich, neben **brief** weitere Befehle zu platzieren. Die komplette Liste kann im Handbuch gefunden werden. In der Folge sollen die wichtigsten aber noch kurz erwähnt werden. Um wichtige Funktionsparameter zu beschreiben, wird der Befehl **param** verwendet. Jede Parameterbeschreibung startet auf einer neuen Linie. Falls die Funktion einen Rückgabewert liefert, so kann dieser mit **return** genauer beschrieben werden. Weitere nützliche Möglichkeiten sind das Einfügen der Autoren, Warnungen oder noch nicht behobene Fehler (Bugs).

```
/**
 * \author { Liste der Autoren }
 * \bug { Fehlerbeschreibung }
 * \warning { Nachricht zur Warnung }
 * \param < Parametername > { Parameterbeschreibung }
 * \return { Beschreibung des Rückgabewertes }
*/
```

Es folgt ein kleines Beispiel welches vom doxygen gelesen und verarbeitet werden kann.

```
/**
 * \brief Set parameters in the SDP Content.
 * Contact information are set in this function. The refered
 * parameters will be set to the SDP Content for transaction.
 * \param encKey Transmitted key for the media channel
 * \author Andreas Gisler
*/

void Sdp::setParameters(String contact, int remSrtpPort, String
encKey){}
```

Das ausführbare **doxygen** ist das Hauptprogramm, welches den Source Code analysiert und anschliessend die Dokumentation generiert. In der Konfigurationsdatei sind alle Einstellungen abgelegt, welche vom **doxygen** benötigt werden. Diese können direkt mit einem Texteditor oder mit dem **doxywizard** verändert werden. **doxywizard** ist ein Frontend, welches **doxygen** Konfigurationsdateien erstellen, lesen und schreiben kann.

Wird nun **doxygen** ausgeführt, so werden die Verzeichnisse html, rtf, latex und man erstellt. Anhand des Namens kann man schon erkennen, dass die Verzeichnisse die generierte Dokumentationen für das jeweilige Format enthält.

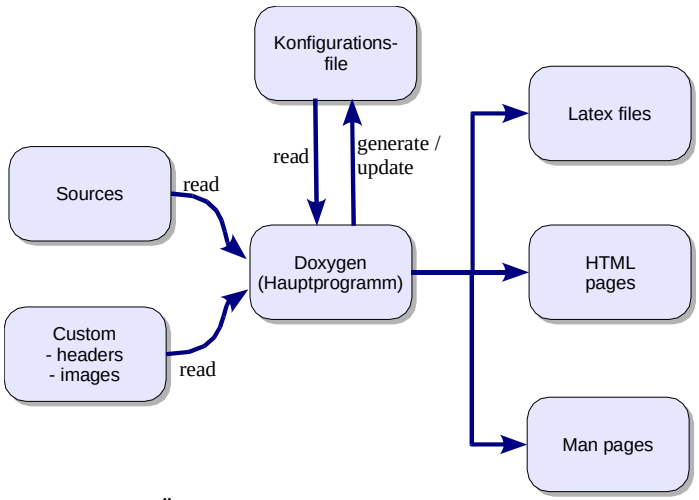


Abbildung 34 Übersicht über die Funktionsweise von Doxygen

8 Kurzanleitung für das Aufsetzen einer SIP Verbindung

Inhaltsverzeichnis

8.1 Installation

8.1.1 Installation von SIPSec

Es wurde ein Tarball erzeugt[i], der sich unter Unix wie üblich installieren lässt:

```
$ tar xzf sipsec-0.1.0.tar.gz
$ cd sipsec-0.1.0/
$ ./configure --with-srtp-inc=/path        \
              --with-srtp-lib=/path        \
              --with-resiprocate-inc=/path \
              --with-resiprocate-lib=/path \
              --with-safept-inc=/path      \
              --with-safept-lib=/path
$ make
# make install
```

Im Moment wird von **make install** abgeraten, da der Client noch in der frühen Entwicklungsphase steckt. Die Angaben der Bibliotheken sind hinfällig, sofern die entsprechenden Bibliotheken installiert sind und in einem der üblichen Pfade[ii] zu finden sind. In diesem Fall sollten die Bibliotheken automatisch gefunden werden.

SRTP kann im Moment nicht installiert werden, so dass hier die Angaben immer notwendig sind. Die **inc**-Endung der Parameter deutet übrigens auf ein Include-Verzeichnis hin, während die **lib**-Endung ein Hinweis auf eine Bibliothek ist.

Weitere Hinweise und Optionen zu liefert:

```
$ ./configure –help
```

8.1.2 Installation von reSIProcate

reSIProcate wird sporadisch als Tarball herausgegeben. Momentan funktioniert jedoch nur die neuste Version aus dem reSIProcate-CVS-Server. Diese kann, wie auf der Projektseite beschrieben, ausgecheckt werden. Natürlich wird ein Tarball mit einer funktionierenden Version auf unserer CD mitgeliefert.

Die Installation verläuft ebenfalls sehr einfach:

```
$ tar xzf resiprocate-cvs-2003-10-20.tar.gz
$ cd sip/
$ ./configure --with-ares=$(pwd)/contrib/ares --with-openssl
$ make
# make install
```

Damit ist der Stack installiert und einsatzfähig. Für Tests können Programme aus dem Test-Verzeichnis gestartet werden. Diese sind im Build-Verzeichnis unter **resiprocate/test/bin.debug.Linux.i686/** zu finden. Empfehlenswert sind z.B. **testSpeed** und **limpc**[iii].

i Kann nach erfolgreichem **./configure** jederzeit durch **make dist** erzeugt werden
ii Übliche Pfade sind **/usr/lib** und **/usr/local/lib**
iii Hierfür müssen z.B. die im Verzeichnis **certs** mitgelieferten Zertifikate nach **$HOME/.sip** kopiert werden

8.1.3 Installation von libsrtp

Eigentlich darf hier nicht von Installation geredet werden, da das Makefile kein **`install`**-Ziel kennt.

Die Installation ist auch hier einfach:

```
$ tar xzf srtp-1.0.6.tgz
$ cd srtp-1.0.6/
$ ./configure
$ make
```

8.1.4 Installation von SafePt

Der Umstand, dass hier ein unübliches Build-System verwendet wird, irritiert ein wenig. Ansonsten verläuft die Installation problemlos. Nicht zu vergessen ist das Setzten der Umgebungsvariable CXX:

```
$ tar xzf safept-0.1.0.tar.gz
$ cd safept-0.1.0/
$ export CXX=$(which g++)
$ ./configure.pl
$ make
# make install
```

8.2 Ungesicherte SIP Verbindung

Eine ungesicherte Verbindung ist auch mit unserem Client möglich. Es kann sogar gewählt werden, ob nur S/MIME oder SRTP abgeschaltet wird. Es können auch beide miteinander abgeschaltet werden. Besonders zum Zweck der Fehlersuche sind diese Hilfsmittel sehr nützlich.

Zum Initieren einer ungesicherten Verbindung wird der Client wie folgt gestartet[i]:

```
$ sipsec --nosmime --nosrtp
```

Nach dem erfolgten Start erscheint eine Ausgabe, die etwa wie die folgende aufgebaut ist:

```
Actual parameters: (Info -> Ports must higher than 1024!)

RTP Port:       10000
SIP Port:       5060

Calling Address: sip:default@localhost
**************************************
For a call use c. To quit use q.
To change parameters use first letter (e.g s for the SIP Port).
```

Nun wird zuerst die Addresse des gewünschten Gegenübers eingegeben, bevor ein Anruf getätigt werden kann:

```
a sip:hugo@muster.zhwin.ch                                    <ENTER>
```

Die neue Addresse sollte jetzt im Feld **`Calling Address`** erscheinen. Damit ist man in der Lage einen Anruf einzuleiten. Der Befehlt **`c`** ist für diesen Zweck vorgesehen.

Nun erscheint auf dem anrufenden Client die Meldung:

```
Ringing...
```

i Ein Aufruf des Clients mit **`--help`** zeigt alle verfügbaren Optionen

Auf dem angerufenen Client die Meldung:

```
Received incoming call, accept?
```

Mit dem Befehl **o** kann in dem angerufenen Client der Anruf entgegengenommen werden.

Ein angenommener Anruf wird auf angerufenem und anrufendem Client mit der folgenden Meldung bestätigt:

```
Call accepted, connection established.
```

Nun kann das Gespräch erfolgen. Da in unserem Client noch keine Sprache integriert ist, steht zumindest ein Chat-Modus zur Verfügung. Mit dem Befehl **m,** gefolgt von einem einzeiligen Text wird der Text zum anderen Client übertragen:

```
m Hello World!                                                    <ENTER>
```

Mit **q** wird das Gespräch wieder beendet.

8.3 Gesicherte SIP Verbindung mit S/MIME

Grundsätzlich funktioniert der Ablauf eines gesicherten Gesprächs wie im vorhergehenden Kapitel 8.2 beschrieben. Natürlich dürfen dann die Kommandozeilen-Argumente **--nosmime** und --**nosrtp** nicht angegeben werden.

Der eigentliche Unterschied besteht darin, dass jetzt Private- und Public-Key, Root-CA-Zertifikat und weitere, eigene Zertifikate benötigt werden. Wie die Zertifikate erstellt werden wurde bereits im Kapitel 7.3.2.2 erklärt. Die Zertifikate müssen im Verzeichnis **$HOME/.sipsec/** liegen. Falls die Zertifikate in einem anderen Verzeichnis liegen, kann das Verzeichniss mit der Kommandozeilenoption **--certdir** übergeben werden.

9 Dokumentation der SIP Sicherheitslösung

Inhaltsverzeichnis

9.1 Überblick

Aus der Abbildung 35 ist ersichtlich, dass sich eine VoIP-Verbindung aus der Signalisierung und dem Transport der Media-Daten (Sprache, Video oder Text) zusammensetzt.

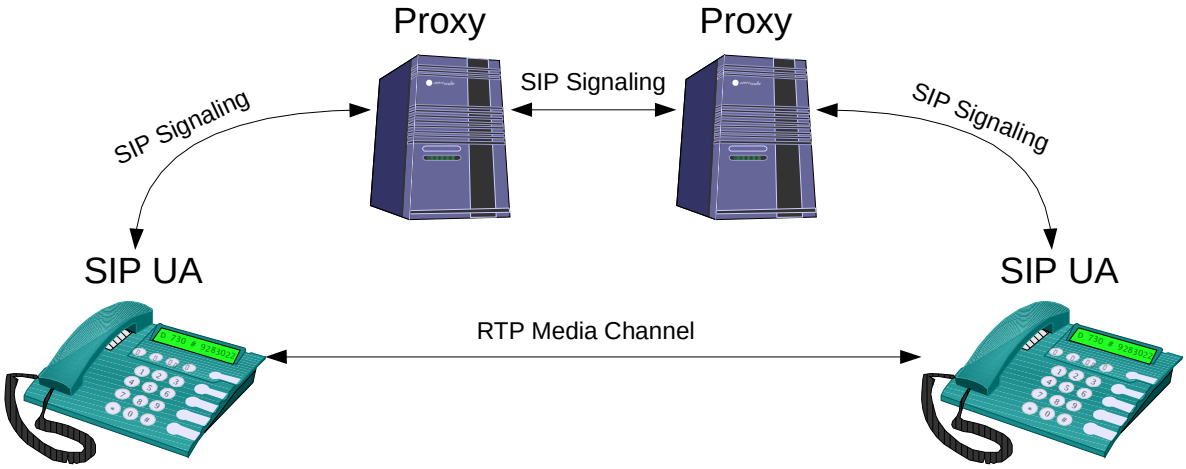


Abbildung 35 Eine ungesicherte VoIP-Verbindung

Bezüglich der Sicherheit wurden folgende Forderungen an die Lösung gestellt:

- **Authentifizierung:** Mittels der Authentifizierung soll die Integrität der Daten überprüft werden. Für diesen Zweck werden digitale Signaturen verwendet. Mit digitalen Signaturen soll sichergestellt werden, dass die Daten auch vom angeblichen Absender stammen und auf dem Weg zum Empfänger nicht durch eine dritte Instanz verändert wurden.
- **Verschlüsselung:** Durch die Verschlüsselung werden die Daten von Klartext in eine nicht erkennbare Form verwandelt. Für die Verschlüsselung wird ein entsprechender Algorithmus verwendet. Mit der Verschlüsselung soll verhindert werden, dass die Daten von Dritten eingesehen oder sogar verändert werden.

Diese Forderungen sind von der SIP-Signalisierung und bei der Übertragung der Media-Daten zu erfüllen. Wie in Abschnitt 6.1 SIP Sicherheitsmechanismen gezeigt, existieren mehrere Mechanismen, um die Sicherheit in SIP zu erhöhen. Es gibt mehrere Gründe die für S/MIME sprechen. S/MIME bietet die meisten Möglichkeiten und ist flexibel, aus diesen Gründen wurde von uns S/MIME gewählt.

In S/MIME werden die zu verschlüsselnden Daten mit einem Secret Key Verfahren gesichert, der dafür verwendete, symmetrische Schlüssel wird zusätzlich mit dem öffentlichen Schlüssel des Empfängers gesichert. Die Signatur wird mit dem privaten Schlüssel des Sender erstellt, sprich, es werden Public Key Verfahren verwendet. Da bei der Sicherung mit S/MIME auch Public Key Verfahren verwendet werden, wird eine Certification Authority (CA) verlangt. Für unsere Zwecke erstellten wir eine eigene CA und die eigenen Client-Zertifikate. Dafür wurde die TinyCA verwendet, die in 7.3.2.2 TinyCA vorgestellt ist.

Für den Transport der Sprache wird in ungeschützten VoIP-Umgebungen das Real-time Transport Protocol (RTP) verwendet. Mit dem Secure Real-time Transport Protocol (SRTP) existiert eine Erweiterung für das Protokoll RTP, welche auch Sicherheitsmechanismen für die Verschlüsselung und Authentifizierung der Daten zur Verfügung stellt. SRTP befindet sich zwar erst im Stadium eines Drafts, dennoch werden damit unsere Media-Daten geschützt. Die Entscheidung ist damit zu begründen, dass SRTP sicher zum Standard, für geschützte VoIP-Umgebungen wird.

Um eine sichere SRTP-Verbindung herzustellen, wird ein symmetrischer Schlüssel benötigt. Es gibt mehrere Möglichkeiten um einen entsprechenden Schlüssel auszuhandeln. Für unsere Zwecke haben wir uns für die Übertragung des Schlüssel im SDP entschieden. Es ist von absoluter Notwendigkeit, dass der symmetrische Schlüssel geheim gehalten wird. Diese Notwendigkeit muss natürlich auch beim Transport des Schlüssel beachtet werden und wird von unserer Lösung auch erfüllt.

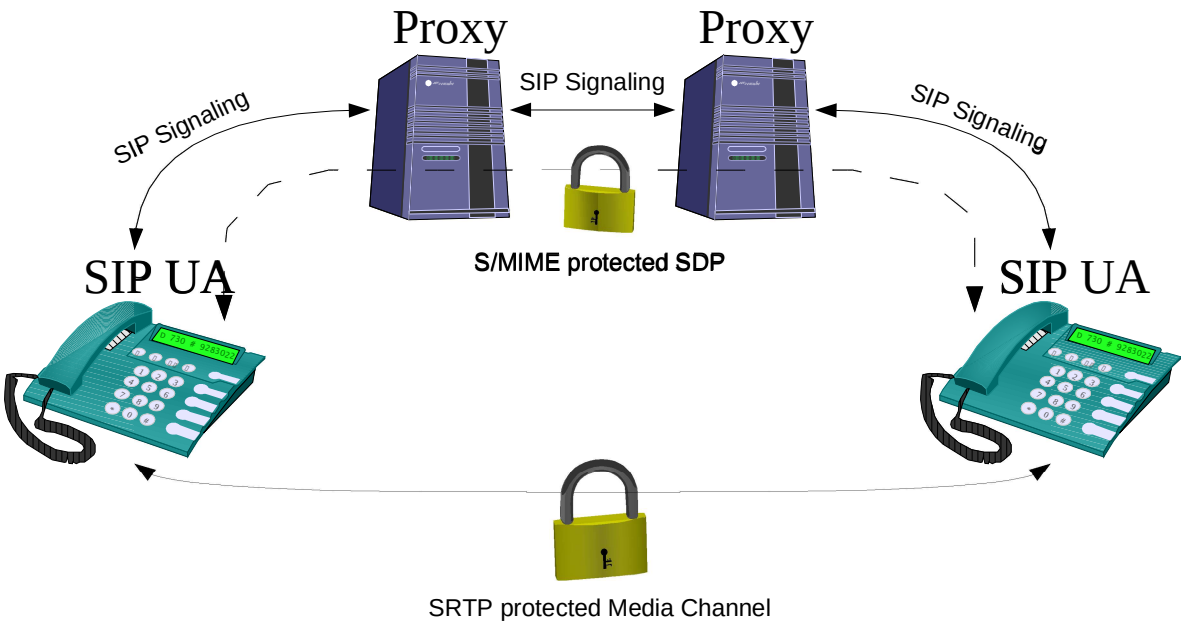


Abbildung 36 Eine gesicherte VoIP-Verbindung

Unten stehendes Beispiel ist aufgeführt, um zu erinnern welche Struktur eine ungesicherte SIP-Nachricht besitzt.

```
INVITE  sip:stricand@dskt6811.zhwin.ch:5060 ;transport=UDP SIP/2.0
To: <sip:stricand@dskt6811.zhwin.ch:5060>
From: <sip:stricand@dskt6811:5000>;tag=563cd63f
Via:  SIP/2.0/UDP  160.85.170.134:5000;branch=z9hG4bK-c87542-911738956-1--
c87542-;rport
Call-ID: f4747a4ea65df208
CSeq: 1 INVITE
Contact: <sip:stricand@dskt6811:5060>
Max-Forwards: 70 Content-Type: application/sdp
User-Agent: SIPSEC
Content-Length: 239

v=0
o=- 3157331353 3157331353 IN IP4 160.85.170.134
s=DA SIP Security 2003 c=IN IP4 160.85.170.134
t=0 0
k=clear:910bc4defa71eb6190008762fca6ae2f1d959e87cdf3c0c5c5076ad38ee8
m=audio 10000 RTP/AVP 0
a=ptime:20
a=rtpmap:0 PCMU/8000
```

Alle Daten werden im Klartext gesendet. Diese Eigenschaft wiegt noch schwerer, da im SDP der Schlüssel für den SRTP-Kanal auch im Klartext vorliegt und somit von Dritten eingesehen werden kann. Eine verschlüsselte SRTP-Verbindung nützt nichts, wenn der Schlüssel auch noch anderen bekannt ist und ein Angreifer leicht in den Besitz des betreffenden Schlüssel kommen kann.

SIP wird dahin gesichert, dass das enthaltene Session Description Protocol (SDP) verschlüsselt und signiert wird. Für diesen Zweck wurde der Inhaltstyp **multipart/signed** gewählt, welcher in Abschnitt Multipart/signed Content Type (Clear Signed Data) erklärt wird. Nach der Anwendung unserer Sicherheitsmechanismen wird aus einer ungeschützten SIP-Verbindung eine gesicherte SIP-Verbindung. Ein Mitschnitt als Beweis für das Funktionieren befindet sich im Anhang unter 12.6.3 Mitschnitt SIP. Zur besseren Übersichtlichkeit wurde der Mitschnitt in 12.6.3 vereinfacht und zeigt die Struktur einer gesicherten Nachricht sehr gut:

```
INVITE sip:stricand@dskt6811.zhwin.ch:5060;transport=UDP SIP/2.0
To: <sip:stricand@dskt6811.zhwin.ch:5060>
From: <sip:stricand@dskt6811:5000>;tag=f855d741
Via: SIP/2.0/UDP 160.85.170.134:5000;branch=z9hG4bK-c87542-911738956-1--
c87542-;
rport
Call-ID: 25d907198cacef54
CSeq: 1 INVITE
Contact: <sip:stricand@dskt6811:5000>
Max-Forwards: 70
Content-Type:
multipart/signed;boundary=992d915fef419824;micalg=sha1;protocol=ap
plication/pkcs7-signature
User-Agent: SIPSEC
Content-Length: 3088

--992d915fef419824
Content-Type:               application/pkcs7-mime;smime-type=enveloped-
data;name=smime.p7m
Content-Disposition: attachment;handling=required;filename=smime.p7
Content-Transfer-Encoding: binary

*****************************************************************
*                                                               *
*                  enveloped-data as described in               *
*              5.3.7.3 EnvelopedData Content Type               *
*                             and                               *
*               5.3.6.5 Enveloped-data Content Type             *
*                      with encapsulated SDP                    *
*                                                               *
*****************************************************************
--992d915fef419824
Content-Type: application/pkcs7-signature;name=smime.p7s
Content-Disposition: attachment;handling=required;filename=smime.p7s
Content-Transfer-Encoding: binary

*****************************************************************
*                                                               *
*           application/pkcs7-signature  as described in        *
*                5.3.7.3 SignedData Content Type,               *
*   5.3.7.3 Multipart/signed Content Type (Clear Signed Data)   *
*                             and                               *
*                 5.3.6.4 Signed-data Content Type              *
*                                                               *
*****************************************************************
--992d915fef419824--
```

Aus dem einfachen Mitschnitt ist sehr gut ersichtlich, dass sich alle zu schützenden Daten im verschlüsselten Teil der Nachricht befinden und somit vor den Augen oder der böswilligen Modifikation Dritter geschützt ist. Der im SDP enthaltene symmetrische Schlüssel für die SRTP-Verbindung ist ebenfalls gesichert und ausser dem bestimmten Empfänger kann dieser Schlüssel von keiner Instanz zurückgewonnen werden. Die Sicherheit des SRTP-Kanals wird also erst durch die Sicherheit der SIP-Nachricht möglich.

Wenn man im Besitz des symmetrischen Schlüssels ist und auch sicher sein kann, dass keine dritte Partei den Schlüssel besitzt, kann eine sichere Kommunikation über SRTP gewährleistet werden. Durch die Verteilung des Schlüssel über das gesicherte SDP wird diese Anforderung gewährleistet. Nachdem man im Besitz des Schlüssels für eine SRTP-Verbindung ist, kann diese erfolgreich erstellt werden. Das Beweisen die im Anhang enthaltenen Mitschnitte. Der Mitschnitt unter 12.6.1 Mitschnitt RTP zeigt eine SRTP-Kommunikation ohne Verschlüsselung und auch ohne Authentifizierung. Ohne Verschlüsselung bedeutet bei unserem verwendeten SRTP-Stack den Gebrauch des Null-Ciphers, welcher den Klartext nicht verändert und unberührt lässt. Durch den Verzicht von Verschlüsselung und Authentifizierung entspricht das SRTP-Paket einem konformen RTP-Paket.

Unter Abschnitt 12.6.2 Mitschnitt SRTP ist ein Mitschnitt enthalten, welcher bei Anwendung von Verschlüsselung und Authentifizierung resultiert. Das Paket in Abschnitt 12.6.2 enthält den gleichen Payload wie das Paket in Abschnitt 12.6.1. Es ist aber zu beachten, dass die Payload nicht mehr in lesbarer Form vorliegt, sondern mit dem AES-CM verschlüsselt wurde. Auch ist ein Unterschied in der Länge des

Paketes sichtbar, welcher 4 Bytes (32 Bits) beträgt. Diese Differenz ist durch das Hinzufügen des Authentication Tags entstanden, welches sich am Ende des SRTP-Paketes befindet. Für die Authentifizierung des Paketes wird der Algorithmus tmmh Version 2 verwendet. Der verwendete Stack stellt nur diesen Algorithmus für die Authentifizierung bereit. Dieser Algorithmus ist im aktuellen Draft nicht mehr enthalten und somit entspricht der Stack nicht mehr dem aktuellen Draft. Dennoch ist der Stack zur Demonstration und zur Sicherung der Media-Daten durch SRTP geeignet.

9.2 Vorteile

- Durch unsere Lösung kann eine End-zu-End-Sicherheit für die SIP und SRTP gewährleistet werden.
- Die Sicherheitsmechanismen sind für die Proxy's transparent und sie müssen kein S/MIME beherrschen. Alle für die für Proxy wichtigen SIP-Informationen sind von der Verschlüsselung nicht betroffen.
- Der symmetrische Schlüssel für die SRTP-Verbindung kann sicher übertragen werden, ist absolut geheim und nur den Partnerinstanzen (UA's) bekannt.

9.3 Schwächen

- Die Proxy's können immer noch Angriffspunkte sein, doch eine Kommunikation kann nur gestört werden. Eine Kommunikation kann abgehört werden.

9.4 Fazit

Es wurde gezeigt, dass die Signalisierung von SIP mit S/MIME gesichert werden kann. Die Daten werden erfolgreich verschlüsselt und können auch erfolgreich authentifiziert werden. Mit der Sicherung von SDP kann auch sichergestellt werden, dass der symmetrische Schlüssel für die SRTP-Verbindung übertragen werden kann, geheim bleibt und nur den Partnerinstanzen bekannt ist. Die Daten für die Kommunikation sind mit Hilfe von SRTP geschützt. Auch diese Daten können erfolgreich verschlüsselt und authentifiziert werden.

10 Probleme

Inhaltsverzeichnis

10.1 setKey Funktion

Für die Übermittlung des Key im SDP ist zwar ein Feld vorhanden, es konnte aber nicht gesetzt werden. Es existierte nur eine Funktion **getKey()** für diesen Parameter. Daher schrieben wir eine zusätzliche Funktion in den Quelltext von reSIProcate, die es ermöglichte, den Schlüssel auf einen Wert zu setzten und im SDP Inhalt mitzugeben. Diese Funktion wurde in der Folge bei der Implementation des Clients verwendet und erfolgreich getestet.

Am 18. Oktober 2003 gab es eine Änderung des Stacks, der neu auch diese Funktion zur Verfügung stellt. Damit ist die von uns geschriebene Funktion nicht mehr notwendig und man kann über die bereitgestellten Funktionen auf den Key im SDP zugreifen.

10.2 Auflösen des Host in die IP Adresse

Das SDP braucht für Verbindungs- und Besitzerinformationen (**o=** und **c=** im SDP Header) die IP Adresse des Rechners (z.B. 160.85.170.136), der die Anfrage stellt. Um für die ersten Tests diese Angaben zu haben, wurden diese Daten fest im Konfigurations-Manager eingegeben, beim Aufruf einer Funktion zurückgegeben und in den SDP Header geschrieben. Am Ende entstand aufgrund unterschiedlicher Funktionsnamen im Konfigurations-Manager ein Fehler. Es wurden nicht alle Teile dieser Testfunktion sauber entfernt. So wurde an einer Stelle immer noch die alte Funktion aufgerufen und dadurch ein seltsamer Fehler produziert, der nicht sofort der Ursache zuzuschreiben war.

Zusätzlich brauchte es auch eine Darstellung der IP-Adresse als unsigend long. Dies führte dazu, dass die komplette Funktion in drei Teile aufgeteilt werden musste, um alle benötigen Zwischenwerte ermitteln zu können.

Die Funktion **ConfigurationManager::getHost()** ermittelt den Namen des lokalen Rechners. Falls dieser nicht aufgelöst werden kann, wird localhost zurückgegeben.

```
String ConfigurationManager::getHost() {
    const int BUF_LEN = 0x0200;     // length of temporary buffer
    SafePt::Mutex::Lock lock(mutex);
    if (thisHost_.empty()) {
        // no hostname specified, try to get it otherwise
        char buf[BUF_LEN];
        if (::gethostname(buf, BUF_LEN-1) < 0) {
            toLog(ERROR,": Cannot get hostname");
            thisHost_ = "localhost";    // set something possible
        } // if
        else thisHost_ = buf;
    } // if (thisHost_.empty())
    lock.unlock();
    return thisHost_;
} // getHost()
```

Eine weitere Funktion ist die **ConfigurationManager::hostToIP()**, welche die übergebene Hostadresse auflöst und die zugehörige IP-Adresse als **unsigned long** zurückgibt. Dieser Wert wird zum Beispiel für das Aufsetzen eines Socket gebraucht. Der struct von **sys/socket.h** benötigt zum Ablegen der IP-Adresse einen **unsigned long**. Weiter muss es möglich sein einen beliebigen Hostnamen (z.B. www.zhwin.ch) aufzulösen, was ebenfalls durch diese Funktion abgedeckt wird.

```
unsigned long ConfigurationManager::hostToIP( String& hostaddr ){
    union hostAddrToLong{
        unsigned long laddr;
        char caddr[4];
    };
    const char *host;
    host =  hostaddr.c_str();
    struct hostent *hostinfo;
    struct in_addr addr;
          char *addr_string;

    hostinfo = gethostbyname(host);
    if (hostinfo == NULL) {
        toLog(ERROR, "invalid query");
        assert(0);
    } // if (hostinfo == NULL)

    if (hostinfo->h_length != 4) {
        toLog(ERROR, "No IPv6 support\n");
        assert(0);
    } // if (hostinfo->h_length != 4)

    hostAddrToLong x;
    for (int i=0; i<4; i++){
        x.caddr[i] = hostinfo->h_addr_list[0][i];
    } // for
    std::cout <<  x.laddr << std::endl;
    return x.laddr;
} // ConfigurationManager::hostToIP()
```

Bei der Funktion **ConfigurationManager::hostToIP()** können Probleme auftreten, wenn die Datei **/etc/hosts** falsch konfiguriert ist. Es ist darauf zu achten, dass diese folgendermassen aussieht:

```
127.0.0.1       localhost
160.85.170.136  dskt6813
```

Die letzte Funktion **ConfigurationManager::convertIPtoString()** liefert dann wirklich die IP-Adresse des Rechners in der Form 160.85.170.136 zurück.

```
String ConfigurationManager::convertIPtoString(unsigned long& ip){
  struct in_addr addr;
  char *addr_string;
  addr.s_addr = ip;
  addr_string = inet_ntoa(addr);
  String ret = addr_string;
  return ret;
} // ConfigurationManager::convertIPtoString()
```

10.3 Linken von statischen Archiven

Auf die Funktionalität der Stacks wird über Archive zugegriffen. Diese Archive müssen beim Linken angegeben werden, damit die Referenzen mit diesen Archiven verbunden werden können. Beim Linken wurde das Archiv angegeben und auch gefunden. Beim Vorgang des Linkens kamen bei den ersten Versuchen auf die Funktionen innerhalb der Archive die Fehlermeldungen 'undefined reference' und das Linken wurde abgebrochen. Wir wussten nicht, dass die Reihenfolge der Argumente beim Linken eine Rolle spielt. Beim Linken müssen zuerst die eigenen Objektdateien angegeben werden und darauf folgend die fremden Archive. Dies ist wichtig zu wissen, da die Rückmeldungen des Linkers nicht gerade aufschlussreich sind und auf andere Fehlerursachen deuten.

10.4 Threadsafe String

Der **std::string** aus der STL von C++ ist nicht geeignet für Multithreading-Anwendungen. Wir hatten von diesem Umstand zwar schon einmal gehört, trotzdem waren wir uns diesem Problem erst richtig bewusst, als es zu einem unerklärlichen Speicherzugriffs-Fehler in einem Allocator kam.

Es musste ein Ersatz für den String aus der Standard Bibliothek gefunden werden. Grundsätzlich gibt es genügend Bibliotheken, welche einen entsprechenden Ersatz anbieten. Zum Beispiel der QString aus der QT Bibliothek **http://www.uwyn.com/projects/tyniq** oder der kleineren TinyQ **http://www.trolltech.com**. Auch im Boost-Projekt **http://www.boost.org** ist ein entsprechender String vorhanden. Damit wäre das Projekt wieder von einer weiteren Bibliothek abhängig gewesen. Da nicht viele String-Funktionen gebraucht werden, wurde eine eigene String-Klasse implementiert. Die Schnittstelle ist kompatibel zu **std::string** ausgelegt, jedoch sind nur die Funktionen implementiert, welche auch benötigt sind.

Wie der String genau implementiert ist, wird im Kapitel 7.2.6.1 erläutert.

10.5 Speicherzugriffsfehler im SipStack Destruktor

Als die ersten Versuche mit dem SIP Stack unternommen wurden, trat im Destructor des Stacks immer ein Speicherzugriffsfehler auf.

Etliche Versuche, den Fehler in unserem Client zu finden scheiterten. Schlussendlich entdeckten wir den Fehler auch in einem der Test-Programme des Stacks, so dass wir davon ausgehen mussten, dass der Fehler im Stack liegt. Wir schrieben ein Beispielprogramm, welches mit minimaler Länge den Fehler reproduzieren konnte und schickten den Fehler der Developer-Mailing-Liste der reSIProcate Entwickler **http://www.resiprocate.org** Die Reaktionen waren durchwegs positiv und der Fehler wurde in kurzer Zeit gefunden und behoben.

Der Fehler kam einerseits dadurch zustande, dass der Multithreaded-Stack noch nicht stabil läuft und andererseits dadurch, dass der Stack mit **SipStack::shutdown()** vor dem Zerstören heruntergefahren werden muss. In Versuchen hatten wir beides korrigiert, aber anscheinend nicht gleichzeitig. Der Fehler, welcher den Zugriffsfehler verursacht hat, wurde mittlerweile korrigiert.

10.6 S/MIME parsing

Gegen das Ende der Diplomarbeit, bei der Integration von S/MIME, sind wir auf einen weiteren Fehler im Stack gestossen.

Beim parsen von soeben entschlüsselten SDP-Paketen verursacht der Stack Fehler in der geparsten Meldung.

Auch diesen Fehler haben wir der Mailing-Liste mitgeteilt und ein Beispielprogramm dazu geschrieben, mit dem der Fehler reproduziert werden kann. Auch hier haben die Entwickler gleich den Fehler gesucht. Diesmal war der Fehler auf ein Design-Problem zurückzuführen, so dass die Entwickler mehrere Tage für die Suche benötigen und bis zum Ende der Diplomarbeit gelöst wurde.

11 Schlusswort

Inhaltsverzeichnis

11.1 Fazit

Diese Diplomarbeit war äusserst lehrreich. Wir konnten durch diese Arbeit viel profitieren und neues Wissen mitnehmen. Die Liste der Gebiete, mit denen wir während der Diplomarbeit in Berührung kamen, ist fast unendlich. Vor allem haben wir sehr viel Neues über VoIP kennen gelernt, welches in Zukunft sicherlich weiter an Bedeutung gewinnen wird.

Eine Menge an theoretischem Wissen konnte aus den gelesen RFC's mitgenommen werden, die im Zusammenhang mit SIP Security zu behandeln waren. Im Speziellen wurden die Themen SIP, SDP, S/MIME und SRTP für den Verlauf der Diplomarbeit benötigt.

Ebenfalls im praktischen Teil konnten neue Erkenntnisse gewonnen werden. Eine weitere Sensibilisierung fand im Bereich der Sicherheit statt, welche in der heutigen Zeit immer wichtiger wird. Durch die Programmierung des Clients konnten viele neue Erkenntnisse gewonnen werden. Speziell die Entwicklung mit C++ wurde praktisch geübt. Im Zusammenhang mit der Entwicklung des Programms, wurde der Umgang mit Linux und die Vor- und Nachteilen von Linux kennen gelernt und weiter vertieft.

Wir realisierten unser erstes grösseres Projekt in C++ mit kanpp 10'000 Zeilen Quelltext. Zusätzlich musste am Anfang sehr viel Zeit in die Einarbeitung investiert werden, da die verwendeten Stacks (SIP und SRTP) etwa 80'000 Zeilen Quelltext umfassen. Für den weiteren Verlauf der Arbeit war es nötig diese beiden Stacks in ihrer Funktionsweise verstanden zu haben. Anhand von reSIProcate sahen wir, was es bedeutet, ein Open Source Projekt in der frühen Entwicklungsphase zu verwenden. Es fehlte an Dokumentation und es wurden bei der Entwicklung Probleme im Stack angetroffen. Der momentane Stillstand des Projektes bei S/MIME kam aufgrund eines Fehlers im SIP Stack zu Stande. So wurde deutlich, dass der Teil mit S/MIME bisher von niemanden ausgiebig getestet oder gar implementiert wurde.

Nebenbei musste die Kommunikation in unserer kleinen Gruppe koordiniert werden, sonst wären Überschneidungen entstanden und unter Umständen Arbeit doppelt oder gar nicht erledigt worden. Hier zeigte sich, dass die Verständigung in der Gruppe sehr gut klappte und keine Probleme bereitete.

Die ganze Diplomarbeit stellte eine sehr gute Ergänzung zu den im Studium besuchten Fächern dar. Vor allem der Besuch des Faches „Sichere Netzwerkkommunikation“ war eine sehr gute Grundlage für diese Diplomarbeit. Im Gegenzug bedeutete die Diplomarbeit eine ausgezeichnete, praktische Ergänzung zum Studium.

Es war ernüchternd zu sehen, wie wenig auf die Sicherheit bei VoIP geachtet wird. Dieser Umstand ist bedenklich, wenn man sich vorstellt, dass VoIP genauso ungeschützt ist, wie es auch andere Anwendungen in IP-Netzwerken sind. Zum Beispiel der Verkehr von E-Mail. Es ist allerdings sicher, so wie VoIP immer weiter an Bedeutung gewinnen wird, so wird auch die Sicherheit bei VoIP immer mehr Aufmerksamkeit bekommen. Aus diesen Gründen war es für uns sehr attraktiv, eine Diplomarbeit in diesem Bereich zu schreiben.

11.2 Ausblick

Mit unserem Client haben wir bewiesen, dass die Signalisierung von SIP mit S/MIME gesichert werden kann. Die in SIP übertragenen Daten können erfolgreich verschlüsselt und erfolgreich authentifiziert werden. Durch die Sicherung von SDP kann sichergestellt werden, dass der symmetrische Schlüssel für die SRTP-Verbindung übertragen werden kann, absolut geheim bleibt und nur den Partnerinstanzen bekannt ist. SRTP schützt die Daten des Media-Streams durch Verschlüsselung und Authentifizierung. Der Client hat somit seinen Zweck erfüllt, nämlich durch „Proof-of-Concept" bewiesen, dass eine sichere VoIP-Kommunikation realisiert werden kann.

Der Client ist aber damit keineswegs abgeschlossen. Unter anderem sind wir durch die Fehler im reSIPreocate-Stack gebremst worden und eine einwandfreie Kommunikation mit S/MIME wird durch einen noch offenen Fehler verunmöglicht. Sobald dieser Fehler behoben sein wird, muss der Client auf die korrekte Funktionsweise kontrolliert werden.

Zu diesem Zeitpunkt können durch den Benutzer nur eingegebene Textnachrichten über den SRTP-Kanal gesendet werden. Als nächster Schritt soll es möglich sein, Sprache über die Soundkarte einzulesen, mit einem Codec zu bearbeiten und dann zur Partnerinstanz zu senden.

Mit Hilfe von SDP wird ein einfaches Verfahren für den Schlüsselaustausch realisiert. Es gibt weitere Verfahren, um einen Schlüssel auszuhandeln. In einem weiteren Schritt könnten weitere Varianten für einen Schlüsselaustausch betrachtet und implementiert werden.

Die Stacks reSIProcate und libsrtp sind unter kontinuierlicher Entwicklung. Der Client muss ständig an die Stacks angepasst werden, um die bereits vorhanden Funktionalität aufrecht zu erhalten.

An dieser Stelle darf auch ein Re-Factoring[i] des Clients nicht vergessen werden. Der Client war für ein „Proof-of-Concept" vorgesehen, dennoch kann er durch ein „Re-Factoring" sehr nützlich für eine weitere Entwicklung sein. Es steckt sehr viel Potenzial im Client, so dass es sich sicher lohnt auf Ihm aufzubauen und zu Ihn erweitern.

i Re-Factoring: Umstellung des Codes, ohne die Funktion zu verändern

11.3 Danksagung

An dieser Stelle wollen wir uns bei allen an der Diplomarbeit beteiligten Personen bedanken. Dank dem betreuenden Dozenten Herrn Prof. Dr. Andreas Steffen hatten wir eine sehr interessante und vielseitige Diplomarbeit. Herr Steffen hat uns an den wöchentlichen Treffen durch diese Diplomarbeit begleitet und hat uns weiter wertvolle Informationen, Tipps und Antworten gegeben.

Daniel Kaufmann hat durch seine grosse Hilfe bei der zugrunde liegenden Evaluation im Rahmen der Projektarbeit ein grosses Teil zum Entstehen dieser Diplomarbeit beigetragen. Für diese Hilfe sind wir Ihm sehr dankbar.

Herrn Jason Fischl war uns während der Diplomarbeit immer wieder eine sehr grosse Hilfe. Für seine schnelle und kompetente Hilfe bei Problemen mit dem reSIProcate Stack sind wir Ihm sehr dankbar.

Mit reSIProcate haben wir einen hervorragenden Stack gefunden. Für diesen grossartigen Stack danken wir dem ganzen Team.

Obwohl sich SRTP erst im Stadium eines Drafts befindet, konnten wird dennoch auf eine Implementation zurückgreifen. Für diesen grossartigen Stack sind wir Herrn David A. McGrew von Cisco sehr dankbar. Auch er hat uns mit seinen kompetenten Antworten weiter geholfen.

Wir wollen auch Herrn T. Ernst für seine Tipps und die Einführung des Thema SIP an der ZHW danken.

12 Anhang

Inhaltsverzeichnis

12.1 Abkürzungen

3DES	Dreifacher DES
AES	Advanced Encryption Standard
AOR	address-of-record
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation one
BER	Basic Encoding Rules
CA	Certification Authoritity
CBC	Cipher Block Chaining
CMS	Cryprtographic Message Syntax
CRL	certificate revocation list
DER	Distinguished Encoding Rules
DES	Digital Encryption Standard
DoS	Denial of Service
DSS	Digital Signature Standard
EDE	Encryption / Decryption / Encryption
FIPS	Federal Information Processing Standard
FSM	Finite State Machine
HTTP	HyperText Transfer Protocol
IDEA	International Data Encryption Algorithm
IETF	Internet Engineering Task Force
ISO	International Standards Organization
IV	Initialisierungsvektor
MAC	Message Authentication Code
Mbone	Multicast Backbone
MD5	Message Digest# 5
MIB	Management Information Base
MIKEY	Multimedia Internet KEYing
MIME	Multipurpose Internet Mail Extension
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
MiM	Man-in-the-Middle
OID	Object Identifier
PEM	Privacy Enhanced Mail
PFS	Perfect Forward Secrecy
PGP	Pretty Goog Privacy

PKCS	Public Key Cryptography Standard
PKI	Public Key Infrastruktur
RC2	Rivest Cipher 2
RC4	Rivest Cipher 4
RFC	Request for comments
RSA	Rivest-Shamir-Adelman
RTSP	Real Time Streaming Protocol
RTP	Real-time Transport Protocol
SA	Security Association
SAP	Session Announcement Protocol
SDP	Session Description Protocol
SHA	Secure Hash Algorithm
SIP	Session Initiation Protocol
S/MIME	Secure MIME
SNMP	Simple Network Management Protocol
SRTP	Secure Real-time Transport Protocol
STL	Standard Template Library, die C++ Standard Bibliothek
TCP	Transmission Control Protocol
TTL	Time To Live
UA	User Agent
UDP	User Datagram Protocol
VoIP	Voice over IP, Telefonie über IP-Netze

12.2 Glossar

ASCII
: Eine Abbildung zwischen Text zeichen und Nummern. ASCII steht für American Standard Code for Information

Base64
: Verschlüsselt die Daten durch Abbildung von 6-Bit-Blöcken der Eingabe auf 8-Bit-Blöcke der Ausgabe, bei denen es sich um druckbare ASCII-Zeichen handelt.

Downgrade Attack
: Der Angreifer versucht die Gesprächspartner zu einem unsicheren oder keinem Verschlüsselungs-Algorithmus zu zwingen.

Early media
: Early media erlaubt einem rufenden Teilnehmer Multimedia-Daten zu empfangen, wie zum Beispiel ein Rufton oder die Stimme des gerufenen Teilnehmers, sobald dieser den Ruf annimmt.

Header
: Ein Header in SIP meint, ein der Methode oder der Statusmeldung folgendes Attribut der Form: Header: Wert(e)

Hop-by-Hop
: Punkt-zu-Punkt Sicherheit: Nur die Kommunikation zwischen zwei Punkten wird verschlüsselt. Jeder Punkt ist in der Lage die Daten zu entschlüsseln und muss deshalb das Vertrauen des Nutzers geniessen.

MD5
: Message Digest 5 [RFC1321]. Cryptographischer Digest-Algorithmus mit 128Bit

Message Digest
: Ein Digest-Algorithmus berechnet aus einer beliebig langen Eingabe (Text, Daten) einen in der Grösse festen Wert. Aus diesem Wert lässt sich die ursprüngliche Nachricht nicht mit vernünfigem Aufwand berechnen (Einwegfunktion).

Methode
: Mit Methode ist in SIP die erste Zeile gemeint, die den Meldungs-Type identifiziert. Anstelle einer Methode gibt es auch Statusmeldungen. Beispiel: INVITE

MIME
: Ermöglicht die Trennung und Kennzeichnung von verschiedenen Daten in E-Mail, HTTP und auch SIP.

PKCS
: steht für „Public-Key Cryptography Standard“. Die Public-Key Cryptography Standards sind von der RSA produzierte Spezifikationen, welche die Absicht haben, den Einsatz von Public-Key Kryptographie zu beschleunigen.

Replay Attack
: Attacke durch wiederholen eines Paketes. Kann durch Seriennummern verhindert werden.

Statusmeldung
: Eine Statusmeldung in SIP dient der Rückmeldung eingegangener Methoden. Typisch ist z.B.: **`SIP/2.0 200 Ok`**

S/MIME
: Schützt MIME-Extensions durch Verschlüsselung und Authentifikation (Integrität). Baut auf MIME auf.

tunneln

Ein Protokoll wird in ein anderes verpackt und so versendet. Das Trägerprotokoll ist dabei in der gleichen oder einer höheren OSI-Schicht angesiedelt.

tmmh

Truncated Multi-Modular Hash Function. tmmh ist eine universelle Hash-Funktion, welche besonders gut für die Authentifizierung von Nachrichten geeignet ist.

Wrapper

Ein Wrapper umhüllt einen anderen Typ (oder ein Konzept) und bietet nur eine andere Schnittstelle ohne neue Funktionalität an.

X.509

Von der ITU festgelegter Standard für die Struktur von Zertifikaten.

12.3 Quellen

Quellenverzeichnis

[PA_SIP] Daniel Kaufmann & Stricker Andreas, SIP Security: Securing SIP based VoIP, 2003

[RFC3261] SIP: Session Initiation Protocol, www.ietf.org/rfc/rfc2543.txt

[SRTP] The Secure Real Time Protocol, www.ietf.org/internet-drafts/

[RFC1889] A Transport Protocol for Real-Time Applications, www.ietf.org/rfc/rfc1889.txt

[X.680] ITU-T, Abstract Syntax Notation One (ASN.1),

[X.690] ITU-T, ASN.1 encoding rules: Specification of Basic Encoding Rules (BER)

[RFC3263] Session Initiation Protocol (SIP): Locating SIP Servers, www.ietf.org/rfc/rfc3263.txt

[RFC3264] An Offer/Answer Model with the Session Description Protocol (SDP), www.ietf.org/rfc/rfc3264.txt

[RFC2976] The SIP INFO Method, www.ietf.org/rfc/rfc2976.txt

[RFC2327] SDP: Session Description Protocol, www.ietf.org/rfc/rfc2327.txt

[RFC2616] Hypertext Transfer Protocol -- HTTP/1.1, www.ietf.org/rfc/rfc2616.txt

[VoIPGW] Ernst Till & Gallizzi Ulrich, Voice over IP Gateway - ISDN2SIP Gateway, 2002

[RFC1890] RTP Profile for Audio and Video Conferences with Minimal Control, http://ietf.org/rfc/rfc1890.txt

[RFC2633] S/MIME Version 3 Message Specification, www.ietf.org/rfc/rfc2633.txt

[RFC2045] Multipurpose Internet Mail Extensions (MIME); Part One, www.ietf.org/rfc/rfc2045.txt

[RFC2046] Multipurpose Internet Mail Extensions (MIME); Part Two, www.ietf.org/rfc/rfc2046.txt

[RFC2047] Multipurpose Internet Mail Extensions (MIME); Part Three, www.ietf.org/rfc/rfc2047.txt

[RFC2048] Multipurpose Internet Mail Extensions (MIME); Part Four, www.ietf.org/rfc/rfc2048.txt

[RFC2049] Multipurpose Internet Mail Extensions (MIME); Part Five, www.ietf.org/rfc/rfc2049.txt

[RFC822] ARPA Internet Text Messages, www.ietf.org/rfc/rfc822.txt

[WIL01] Sicherheit im Internet: Anwendungen und Standards / William Stalings / Addison-Wesley, 2001, ISBN:3-8273-1697-9

[KT07] Thomas Müller, Netzwerk-Applikationen und Protokolle, 2002

[RFC821] Simple Mail Transfer Protocol, www.ietf.org/rfc/rfc821.txt

[STA01] Sicherheit im Internet: Anwendungen und Standards / William Stalings / Addison-Wesley, 2001, ISBN:3-8273-1697-9

[RFC1521] MIME (Multipurpose Internet Mail Extensions); Part One, www.ietf.org/rfc/rfc1521.txt

[PKCS#7] PKCS#7: Cryptographic Message Syntax Standard, 1993

[RFC2315] PKCS #7: Cryptographic Message Syntax Version 1.5, www.ietf.org/rfc/rf2315.txt

[RFC3369] Cryptographic Message Syntax, www.ietf.org/rfc/rfc3369.txt

[RFC1847] Security Multiparts for MIME: Multipart/Signed und Multipart/Encrypted, www.ietf.org/rfc/rfc1847.txt

[RFC2311] S/MIME Version 2 Message Specification, www.ietf.org/rfc/rfc2311.txt
[RFC2119] Key words for use in RFCs to Indicate Requirement Levels, www.ietf.org/rfc/rfc2119.txt
[FIPS-197] Federal Information Processing Standards, Announcing the ADVANCED ENCRYPTION STANDARD (AES), 2001
[MIKEY] Arkko, et al., MIKEY: Multimedia Internet KEYing, 2003
[RFC2543] SIP: Session Initiation Protocol, www.ietf.org/rfc/rfc2543.txt
[RFC2069] An Extension to HTTP: Digest Access Authentication, www.ietf.org/rfc/rfc2069.txt
[RFC2617] HTTP Authentication: Basic and Digest Access Authentication, www.ietf.org/rfc/rfc2617.txt
[RFC2246] The TLS Protocol Version 1.0, www.ietf.org/rfc/rfc2246.txt
[GNUAUTO] GNU Autoconf, Automake, and Libtool / Gary V. Vaughan, Ben Elliston, Tom Tromey, Ian Lance Taylor / New Riders, 2000, ISBN:1-57870-190-2

12.4 Inhalt CD-ROM

/README.TXT
: Beschreibung des Inhaltes und Ergänzungen zu dieser Inhaltsangabe.

/HOWTO
: Installations-Howto

/documentation
: Enthält diese Dokumentation in digitaler Form (PDF, PS und OpenOffice).

/literature
: Enthält die verwendete Literatur und die für die Diplomarbeit relevanten RFC's.

/source
: Hier sind alle Quelldateien die während der Diplomarbeit programmiert wurden enthalten.

/software
: Verwendete Softwarepakete

/software/lib
: Notwendige Bibliotheken

Beachten Sie die README.TXT im Hauptverzeichnis der CD-ROM. Diese enthält Ergänzungen zu dieser Inhaltsangabe.

12.5 Klassendokumentation

Eine genauere Klassendokumentation ist auf der CD im HTML-Format vorhanden. Dort kann komfortabel und übersichtlich in der Klassenhierachie umher gesprungen werden.

12.5.1 Klassen Hierarchie

basic_String< T >	Operation
ConfigurationManager	GenericOperation
Dialog	OperationOffHook
Event	OperationOnHook
EventOffOnHook	OperationRecBye
EventRecAck	OperationRecInvite
EventRecBye	OperationRecOk
EventRecCancel	OperationRecRinging
EventRecInvite	OperationRecTrying
EventRecOk	OperationSendInvite
EventRecRinging	Payload
EventRecTrying	Random
EventSendInvite	reader_thread
EventSrtpExit	Sdp
EventSrtpInit	SdpSessionId
EventSrtpReceive	SipFsmThread
EventSrtpSend	SipWrapper
EventTimeout	Sptr< T >
EventTimer	SptrHelper
GenericEvent	SrtpReceiverThread
Exception	SrtpSenderThread
SRTPException	SRTPWrapper
StringException	SRTPReceiver
Fifo	SRTPSender
FinalStateMachine	State
SipFsm	DebugState
	GenericState
	TimerThread

12.5.2 Klassenbeschreibung

Class	Documentation
basic_String< T >	Eine generische und einfache Implementierung einer String Template Klasse Der STL String wurde ersetzt, da dieser nicht Thread-safe ist. Diese Klasse ist Thread-safe, weil sie kein Copy-on-Write einsetzt, auch wenn das ein wenig Performance kostet.
ConfigurationManager	Der ConfigurationManager speicher mehr oder weniger statische Informationen. Diese Manager speichert nicht nur Port Nummern, lokale URI und andere Einstellungen, sondern er liest auch System-einstellungen aus Umgebungsvariablen und kann Standard-Werte zurückliefern, falls die entspechenden Angaben nicht gemacht wurden. Die Idee hinter dieser Klasse ist nicht, einen Kommunikationskanal zu erhalten, sondern einen zentralen Punkt zum Einstellen und Lesen von mehr oder wenig statischen Informationen. Für die aktive Informations-Übertragung sind die Fifo's die bessere Wahl. Alle öffentlichen Methoden sind mit einem Mutex vor gleichzeitigem Zugriff geschützt. Diese Klasse wendet das Singleton-Pattern an. Verwende die Methode **getInstance()** um auf die Instanz zuzugreiffen.
Dialog	Dialog ist zuständig für das Verfolgen einer SIP-Verbindung. Er hilft beim Erstellen von Meldungen und speichert die Session-Informationen während einem Anruf.
Event	Event ist die übergeordnete Klasse aller Events. Um einen neuen Event zu erzeugen, muss von dieser Klasse abgeleitet werden und die Methode **getEventType()** überschrieben werden. Der Rückgabewert dieser Methode muss eindeutig sein.
EventRecAck	Tritt auf, wenn ein ACK empfangen wurde.
EventRecInvite	Tritt auf, wenn ein INVITE empfangen wurde.
EventRecRinging	Tritt auf, wenn ein „180 Ringing“ empfangen wurde.
EventSrtpExit	Weist den SRTP-Sender und den SRTP-Empfänger an, sich zu beenden.
EventSrtpInit	Initialisiert den SRTP-Sender und den SRTP-Empfänger.
EventSrtpReceive	Weist den SRTP-Empfänger an, in den Empfangs-Zustand zu wechseln.
EventSrtpSend	Weist den SRTP-Sender an, Daten zu senden.
EventTimeout	Tritt auf, wenn ein Timer abgelaufen ist.
EventTimer	Dieser Event wird dazu genutzt, einen Timer zu setzen. Timer wird mit einer Zeit und einem Event ausgestattet und an den TimerThread gesendet. Dieser wird dann zur gewünschten Zeit den enthaltenen Event auslösen.
Exception	Basis-Exception für alle Exception, welche in diesem Client geworfen werden.

Class	Documentation
Fifo	Ein Thread-safe Fifo-Buffer. Diese Klasse realisiert einen Thread-safe Fifo-Buffer für die Kommunikation zwischen den Threads. Diese Klasse benutzt intern eine normale Queue der STL, da diese nicht Thread-safe ist. Für die Synchronisation wird ein pthread-Mutex verwendet.
FinalStateMachine	Diese Klasse repräsentiert eine Zustands-Maschine. Sie baut den Zustands-Graphen und wechselt zwischen den Zuständen. Um eine neue Zustandsmaschine zu erhalten, muss von dieser Klasse abgeleitet werden und die Methode **build()** überschrieben werden. In der **build()**-Methode wird die eigentliche Zustandsmaschine zusammengebaut. Der Zustandsmaschinen-Destruktor ist dafür zuständig, die ganze Hierarchie wieder abzubauen und alle enthaltenen Klassen zu zerstören.
GenericEvent	Generisches Event. Der Rückgabewert von **getEventType()** lässt sich im Konstruktor setzen.
GenericOperation	Generische Operation, wenn kein Code in **process()** ausgeführt werden muss. Lässt sich dort einsetzten, wo eine Aktion notwendig ist.
GenericState	Generischer Zustand, wenn keine Spezialisierung notwendig ist. Wenn keine **onEnter()** oder **onExit()** benötigt wird, kann dieser Zustand verwendet werden.
Operation	Interface für Operation, die bei einem Zustandswechsel ausgeführt werden muss. Die **process()** Methode muss von der abgeleiteten Klasse überschrieben werden. Darin sind die auszuführenden Operationen zu implementieren.
OperationOffHook	Operation, wenn ein Benutzer den Hörer abnimmt.
OperationOnHook	Operation, wenn ein Benutzer den Hörer wieder aufsetzt
OperationRecBye	Operation, wenn ein „BYE“ empfangen wurde.
OperationRecInvite	Operation, wenn ein „INVITE“ empfangen wurde.
OperationRecOk	Operation, wenn ein „200 Ok“ empfangen wurde.
OperationRecRinging	Operation, wenn ein „180 Ringing“ empfangen wurde.
OperationRecTrying	Operation, wenn ein „100 Trying“ empfangen wurde.
OperationSendInvite	Operation, wenn ein „INVITE“ gesendet werden soll
Payload	Diese Klasse enthält die Nutzlast, die über SRTP versendet werden soll.
Random	Stellt einen Zufallsgenerator zur Verfügung. Der Zufallsgenerator wird für die Schlüsselgenerierung des Master- / Salt-Key benötigt. Auch für die Generierung der SDP Session-ID ist sie zuständig. Intern verwendet die Klasse die Funktion RAND_bytes des OpenSSL toolkits der Bibliothek libcrypto.

Class	Documentation
Sdp	Die Klasse erzeugt SDP-Header und kann von empfangenen SDP-Packeten die Werte auslesen.
SipFsm	Die SipFsm definiert das Verhalten des Clients als Haupt-Zustandsmaschine. Einmal instanziert, erstellt es die Struktur der Zustandsmaschine in der **build()**-Methode. Es lohnt sich einen Blick auf diese Methode zu werfen, da das Verhalten sehr kompakt ersichtlich ist. Die Zustandsmaschine muss die aktuelle Instanz des SipWrappers kennen. Dies ist deshalb notwendig, weil einige Operationen mit dem SipWrapper interagieren müssen und selbst nur die SipFsm kennen.
SipFsmThread	Im SipFsmWrapper wurde der Thread für den SIP- und FSM-Teil realisiert. Sie enthält die SipFsm als zentrale Zustandsmaschine.
SipWrapper	Der SIP Protokoll Wrapper ist um den SipStack ge-“wrapped“, um damit einen zentralen Punkt zu erhalten, der schlussendlich mit dem Stack interagiert.

Sptr< T >	Smart Pointer mit Referenzen-Zähl-Algorithmus und automatischem Zerstören der zugewiesenen Objekte nachdem keine Referenz mehr auf den Pointer zeigt.
	Smart Pointer sind mit einem Mutex gesichert, so dass es auch möglich ist, sie über Thread-Grenzen hinweg zu verwenden. Das schützen der Daten auf welche gezeigt wird ist aber immernoch Sache des Anwenders.
	Um Fehler im Zusammenhang mit **Sptr** zu entdecken, kann das Makro DEBUG_SPTR definiert werden, das zu entsprechenden Debug-Ausgaben führt.
	Hinweis: Vermeide zirkuläre Referenzen mit Smart-Pointer
	Die Verwendung verläuft nach folgendem Beispiel:
	`Sptr<MyClass> a;  // new empty smart pointer`
	`{`
	`  // create new pointer`
	`  Sptr<MyClass> p = Sptr<MyClass>(new MyClass(23));`
	`  a = p; // increments internal reference count to 2`
	`}  // at the end of this block, p is`
	`  // destroyed, but not MyClass instance`
	`a->doSomething();`
	`// instance of MyClass(23) still avaiable`
	`// at the end of the block, a is destoryed,`
	`// and with it MyClass`

SptrHelper	Hilfsklasse für **Sptr**. **Hinweis:** Diese Klasse sollte nicht angerührt werden, ausser die Methode **debugInfo()** für Debug-Zwecke.
SRTPException	Diese Klasse enthält die Exception für das SRTP-Modul. SRTPException ist von Exception abgeleitet und wird im SRTP-Modul geworfen.
SRTPReceiver	Diese Klasse implementiert den SRTP-Empfänger. Die Klasse wurde von SRTPWrapper abgeleitet und verwendet jenen Code.

Class	Documentation
SrtpReceiverThread	Diese Klasse implementiert die run-Methode für den SRTP-Empfänger-Thread.
SRTPSender	Diese Klasse implementiert den SRTP-Sender Die Klasse wurde von SRTPWrapper abgeleitet und verwendet jenen Code.
SRTPWrapper	Diese Methode implementiert gemeinsame Funktionen für den SRTP-Stack und wird vom Sender und Empfänger geerbt. Der **SRTPWrapper** enhält Funktionen für **SRTPSender** und **SRTPReceiver**. Deshalb kann auch der SRTP-Stack einfach ausgetauscht werden, wenn dies gewünscht ist.
State	Die Klasse **State** ist die Eltern-Klasse aller Zustände. Um einen neuen Zustand zu erhalten, muss von State abgeleitet werden und die Methoden **onEnter()** bzw. **onExit()** implementiert werden. Wenn keine der beiden Methoden benötigt wird, kann auch einfach eine Instanz der Klasse **GenericState** verwendet werden.
TimerThread	Der **TimerThread** realisert einen Timer für das Versenden von Timer-Events. Dieser Thread wird benötigt um Zeitabhängige Events für die Zustandsmaschine zu erzeugen. Um einen neuen Timer zu setzen muss ein **TimerEvent** an den TimerThread geschickt werden.

12.5.3 Datei-Liste

file	description
configurationmanager.h **configurationmanager.cpp**	Der ConfigurationManager speichert globale Einstellungen
dialog.h **dialog.cpp**	Hilfsklasse um einen SIP-Dialog zu führen
event.h **event.cpp**	Event-Interface um der Zustandsmaschine ein Ereigniss zuzusenden
exception.h **exception.cpp**	Basis Exception für alle Exceptions, die im Client geworfen werden.
Fifo.h **Fifo.cpp**	Ein Thread-safe Fifo
finalstatemachine.h **finalstatemachine.cpp**	Repräsentiert eine Zustandsmaschine
help.h	Hilfsmeldungen für die Kommandozeile
logger.h **logger.cpp**	Log-Meldungs-Funktionen
main.cpp	Hauptprogramm und Benutzer-Interface
operation.h **operation.cpp**	Eine Operation, die ausgeführt wird, wenn ein State gewechselt wird.
Payload.h **Payload.cpp**	Enthält die Nutzlast, welche vom SRTP-Stack versendet wird.
Random.h **Random.cpp**	Stellt einen Zufallszahlen-Generator zur Verfügung
sdp.h **sdp.cpp**	Umgang mit SDP-Protokoll-Daten
sipEvent.cpp **sipEvent.cpp**	Implementierung der Events
sipfsm.h **sipfsm.cpp**	Die SIP-Zustandsmaschine
sipFsmThread.h **sipFsmThread.cpp**	Thread für die SIP-Zustandsmaschine und den SIP-Stack
sipsecstring.h	Eine Thread-safe Implementierung einer String-Klasse
sipwrapper.h **sipwrapper.cpp**	Wrapper um den SIP-Stack
sptr.h **sptr.cpp**	Smart-Pointers mit Referenzen-Zählung und automatischer-Objekt-Zerstörung

file	description
SRTPException.h **SRTPException.cpp**	Exceptions für das SRTP-Modul
SRTPReceiver.h **SRTPReceiver.cpp**	Empfänger-Modul für den SRTP-Stack
srtpReceiverThread.h **srtpReceiverThread.cpp**	Thread in dem der SRTP-Empfänger läuft
SRTPSender.h **SRTPSender.cpp**	Sender-Modul für den SRTP-Stack
srtpSenderThread.h **srtpSenderThread.cpp**	Thread in dem der SRTP-Sender läuft
SRTPWrapper.h **SRTPWrapper.cpp**	Gemeinsame Funktionen für den SRTP-Empfänger und -Sender
state.h **state.cpp**	Ein Zustand in einer Zustandsmaschine
timerThread.h **timerThread.cpp**	Timer für zeitgesteuerte Events.

12.6 Mitschnitte der Kommunikation

12.6.1 Mitschnitt RTP

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 5f 00 00 40 00 40 11  a4 d0 a0 55 aa 89 a0 55   ._..@.@. ...U...U
0020  aa 89 81 54 13 8c 00 4b  28 f5 80 0f 83 77 01 00   ...T...K (....w..
0030  00 00 61 74 73 5d 2b 2b  2b 2b 2b 2b 2b 2b 2b 2b   ..ats]++ ++++++++
0040  2b 2b 2b 2b 2b 2b 2b 2b  2b 2b 48 61 6c 6c 6f 20   ++++++++ ++Hallo
0050  57 6f 72 6c 64 21 21 21  2b 2b 2b 2b 2b 2b 2b 2b   World!!! ++++++++
0060  2b 2b 2b 2b 2b 2b 2b 2b  2b 2b 2b 2b 00            ++++++++ ++++.
```

12.6.2 Mitschnitt SRTP

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 63 00 00 40 00 40 11  a4 cc a0 55 aa 89 a0 55   .c..@.@. ...U...U
0020  aa 89 81 3a 13 8c 00 4f  79 b7 80 0f eb 36 01 00   ...:...O y....6..
0030  00 00 a8 34 2a 68 58 07  f6 66 31 9a 32 35 cf 47   ...4*hX. .f1.25.G
0040  79 f5 bd 74 49 dc 9b c9  28 c6 3e d5 1c 0f ab fd   y..tI... (.>.....
0050  12 30 6e 42 55 1a 7d 22  81 3c ec 2a e0 c3 92 35   .0nBU.}" .<.*...5
0060  72 34 ab b2 60 04 6a 53  ac bf cd 36 09 e6 31 c7   r4..`.jS ...6..1.
0070  7d                                                }
```

12.6.3 Mitschnitt SIP

```
+--------+-------------------------------------------------+------------------+
| offset | 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f | plain text       |
+--------+-------------------------------------------------+------------------+
| 0000000: 49 4e 56 49 54 45 20 73 69 70 3a 73 74 72 69 63 | INVITE sip:stric |
| 0000010: 61 6e 64 40 64 73 6b 74 36 38 31 31 2e 7a 68 77 | and@dskt6811.zhw |
| 0000020: 69 6e 2e 63 68 3a 35 30 36 30 3b 74 72 61 6e 73 | in.ch:5060;trans |
| 0000030: 70 6f 72 74 3d 55 44 50 20 53 49 50 2f 32 2e 30 | port=UDP SIP/2.0 |
| 0000040: 0d 0a 54 6f 3a 20 3c 73 69 70 3a 73 74 72 69 63 | ..To: <sip:stric |
| 0000050: 61 6e 64 40 64 73 6b 74 36 38 31 31 2e 7a 68 77 | and@dskt6811.zhw |
| 0000060: 69 6e 2e 63 68 3a 35 30 36 30 3e 0d 0a 46 72 6f | in.ch:5060>..Fro |
| 0000070: 6d 3a 20 3c 73 69 70 3a 73 74 72 69 63 61 6e 64 | m: <sip:stricand |
| 0000080: 40 64 73 6b 74 36 38 31 31 3a 35 30 30 30 3e 3b | @dskt6811:5000>; |
| 0000090: 74 61 67 3d 66 38 35 35 64 37 34 31 0d 0a 56 69 | tag=f855d741..Vi |
| 00000a0: 61 3a 20 53 49 50 2f 32 2e 30 2f 55 44 50 20 31 | a: SIP/2.0/UDP 1 |
| 00000b0: 36 30 2e 38 35 2e 31 37 30 2e 31 33 34 3a 35 30 | 60.85.170.134:50 |
| 00000c0: 30 30 3b 62 72 61 6e 63 68 3d 7a 39 68 47 34 62 | 00;branch=z9hG4b |
| 00000d0: 4b 2d 63 38 37 35 34 32 2d 39 31 31 37 33 38 39 | K-c87542-9117389 |
| 00000e0: 35 36 2d 31 2d 2d 63 38 37 35 34 32 2d 3b 72 70 | 56-1--c87542-;rp |
| 00000f0: 6f 72 74 0d 0a 43 61 6c 6c 2d 49 44 3a 20 32 35 | ort..Call-ID: 25 |
| 0000100: 64 39 30 37 31 39 38 63 61 63 65 66 35 34 0d 0a | d907198cacef54.. |
| 0000110: 43 53 65 71 3a 20 31 20 49 4e 56 49 54 45 0d 0a | CSeq: 1 INVITE.. |
| 0000120: 43 6f 6e 74 61 63 74 3a 20 3c 73 69 70 3a 73 74 | Contact: <sip:st |
| 0000130: 72 69 63 61 6e 64 40 64 73 6b 74 36 38 31 31 3a | ricand@dskt6811: |
| 0000140: 35 30 30 30 3e 0d 0a 4d 61 78 2d 46 6f 72 77 61 | 5000>..Max-Forwa |
| 0000150: 72 64 73 3a 20 37 30 0d 0a 43 6f 6e 74 65 6e 74 | rds: 70..Content |
| 0000160: 2d 54 79 70 65 3a 20 6d 75 6c 74 69 70 61 72 74 | -Type: multipart |
| 0000170: 2f 73 69 67 6e 65 64 3b 62 6f 75 6e 64 61 72 79 | /signed;boundary |
| 0000180: 3d 39 39 32 64 39 31 35 66 65 66 34 31 39 38 32 | =992d915fef41982 |
```

```
| 0000190: 34 3b 6d 69 63 61 6c 67 3d 73 68 61 31 3b 70 72 | 4;micalg=sha1;pr |
| 00001a0: 6f 74 6f 63 6f 6c 3d 61 70 70 6c 69 63 61 74 69 | otocol=applicati |
| 00001b0: 6f 6e 2f 70 6b 63 73 37 2d 73 69 67 6e 61 74 75 | on/pkcs7-signatu |
| 00001c0: 72 65 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 | re..User-Agent:  |
| 00001d0: 53 49 50 53 45 43 0d 0a 43 6f 6e 74 65 6e 74 2d | SIPSEC..Content- |
| 00001e0: 4c 65 6e 67 74 68 3a 20 33 30 38 38 0d 0a 0d 0a | Length: 3088.... |
| 00001f0: 2d 2d 39 39 32 64 39 31 35 66 65 66 34 31 39 38 | --992d915fef4198 |
| 0000200: 32 34 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 | 24..Content-Type |
| 0000210: 3a 20 61 70 70 6c 69 63 61 74 69 6f 6e 2f 70 6b | : application/pk |
| 0000220: 63 73 37 2d 6d 69 6d 65 3b 73 6d 69 6d 65 2d 74 | cs7-mime;smime-t |
| 0000230: 79 70 65 3d 65 6e 76 65 6c 6f 70 65 64 2d 64 61 | ype=enveloped-da |
| 0000240: 74 61 3b 6e 61 6d 65 3d 73 6d 69 6d 65 2e 70 37 | ta;name=smime.p7 |
| 0000250: 6d 0d 0a 43 6f 6e 74 65 6e 74 2d 44 69 73 70 6f | m..Content-Dispo |
| 0000260: 73 69 74 69 6f 6e 3a 20 61 74 74 61 63 68 6d 65 | sition: attachme |
| 0000270: 6e 74 3b 68 61 6e 64 6c 69 6e 67 3d 72 65 71 75 | nt;handling=requ |
| 0000280: 69 72 65 64 3b 66 69 6c 65 6e 61 6d 65 3d 73 6d | ired;filename=sm |
| 0000290: 69 6d 65 2e 70 37 0d 0a 43 6f 6e 74 65 6e 74 2d | ime.p7..Content- |
| 00002a0: 54 72 61 6e 73 66 65 72 2d 45 6e 63 6f 64 69 6e | Transfer-Encodin |
| 00002b0: 67 3a 20 62 69 6e 61 72 79 0d 0a 0d 0a 30 82 02 | g: binary....0.. |
| 00002c0: f2 06 09 2a 86 48 86 f7 0d 01 07 03 a0 82 02 e3 | ...*.H.......... |
| 00002d0: 30 82 02 df 02 01 00 31 82 01 9f 30 82 01 9b 02 | 0......1...0.... |
| 00002e0: 01 00 30 81 82 30 7d 31 0b 30 09 06 03 55 04 06 | ..0..0}1.0...U.. |
| 00002f0: 13 02 43 48 31 13 30 11 06 03 55 04 07 13 0a 57 | ..CH1.0...U....W |
| 0000300: 69 6e 74 65 72 74 68 75 72 31 0c 30 0a 06 03 55 | interthur1.0...U |
| 0000310: 04 0a 13 03 5a 48 57 31 15 30 13 06 03 55 04 0b | ....ZHW1.0...U.. |
| 0000320: 13 0c 44 41 20 32 30 30 33 20 53 6e 61 32 31 12 | ..DA 2003 Sna21. |
| 0000330: 30 10 06 03 55 04 03 13 09 73 69 70 73 65 63 20 | 0...U....sipsec  |
| 0000340: 43 41 31 20 30 1e 06 09 2a 86 48 86 f7 0d 01 09 | CA1 0...*.H..... |
| 0000350: 01 16 11 6c 6f 72 65 74 6d 61 6e 40 7a 68 77 69 | ...loretman@zhwi |
| 0000360: 6e 2e 63 68 02 01 04 30 0d 06 09 2a 86 48 86 f7 | n.ch...0...*.H.. |
| 0000370: 0d 01 01 01 05 00 04 82 01 00 0c 52 6b da 29 2f | ...........Rk.)/ |
| 0000380: 73 d1 05 42 65 b0 96 87 ad 72 f5 2e d8 7f 13 7d | s..Be....r.....} |
| 0000390: a8 18 07 16 70 18 21 2e ea 23 98 14 64 9a fd ed | ....p.!..#..d... |
| 00003a0: 93 83 fa 8a f5 22 95 36 8d c4 b2 e0 5e 08 10 e4 | .....".6....^... |
| 00003b0: e9 85 16 ca 6c 98 3f c5 38 31 a9 c6 37 33 cc 6e | ....l.?.81..73.n |
| 00003c0: ea 89 88 9a 18 5f 7f 42 e2 73 0e b4 f5 b7 b5 c2 | ....._.B.s...... |
| 00003d0: c6 ca 40 18 52 90 2c f8 81 a7 5a d4 a4 e6 be 8a | ..@.R.,...Z..... |
| 00003e0: 7e e4 2f 35 d9 82 d7 8d fb 51 9c fc a0 1c 3e 8a | ~./5.....Q....>. |
| 00003f0: 44 7a 39 c9 b6 15 c1 fa 33 c2 98 a4 0a e8 80 50 | Dz9.....3......P |
| 0000400: 3a 44 9b 84 a3 c0 15 c1 d2 06 a9 81 37 7e d3 0b | :D.........7~.. |
| 0000410: 82 28 3c 79 48 7c 67 fe 10 30 8f 40 e2 43 df 4f | .(<yH|g..0.@.C.O |
| 0000420: 42 3e b6 de c8 1e d7 2e 72 97 8d c9 ce 8e 15 fa | B>......r....... |
| 0000430: ec 05 0b 12 85 9e 22 70 dd a8 7f 5b e5 c6 68 8c | ......"p...[..h. |
| 0000440: 28 a9 f4 42 c3 be 69 35 79 23 51 b5 c0 81 83 4a | (..B..i5y#Q....J |
| 0000450: e3 24 db c2 f8 85 30 31 12 7e af a3 86 5d 20 da | .$....01.~...] . |
| 0000460: a9 80 6f ca dc 1a 3c 50 70 a7 de 66 1e d8 8e 68 | ..o...<Pp..f...h |
| 0000470: c1 b5 df 42 83 08 73 bc 3a 95 30 82 01 35 06 09 | ...B..s.:.0..5.. |
| 0000480: 2a 86 48 86 f7 0d 01 07 01 30 14 06 08 2a 86 48 | *.H......0...*.H |
| 0000490: 86 f7 0d 03 07 04 08 a7 47 1f 07 4f 3c 25 f9 80 | ........G..O<%.. |
| 00004a0: 82 01 10 12 f7 87 88 bd 04 dd 4f 47 05 dc a7 95 | ..........OG.... |
| 00004b0: 5d 0e 7b 09 47 b8 f8 df 3d 0c 8a 07 bb 5d ca 90 | ].{.G...=....].. |
| 00004c0: a0 8e d1 31 6c 11 91 22 ac 78 30 6b f1 c4 80 cc | ...1l..".x0k.... |
| 00004d0: 6d 53 bc 17 d1 c3 74 a9 86 84 f2 7a e4 a9 a6 2e | mS....t....z.... |
```

```
| 00004e0: c7 53 22 3d 5b e8 a7 d5 35 23 79 aa 6c c3 0f a4 | .S"=[...5#y.l... |
| 00004f0: e7 90 8f 3a 88 ab cf 91 5b 30 49 7f 7e 9a 2b 45 | ...:....[0I.~.+E |
| 0000500: 7b 2c d2 4a 68 28 f2 40 0b 90 83 f9 cc eb 95 11 | {,.Jh(.@........ |
| 0000510: 7a f3 86 68 1d fd 05 80 5f 75 73 e1 9f 33 66 5b | z..h...._us..3f[ |
| 0000520: 2e 65 ea 03 b1 94 0c 83 f1 10 9f 78 7b 1c ab 36 | .e.........x{..6 |
| 0000530: 65 4b c3 ed bb b6 1b 41 4d 8f 78 43 24 33 0d 97 | eK.....AM.xC$3.. |
| 0000540: 3d 91 6c c7 40 b0 1e ee cc c1 00 97 05 d9 20 d1 | =.l.@......... . |
| 0000550: 68 84 08 62 53 de 06 ca c7 2a 5f d6 48 ac 21 ad | h..bS....*_.H.!. |
| 0000560: 4b 33 f2 8e cd 30 c9 9c b5 c4 67 fb 6f 9b af e5 | K3...0....g.o... |
| 0000570: 07 73 f1 79 4d 78 4c 7a 39 6d 09 08 a8 39 15 cc | .s.yMxLz9m...9.. |
| 0000580: 58 11 13 c8 3d c8 84 16 f2 3c e4 6c 08 74 3d cb | X...=....<.l.t=. |
| 0000590: 09 93 24 2c e6 be e4 73 b3 64 2a b2 56 d4 93 35 | ..$,...s.d*.V..5 |
| 00005a0: 94 4f 72 9c 12 1f 5b 28 6c 31 e9 bc 5d c1 a3 d4 | .Or...[(l1..]... |
| 00005b0: bd a7 89 0d 0a 2d 2d 39 39 32 64 39 31 35 66 65 | .....--992d915fe |
| 00005c0: 66 34 31 39 38 32 34 0d 0a 43 6f 6e 74 65 6e 74 | f419824..Content |
| 00005d0: 2d 54 79 70 65 3a 20 61 70 70 6c 69 63 61 74 69 | -Type: applicati |
| 00005e0: 6f 6e 2f 70 6b 63 73 37 2d 73 69 67 6e 61 74 75 | on/pkcs7-signatu |
| 00005f0: 72 65 3b 6e 61 6d 65 3d 73 6d 69 6d 65 2e 70 37 | re;name=smime.p7 |
| 0000600: 73 0d 0a 43 6f 6e 74 65 6e 74 2d 44 69 73 70 6f | s..Content-Dispo |
| 0000610: 73 69 74 69 6f 6e 3a 20 61 74 74 61 63 68 6d 65 | sition: attachme |
| 0000620: 6e 74 3b 68 61 6e 64 6c 69 6e 67 3d 72 65 71 75 | nt;handling=requ |
| 0000630: 69 72 65 64 3b 66 69 6c 65 6e 61 6d 65 3d 73 6d | ired;filename=sm |
| 0000640: 69 6d 65 2e 70 37 73 0d 0a 43 6f 6e 74 65 6e 74 | ime.p7s..Content |
| 0000650: 2d 54 72 61 6e 73 66 65 72 2d 45 6e 63 6f 64 69 | -Transfer-Encodi |
| 0000660: 6e 67 3a 20 62 69 6e 61 72 79 0d 0a 0d 0a 30 82 | ng: binary....0. |
| 0000670: 07 78 06 09 2a 86 48 86 f7 0d 01 07 02 a0 82 07 | .x..*.H......... |
| 0000680: 69 30 82 07 65 02 01 01 31 0b 30 09 06 05 2b 0e | i0..e...1.0...+. |
| 0000690: 03 02 1a 05 00 30 0b 06 09 2a 86 48 86 f7 0d 01 | .....0...*.H.... |
| 00006a0: 07 01 a0 82 04 e2 30 82 04 de 30 82 03 c6 a0 03 | ......0...0..... |
| 00006b0: 02 01 02 02 01 04 30 0d 06 09 2a 86 48 86 f7 0d | ......0...*.H... |
| 00006c0: 01 01 04 05 00 30 7d 31 0b 30 09 06 03 55 04 06 | .....0}1.0...U.. |
| 00006d0: 13 02 43 48 31 13 30 11 06 03 55 04 07 13 0a 57 | ..CH1.0...U....W |
| 00006e0: 69 6e 74 65 72 74 68 75 72 31 0c 30 0a 06 03 55 | interthur1.0...U |
| 00006f0: 04 0a 13 03 5a 48 57 31 15 30 13 06 03 55 04 0b | ....ZHW1.0...U.. |
| 0000700: 13 0c 44 41 20 32 30 30 33 20 53 6e 61 32 31 12 | ..DA 2003 Sna21. |
| 0000710: 30 10 06 03 55 04 03 13 09 73 69 70 73 65 63 20 | 0...U....sipsec  |
| 0000720: 43 41 31 20 30 1e 06 09 2a 86 48 86 f7 0d 01 09 | CA1 0...*.H..... |
| 0000730: 01 16 11 6c 6f 72 65 74 6d 61 6e 40 7a 68 77 69 | ...loretman@zhwi |
| 0000740: 6e 2e 63 68 30 1e 17 0d 30 33 31 30 31 33 30 38 | n.ch0...03101308 |
| 0000750: 35 36 32 36 5a 17 0d 30 34 31 30 31 32 30 38 35 | 5626Z..041012085 |
| 0000760: 36 32 36 5a 30 81 8e 31 0b 30 09 06 03 55 04 06 | 626Z0..1.0...U.. |
| 0000770: 13 02 43 48 31 13 30 11 06 03 55 04 07 13 0a 57 | ..CH1.0...U....W |
| 0000780: 69 6e 74 65 72 74 68 75 72 31 0c 30 0a 06 03 55 | interthur1.0...U |
| 0000790: 04 0a 13 03 5a 48 57 31 15 30 13 06 03 55 04 0b | ....ZHW1.0...U.. |
| 00007a0: 13 0c 44 41 20 32 30 30 33 20 53 6e 61 32 31 23 | ..DA 2003 Sna21# |
| 00007b0: 30 21 06 03 55 04 03 14 1a 73 74 72 69 63 61 6e | 0!..U....strican |
| 00007c0: 64 40 64 73 6b 74 36 38 31 31 2e 7a 68 77 69 6e | d@dskt6811.zhwin |
| 00007d0: 2e 63 68 31 20 30 1e 06 09 2a 86 48 86 f7 0d 01 | .ch1 0...*.H.... |
| 00007e0: 09 01 16 11 73 74 72 69 63 61 6e 64 40 7a 68 77 | ....stricand@zhw |
| 00007f0: 69 6e 2e 63 68 30 82 01 22 30 0d 06 09 2a 86 48 | in.ch0.."0...*.H |
| 0000800: 86 f7 0d 01 01 01 05 00 03 82 01 0f 00 30 82 01 | .............0.. |
| 0000810: 0a 02 82 01 01 00 94 4c 6d 57 11 2b 15 58 37 de | .......LmW.+.X7. |
| 0000820: d3 0b 55 ec 14 b5 9a 2e 06 cc 42 bc 6f 58 e3 f6 | ..U.......B.oX.. |
```

```
| 0000830: 6d bf 16 04 bd 4d c3 43 fe 1a 54 48 12 c9 d6 d2 | m....M.C..TH.... |
| 0000840: 09 01 6f 44 24 fb b3 75 70 cc c4 94 f6 d4 b4 83 | ..oD$..up....... |
| 0000850: 1f 81 a0 a5 0f 7e 15 a7 58 8e 84 9c df 1d f5 03 | .....~..X....... |
| 0000860: ed 59 37 69 95 63 61 73 79 ce db eb 80 79 f7 c1 | .Y7i.casy....y.. |
| 0000870: b5 79 5b 42 2c cb 4d 1c c7 90 c2 ed a2 85 9d b4 | .y[B,.M......... |
| 0000880: b8 f2 2b e6 94 96 58 37 ad 7b 95 df 2f b8 01 86 | ..+...X7.{../... |
| 0000890: 51 27 e7 c0 63 fd a3 78 0d b9 7c f9 6e 2f 2c 7c | Q'..c..x..|.n/,| |
| 00008a0: 4c d3 7d 7a fd 93 76 8f db 2b 14 95 3c ef 46 77 | L.}z..v..+..<.Fw |
| 00008b0: 31 8a 68 a3 d2 d8 5d 11 2f 90 62 ed fa 08 71 82 | 1.h...]./.b...q. |
| 00008c0: b3 a0 5d e8 6a 63 6e da 2d 2b 55 da 68 bb 11 34 | ..].jcn.-+U.h..4 |
| 00008d0: d8 59 3b 38 4a e6 43 85 87 1c a5 c1 12 c4 5c 69 | .Y;8J.C.......\i |
| 00008e0: ca 13 46 b3 17 8b 0e 56 9b f3 50 c1 d2 04 52 a8 | ..F....V..P...R. |
| 00008f0: 3e 62 b8 7d 79 52 70 4c 7a fc 3c c8 01 d2 32 a5 | >b.}yRpLz.<...2. |
| 0000900: 94 ff 4d a7 5a ed 5e a2 77 3f ce 87 f0 c4 d5 1f | ..M.Z.^.w?...... |
| 0000910: 6a e1 3e b1 6f 45 02 03 01 00 01 a3 82 01 55 30 | j.>.oE........U0 |
| 0000920: 82 01 51 30 09 06 03 55 1d 13 04 02 30 00 30 11 | ..Q0...U....0.0. |
| 0000930: 06 09 60 86 48 01 86 f8 42 01 01 04 04 03 02 04 | ..`.H...B....... |
| 0000940: b0 30 2b 06 09 60 86 48 01 86 f8 42 01 0d 04 1e | .0+..`.H...B.... |
| 0000950: 16 1c 54 69 6e 79 43 41 20 47 65 6e 65 72 61 74 | ..TinyCA Generat |
| 0000960: 65 64 20 43 65 72 74 69 66 69 63 61 74 65 30 1d | ed Certificate0. |
| 0000970: 06 03 55 1d 0e 04 16 04 14 ac 38 ad 51 73 d9 dc | ..U.......8.Qs.. |
| 0000980: 71 1f d8 f0 e4 cb be 6d 9d 5e 28 a1 04 30 81 a8 | q......m.^(..0.. |
| 0000990: 06 03 55 1d 23 04 81 a0 30 81 9d 80 14 cd 53 a5 | ..U.#...0.....S. |
| 00009a0: 6f d2 3a 70 3e eb 8e 7e 18 25 0d 2e 1d d2 54 5a | o.:p>..~.%....TZ |
| 00009b0: af a1 81 81 a4 7f 30 7d 31 0b 30 09 06 03 55 04 | ......0}1.0...U. |
| 00009c0: 06 13 02 43 48 31 13 30 11 06 03 55 04 07 13 0a | ...CH1.0...U.... |
| 00009d0: 57 69 6e 74 65 72 74 68 75 72 31 0c 30 0a 06 03 | Winterthur1.0... |
| 00009e0: 55 04 0a 13 03 5a 48 57 31 15 30 13 06 03 55 04 | U....ZHW1.0...U. |
| 00009f0: 0b 13 0c 44 41 20 32 30 30 33 20 53 6e 61 32 31 | ...DA 2003 Sna21 |
| 0000a00: 12 30 10 06 03 55 04 03 13 09 73 69 70 73 65 63 | .0...U....sipsec |
| 0000a10: 20 43 41 31 20 30 1e 06 09 2a 86 48 86 f7 0d 01 |  CA1 0...*.H.... |
| 0000a20: 09 01 16 11 6c 6f 72 65 74 6d 61 6e 40 7a 68 77 | ....loretman@zhw |
| 0000a30: 69 6e 2e 63 68 82 01 00 30 1c 06 03 55 1d 11 04 | in.ch...0...U... |
| 0000a40: 15 30 13 81 11 73 74 72 69 63 61 6e 64 40 7a 68 | .0...stricand@zh |
| 0000a50: 77 69 6e 2e 63 68 30 1c 06 03 55 1d 12 04 15 30 | win.ch0...U....0 |
| 0000a60: 13 81 11 6c 6f 72 65 74 6d 61 6e 40 7a 68 77 69 | ...loretman@zhwi |
| 0000a70: 6e 2e 63 68 30 0d 06 09 2a 86 48 86 f7 0d 01 01 | n.ch0...*.H..... |
| 0000a80: 04 05 00 03 82 01 01 00 98 7a c6 df 84 fb d4 46 | .........z.....F |
| 0000a90: 96 f8 bb eb 81 5c e2 aa e7 43 3f 42 b1 99 de 3c | .....\...C?B...< |
| 0000aa0: f2 25 3d 2a 9c da d6 c1 8a 69 80 cb 90 b2 bb fa | .%=*.....i...... |
| 0000ab0: 20 24 d3 26 84 02 83 dc 62 db ca 7d 44 7f fa 60 |  $.&....b..}D..` |
| 0000ac0: d7 cf 16 54 6b 73 7e a1 b3 84 fd 46 6f 46 fa de | ...Tks~....FoF.. |
| 0000ad0: 43 85 b0 7c 2b 6e 63 04 83 b3 38 ba 42 d3 15 9f | C..|+nc...8.B... |
| 0000ae0: 73 33 95 d3 f1 7a c9 68 70 4b 68 65 dd fb 16 5c | s3...z.hpKhe...\ |
| 0000af0: 1b 96 4f a3 01 5e 14 93 76 e5 df 35 50 89 4e 89 | ..O..^..v..5P.N. |
| 0000b00: 38 47 19 62 25 45 28 60 a6 8e 80 72 34 6e 1d 8b | 8G.b%E(`...r4n.. |
| 0000b10: 31 34 ac 19 71 fc c4 69 6a 4f 01 bf 78 db c9 0f | 14..q..ijO..x... |
| 0000b20: 28 15 87 83 63 70 85 42 e7 8a be 61 85 0d 06 6a | (...cp.B...a...j |
| 0000b30: 9c 42 09 73 26 d2 17 39 03 94 17 f3 15 7a 62 e6 | .B.s&..9.....zb. |
| 0000b40: 7c 92 fa 2b 6a 11 fd 0e d2 47 dc 1c bb 12 28 87 | |..+j....G....(. |
| 0000b50: fa bd 5f 48 d6 3b 47 84 a9 b5 c6 13 2a 42 97 89 | .._H.;G.....*B.. |
| 0000b60: 02 25 45 93 fd fd 74 61 1c 53 0b 99 26 f4 e8 39 | .%E...ta.S..&..9 |
| 0000b70: b7 43 df 42 f6 31 4a 7b 49 f7 60 b3 cd 94 09 37 | .C.B.1J{I.`....7 |
```

```
| 0000b80: c4 30 75 43 20 ad 35 c9 31 82 02 5e 30 82 02 5a | .0uC .5.1..^0..Z |
| 0000b90: 02 01 01 30 81 82 30 7d 31 0b 30 09 06 03 55 04 | ...0..0}1.0...U. |
| 0000ba0: 06 13 02 43 48 31 13 30 11 06 03 55 04 07 13 0a | ...CH1.0...U.... |
| 0000bb0: 57 69 6e 74 65 72 74 68 75 72 31 0c 30 0a 06 03 | Winterthur1.0... |
| 0000bc0: 55 04 0a 13 03 5a 48 57 31 15 30 13 06 03 55 04 | U....ZHW1.0...U. |
| 0000bd0: 0b 13 0c 44 41 20 32 30 30 33 20 53 6e 61 32 31 | ...DA 2003 Sna21 |
| 0000be0: 12 30 10 06 03 55 04 03 13 09 73 69 70 73 65 63 | .0...U....sipsec |
| 0000bf0: 20 43 41 31 20 30 1e 06 09 2a 86 48 86 f7 0d 01 |  CA1 0...*.H.... |
| 0000c00: 09 01 16 11 6c 6f 72 65 74 6d 61 6e 40 7a 68 77 | ....loretman@zhw |
| 0000c10: 69 6e 2e 63 68 02 01 04 30 09 06 05 2b 0e 03 02 | in.ch...0...+... |
| 0000c20: 1a 05 00 a0 81 b1 30 18 06 09 2a 86 48 86 f7 0d | ......0...*.H... |
| 0000c30: 01 09 03 31 0b 06 09 2a 86 48 86 f7 0d 01 07 01 | ...1...*.H...... |
| 0000c40: 30 1c 06 09 2a 86 48 86 f7 0d 01 09 05 31 0f 17 | 0...*.H......1.. |
| 0000c50: 0d 30 33 31 30 32 31 31 31 31 30 31 34 5a 30 23 | .031021111014Z0# |
| 0000c60: 06 09 2a 86 48 86 f7 0d 01 09 04 31 16 04 14 05 | ..*.H......1.... |
| 0000c70: 63 fa 4a f8 8f d9 41 a5 14 40 a2 4d 66 f2 56 0b | c.J...A..@.Mf.V. |
| 0000c80: 6e 51 6b 30 52 06 09 2a 86 48 86 f7 0d 01 09 0f | nQk0R..*.H...... |
| 0000c90: 31 45 30 43 30 0a 06 08 2a 86 48 86 f7 0d 03 07 | 1E0C0...*.H..... |
| 0000ca0: 30 0e 06 08 2a 86 48 86 f7 0d 03 02 02 02 00 80 | 0...*.H......... |
| 0000cb0: 30 0d 06 08 2a 86 48 86 f7 0d 03 02 02 01 40 30 | 0...*.H.......@0 |
| 0000cc0: 07 06 05 2b 0e 03 02 07 30 0d 06 08 2a 86 48 86 | ...+....0...*.H. |
| 0000cd0: f7 0d 03 02 02 01 28 30 0d 06 09 2a 86 48 86 f7 | ......(0...*.H.. |
| 0000ce0: 0d 01 01 01 05 00 04 82 01 00 8b a6 36 f8 62 b8 | ............6.b. |
| 0000cf0: 04 d0 da d4 d0 56 f7 4a 39 e5 a3 25 ca 7f 17 6c | .....V.J9..%...l |
| 0000d00: e1 17 20 70 c2 95 41 03 99 09 4c c4 ea f2 f8 e3 | .. p..A...L..... |
| 0000d10: f5 5c 55 b0 50 47 3d b3 28 23 08 a4 72 41 54 ba | .\U.PG=.(#..rAT. |
| 0000d20: 7f 88 99 2e 77 7b 8b 35 8f ce 00 21 1d 38 ec f3 | ....w{.5...!.8.. |
| 0000d30: d8 32 d3 12 9c 72 a1 23 47 8e 90 a5 f7 a5 b4 70 | .2...r.#G......p |
| 0000d40: a2 38 8b 30 ad 50 04 46 2f 86 17 cc 7b 20 06 b5 | .8.0.P.F/...{ .. |
| 0000d50: 5a 9f 68 1c 7f c4 73 03 93 a2 b3 82 56 d3 af f0 | Z.h...s.....V... |
| 0000d60: d9 b2 f4 1a 68 be e8 ae a0 02 49 61 f4 3c c5 cd | ....h.....Ia.<.. |
| 0000d70: 59 5a df 4f 16 4c 66 52 d4 11 bc b7 a7 8f 71 41 | YZ.O.LfR......qA |
| 0000d80: b1 67 76 4d c3 5b f1 a3 a4 9a f5 37 be 49 a1 1e | .gvM.[.....7.I.. |
| 0000d90: 1d 8b 33 a6 76 63 13 e5 ae 45 e3 62 39 3e 56 0f | ..3.vc...E.b9>V. |
| 0000da0: 48 43 bb 11 33 aa c2 fc de e9 4a 7b ea 2c da 37 | HC..3.....J{.,.7 |
| 0000db0: 0c eb 9c 58 09 3b 0e 08 78 00 43 52 77 b7 58 3c | ...X.;..x.CRw.X< |
| 0000dc0: dc a5 6f 81 50 df 5f e4 52 b3 74 06 ea 4f 80 21 | ..o.P._.R.t..O.! |
| 0000dd0: 15 f2 9f 44 ca 38 75 83 43 4d 13 a6 2f 61 c5 78 | ...D.8u.CM../a.x |
| 0000de0: 21 e1 24 62 6c ff af d6 39 e2 0d 0a 2d 2d 39 39 | !.$bl...9...--99 |
| 0000df0: 32 64 39 31 35 66 65 66 34 31 39 38 32 34 2d 2d | 2d915fef419824-- |
+--------+-------------------------------------------------+------------------+
```

12.7 Mini-Programm

```
#include <iostream>
#include <memory>
#include <cassert>
#include <resiprocate/SipStack.hxx>
#include <resiprocate/Helper.hxx>
#include <resiprocate/SipMessage.hxx>
#include <resiprocate/Security.hxx>
#include <resiprocate/SdpContents.hxx>
#include <resiprocate/Pkcs7Contents.hxx>
#include <resiprocate/MultipartSignedContents.hxx>
#include <resiprocate/Contents.hxx>
using namespace resip;

int main(int argc, char *argv[]) {
    try {
        ////////////////////////
        // initialize stack //
        ////////////////////////

        SipStack sip(false);
        try {
            sip.addTransport(UDP, 5060);
//            sip.addTransport(TCP, 5060);
        } catch (Transport::Exception e) {
            std::cout << "cannot add Transport" << std::endl;
            assert(0);
        }
        sip.security->loadAllCerts("", "../..");

        ////////////////////
        // make a message //
        ////////////////////

        NameAddr from;
        from.uri().scheme() = Data("sip");
        from.uri().host() = Data("localhost");
        from.uri().user() = Data("user");
        from.uri().port() = 5060;

        NameAddr target;
        NameAddr contact;

        target = contact = from;
//        target.uri().scheme() = Data("sip");
//        target.uri().host() = Data("dskt6608.zhwin.ch");
//        target.uri().user() = Data("user");
//        target.uri().port() = 5060;

        std::auto_ptr<SipMessage> message;
        message  = std::auto_ptr<SipMessage>(Helper::makeInvite(target, from, contact));

        ////////////////////////
        // add sdp contents //
        ////////////////////////

        unsigned long sessionId = 123456789UL;
        Data host = "localhost";
        Data address = "127.0.0.1";
        int port = 5005;
        SdpContents::Session::Encryption::KeyType keyType;
        keyType = SdpContents::Session::Encryption::Clear;
        Data key = "7aabfc64b755b2d560bfbbc73fe9f341d81c213b6034825ecb8923d97762";
        SdpContents *sdp = new SdpContents();
        SdpContents::Session::Origin origin("-", sessionId, sessionId,
                                            SdpContents::IP4, address);
        SdpContents::Session session(0, origin, "DA SIP Security 2003");
        SdpContents::Session::Medium medium("audio", port, 0, "RTP/AVP");
        medium.addFormat("0");
        medium.addAttribute("rtpmap", "0 PCMU/8000");
        medium.addAttribute("ptime", "20");
        session.addMedium(medium);
        session.connection().setAddress(address);
        session.encryption().method() = keyType;
        session.encryption().key() = key;
        sdp->session() = session;
```

```
        ////////////////////////////////////
        // encrypt contents and send message //
        ////////////////////////////////////

        Pkcs7Contents *encrypted;
        encrypted = sip.security->encrypt(sdp, "stricand@dskt6811.zhwin.ch");
        message->setContents(encrypted);

        sip.send(*message);

        ////////////////////
        // receive message //
        ////////////////////

        for (;;) {
            FdSet fdset;
            sip.buildFdSet(fdset);
            if (fdset.selectMilliSeconds( sip.getTimeTillNextProcessMS()) == -1) {
                std::cout << "error! fdset" << std::endl;
                assert(0);
            }
            sip.process(fdset);
            SipMessage *msg = NULL;
            msg = sip.receive();
            if (msg) {
                std::cout << ">>>>> got message! <<<<<" << std::endl;
                Contents *recvContents = msg->getContents();
                SdpContents *recvSdp = NULL;
                Pkcs7Contents *pkcs7 = dynamic_cast<Pkcs7Contents*>(recvContents);
                if (! pkcs7 ) {
                    std::cout << "could not convert" << std::cout;
                    assert(0);
                }
                recvContents = sip.security->decrypt(pkcs7);
                if (! recvContents) {
                    std::cout << "could not decrypt" << std::endl;
                    assert(0);
                }
                recvSdp = dynamic_cast<SdpContents*>(recvContents);
                if (! recvSdp ) {
                    std::cout << "could not convert" << std::endl;
                    assert(0);
                }
                std::cout << recvSdp->session().getEncryption().getKey() << std::endl;
                break;
            }
        }
        sip.shutdown();
    } catch (...) {
        std::cout << "error, exception" << std::endl;
    }
    return 0;
}
```