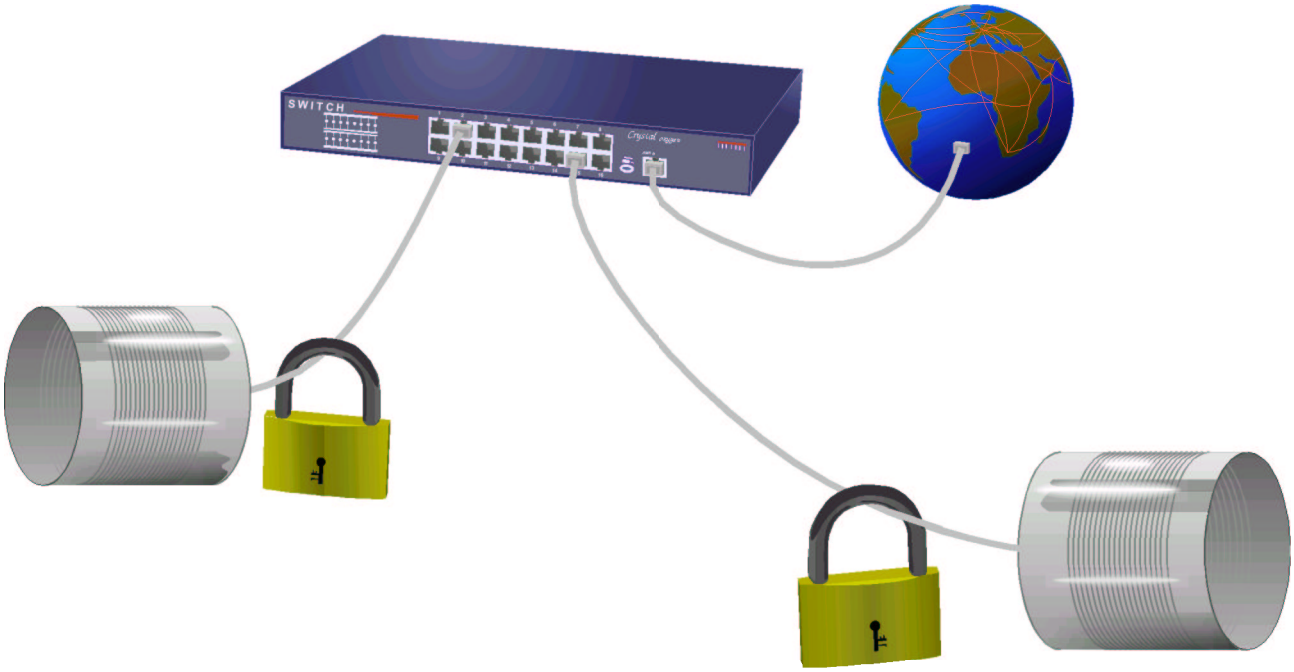
Zürcher
Hochschule
Winterthur

SIP Security

Securing SIP based VoIP



PA2 Sna 03/02

Daniel Kaufmann
Andreas Stricker

Dozent: Andreas Steffen

Inhalt

1 Zusammenfassung

Zunehmend, wenn auch nicht mit dem prophezeiten Erfolg, gewinnen Voice-over-IP-Lösungen (VoIP) immer mehr Beachtung. Als Nachfolger bisheriger Signalisierungs-Protokolle gewinnt SIP (Session Initiation Protocol) immer mehr Bedeutung.

Mit der grösseren Verbreitung wird es immer wichtiger, die Sicherheit der VoIP-Infrastruktur zu gewährleisten. Der SIP-Standard sieht verschiedene Sicherheitsmechanismen vor. Im Rahmen dieser Projektarbeit wurden die Möglichkeiten zur Absicherung von SIP zusammengestellt. Die Sicherheitsmechanismen wurden auf ihre Wirksamkeit und praktische Verwendbarkeit untersucht.

Mit dem Absichern der SIP-Kommunikation ist die Arbeit noch nicht getan: Der Medienkanal (für Sprache oder Video) will ebenfalls vor Lauschaktionen gesichert sein.

Wir setzten uns zuerst allgemein mit der SIP-Infrastruktur auseinander und arbeiteten uns dann in die Sicherheitsmechanismen ein. Dafür bauten wir eine einfache SIP-Infrastruktur mit einem Server und zwei Clienten auf. An dieser Infrastruktur wurden auch alle verfügbaren Sicherheitsmechanismen getestet. Alle Versuche wurden unter GNU/Linux mit dem Open Source System Vovida Vocal durchgeführt, dass eine vollständige SIP-Infrastruktur mitbringt.

Anrufe gehen normalerweise über einen Server. Dieser kann mittels einer Digest-Authentifikation das Endgerät identifizieren und vice-versa. Die Authentifikation mit zusätzlicher Verschlüsselung kann hier auch mit Hilfe von TLS (SSL) erfolgen.

Für die sichere Kommunikation zweier Endgeräte sieht der SIP-Standard S/MIME vor. Damit können sich die beiden Endgeräte nicht nur identifizieren, sondern es kann auch ein Schlüssel für den Sprachkanal sicher ausgetauscht werden. Das in der ersten Version von SIP vorgesehene PGP wurde aus dem aktuellen Standard gestrichen.

Für die Verschlüsselung des Sprachkanals eignet sich die sichere Variante von RTP, genannt SRTP. Da dieses Protokoll auf ein weiteres Protokoll für den Schlüsselaustausch angewiesen ist, setzt dieses auf ein eigenes Schlüssel-Austausch-Protokoll namens MIKEY. Für einfache Anwendungen lässt sich hier auch das bereits erwähnte S/MIME einsetzen.

Wir untersuchten, wie weit sich eine im Standard nicht vorgesehene Lösung mit IPsec, für SIP und Sprachübertragung eignet. Zwischen Telefon und Server ist eine einfache Lösung möglich, für den Medienkanal muss eine dynamische Verbindungsmethode implementiert werden.

Zukünftige SIP-IP-Telefone sollten S/MIME und SRTP mit MIKEY unterstützen. Damit sind sie in der Lage in den meisten Umgebungen die bestmögliche Sicherheit zu gewährleisten.

Momentan fehlen den SIP-Implementationen gute Sicherheitsmechanismen, welche die Akzeptanz der IP-Telefonie weiter fördern können. Vermutlich ist es nur eine Frage der Zeit, bis diese Schwäche verschwindet.

Winterthur, 4.7.2003

Daniel Kaufmann Andreas Stricker

2 Grundlagen SIP

2.1 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 dessen Parameter und sorgt für einen korrekten Abbau der Verbindung. Neben dem typischen Verbindungsaufbau zweier Gesprächspartner kann SIP auch mehrere Teilnehmer verbinden; ist 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 die Lokation 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]

2.2 SIP Infrastruktur

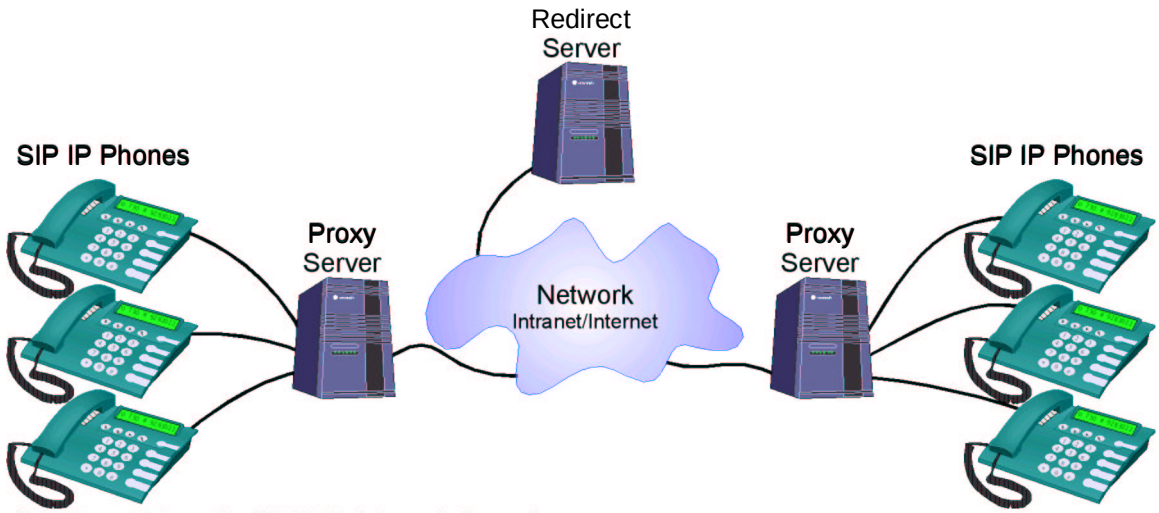


Abbildung 1 ***Typische SIP IP Telefonie Infrastruktur***

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 machbar.

Üblicherweise wird die SIP-Signalisation über eine Proxy/Redirect-Server Infrastruktur abgewickelt. In Abbildung 1 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

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

Lokation der UAs kennt der Redirect-Server[i]. 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.

2.3 SIP Meldungen und Signalisations-Fluss

2.3.1 Einfaches Gespräch

Ein Gespräch läuft idealerweise so wie in Abbildung 2 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) Alice (11) über die Proxys (10) 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[ii] 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).

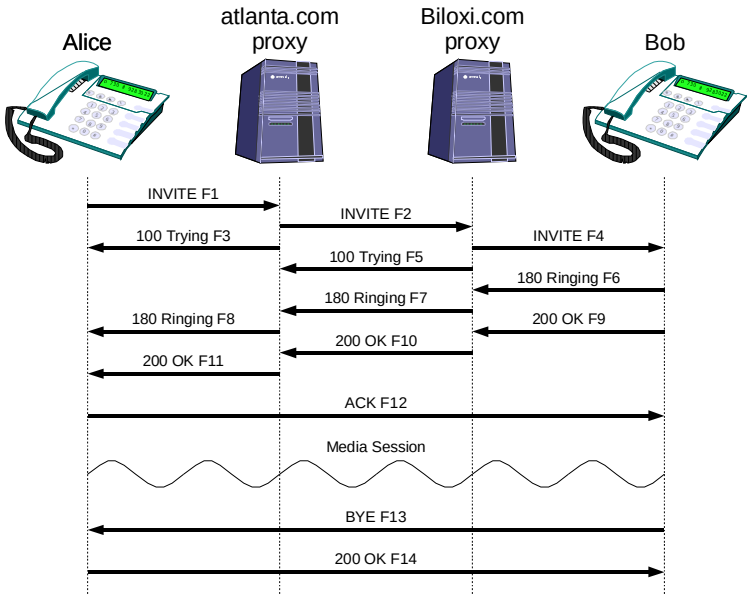


Abbildung 2 SIP Session aus RFC 3261 Seite 11

2.3.2 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 in Abbildung 1 sind zu beachten.

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

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

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

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

Tabelle 1 Reservierte Ports für SIP

2.3.3 SIP Nachricht

Im Folgenden schauen wir tiefer in das Protokoll anhand echter Mitschnitte einer einfachen Verbindung. Dazu wurde der Versuchsaufbau wie in Abbildung 3 dargestellt aufgebaut.

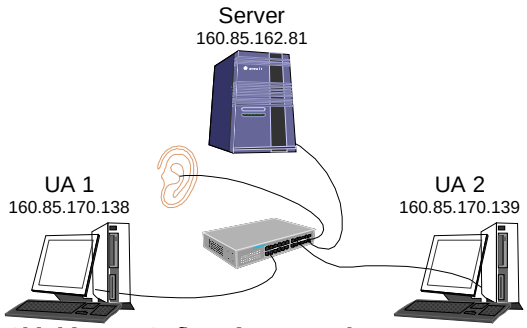


Abbildung 3 Aufbau des Versuches

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.

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 mit **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.

2.3.4 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 Message kann bereits Informationen zum Typ der Session enthalten.
ACK	Dieser Request bestätigt den Erhalt einer entgültigen 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 Message 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.

Table 2 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

Table 3 SIP Status Codes

2.3.5 Redirection

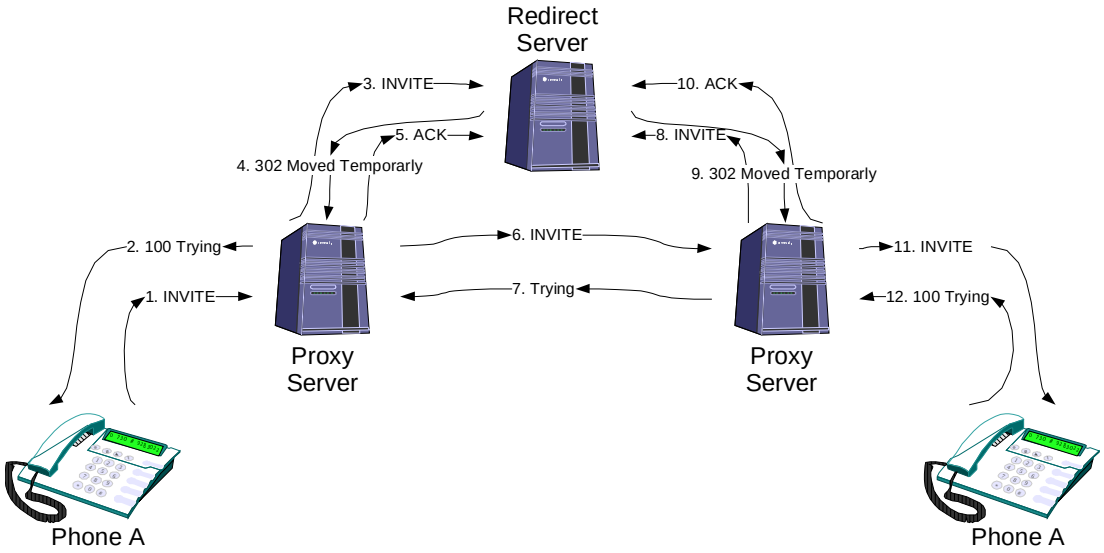


Abbildung 4 Funktion des Redirect-Server

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.

Im 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.

3 Installation der SIP-Infrastruktur

3.1 Wahl des Sip-Stacks

Die Auswahl von SIP-Implementationen ist fast unüberschaubar. Viele diese Produkte sind kommerziell und bei den meisten Produkten ist eine Lizenzgebühr fällig.

Bei der Auswahl eines Produktes sind wir auf den Vorschlag in der Aufgabenstellung eingegangen und haben das Produkt Vocal der Firma Vovida gewählt. Dieser SIP-Stack ist weit verbreitet und entsprechend gut dokumentiert. Vocal ist frei verfügbar und der Quelltext ist offen gelegt[i]. Weiter bietet Vocal nicht nur eine Implementation des Session-Initiation-Protokolls (SIP) sondern liefert gleich eine komplette Infrastruktur zum Aufbau einer umfangreichen VoIP Umgebung auf SIP Basis, deren Grösse und Komplexität nur durch die verwendete Hardware begrenzt wird. Bei dem Aufbau der Versuchsumgebung wurden mehrheitlich Applikationen verwendet, die in Vocal integriert sind.

3.2 Versuchsaufbau

3.2.1 Topologie

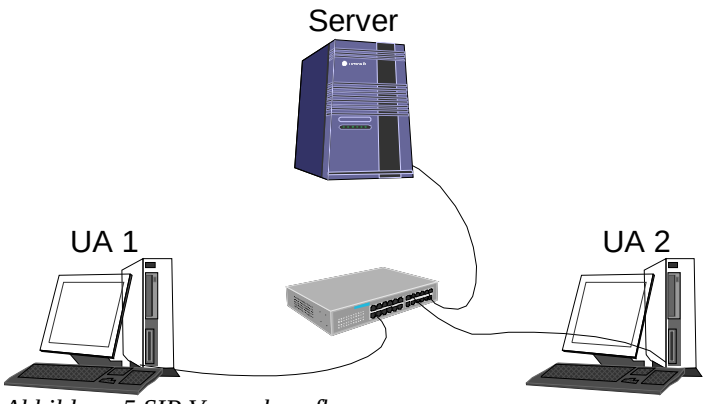


Abbildung 5 SIP Versuchsaufbau

Der Versuchsaufbau besteht aus zwei User-Agents (UA) und einem zentralen Server, der als kleine Telefonzentrale agiert.

i Quelltext kann modifiziert und weiterverbreitet werden, solange das Copyright der Autoren nicht entfernt wird.

3.2.2 Systemkonfiguration

Server	Hostname: dskt6621 (Hugo) Services: Proxy-Server, Registrar-Server, HTTP-Server, SIP-User-Agent (gua) Betriebssystem: Linux Debian IP Adr: 160.85.162.81
User-Agent 1	Hostname: dskt6815 Services: SIP-User-Agent (SIPSet) Betriebssystem: Linux RedHat 7.2 IP Adr: 160.85.170.138
User-Agent 2	Hostname: dskt6816 Services: SIP-User-Agent (SIPSet) Betriebssystem: Linux Knoppix 3.2 (Debian) IP Adr: 160.85.170.139

Es wird an dieser Stelle verzichtet, detailliert auf die Systemkonfiguration der einzelnen Maschinen einzugehen. Weiterführende Informationen sind im Vocal Installation Guide zu finden [VOVIDA].

3.3 Installation Server

Die Server-Installation besteht aus den einzelnen Services wie SIP-Proxy, Registrar usw. In der Regel werden diese Dienste auf derselben Maschine installiert und betrieben. Diese Services können bei wachsendem Netz ohne weiteres auf weitere Server ausgelagert werden. Vocal enthält alle nötigen Services für eine komplette Infrastruktur.

Um den Server zu einem späteren Zeitpunkt bequem mit grafischen Werkzeugen zu konfigurieren sollte auf dem Zielsystem auch eine Java-Laufzeitumgebung installiert sein.

3.3.1 Installation mittels RPM

Das Vorgehen zur Installation von Vocal auf dem Server unterscheidet sich je nach benutzter Linux-Distribution markant. Am einfachsten ist die Installation unter einem RedHat System mittels RPM. Vocal besteht aus den zwei RPM-Paketen **`vocalbin-1.5.0-20.i386.rpm`** (Grundsystem) und **`vocalbin-tools-1.4.0-17.i386.rpm.`** Diese Pakete sind im dem Internet auf der Vovida Homepage oder auf der beigelegten CD zu finden. Die Installation wird unter Root-Rechten mit den Befehlen **`rpm -i vocalbin-1.5.0-20.i386.rpm`** und **`rpm -i sipset-1.5.0.i386.rpm`** gestartet und das Installationsscript sollte nun ohne Fehler abgearbeitet werden. Zur Grundkonfiguration wird nun das Script ausgeführt um eine Testumgebung einzurichten:

```
/usr/local/vocal/allinoneconfigure/allinoneconfigure
```

3.3.2 Installation ab Quelltext

Die Installation von Vocal kann auch auf einem System erfolgen, das nicht auf RedHat basiert. Beim Versuchsaufbau zu dieser Projektarbeit haben wir uns entschlossen, das Vocal-System auf einem Debian GNU/Linux (Woody) aufzusetzen. Dazu ist es notwendig, Vocal ab Quelltext auf dem Zielsystem zu installieren. Um den Quelltext erfolgreich zu übersetzten, ist etwa 1,5

GByte freier Plattenplatz nötig. Der Quelltext wird entpackt (**tar xzf vocal-1.5.0.tar.gz**) und in das erstellte Vocal Verzeichnis gewechselt. Nun wird das System mit folgenden Befehlen installiert.

```
$ ./configure –with-openssl
$ make
$ make install
```

Wird die Installation auf einem System ohne X durchgeführt, kann make nicht ohne Fehler durchgeführt werden. Die GTK-Header werden dafür benötigt. Unter Debian installiert **apt-get install libgtk2.0-dev** gleich das gesamte X. Ausserdem kann das SIPSet sowieso nicht auf diesem Rechner verwendet werden.

Auf einem Vocal-Server wird weder X noch ein grafischer User-Agent benötigt. Um dieses Problem zu lösen ist im Rahmen dieser Projektarbeit ein Installationsscript entstanden, dass die gesamte Serverinstallation ohne X und ohne grafischen UA bewerkstelligt. Das Installationsscript ist im Anhang C abgebildet. Um den Quelltext zu kompilieren wird nun das Script **makesip.sh** eine Ebene unter dem Quelltext-Verzeichnis gestartet. Das Übersetzten wird je nach Hardware einige Stunden in Anspruch nehmen. Einzig das Übersetzten und Installieren von SIPSet, dem grafischen User-Agent, sollte eine Fehlermeldung erzeugen.

Um den übersetzten Code nun zu installieren, bzw. an die richtigen Orte zu kopieren, wird **make install** ausgeführt. Nach Ablauf des Scripts ist die Installation nun komplett.

Für die Grundkonfiguration wird noch das Konfigurationsscript ausgeführt:

```
/usr/local/vocal/allinoneconfigure/allinoneconfigure
```

Im folgenden Dialog werden einige Fragen zur Konfiguration der IP-Adresse, Ports, Logfiles usw. gestellt. Bei den meisten Antworten können die vorgeschlagenen Standartwerte übernommen werden.

3.4 Installation und Konfiguration Clients

Die Softwareinstallation der Clients beschränkt sich auf die Installation eines grafischen User-Agents.

3.4.1 Vorbereitungen

Das System, auf dem ein SIP-User-Agent installiert wird, sollte Netzwerkzugriff haben. Weiter sollte ein DNS-Server bekannt und erreichbar sein (Datei **/etc/resolv.conf** mit Eintrag für Domainname und Nameserver richtig konfiguriert). Zusätzlich muss die Konfigurationsdatei **/etc/hosts** kontrolliert und falls nötig angepasst werden, so dass neben dem Loopback-Device auch eine weitere IP-Adresse mit zugehörigem Hostnamen aufgeführt ist.

Damit sind die notwendigen Installationsvorbereitungen abgeschlossen.

3.4.2 Installation und Konfiguration unter Knoppix 3.2 (Debian)

Um den User-Agent unter Debian einfacher zu installieren, wurde von uns ein deb-Package angefertigt. So muss das Grundsystem und der User-Agent nicht selbst kompiliert werden und es entfallen einige mögliche Fehlerquellen. Das Debian Package **sipset_1.5.0-1_i386.deb** ist

auf der beigelegten CD zu finden. Das Paket wird mir **dpkg -i sipset_1.5.0-1_i386.deb** installiert. Die Installation sollte ohne Fehlermeldung ablaufen. Eine zusätzliche Konfiguration ist nicht notwendig.

3.4.3 Installation und Konfiguration unter RedHat 7.2

Der User-Agent lässt sich mit zwei RPM-Paketen installieren. Die entsprechenden Pakete **vocalbin-1.5.0-20.i386.rpm** (Grundsystem) und **sipset-1.5.0.i386.rpm** (Grafischer User-Agent) sind auch auf der CD zu finden, oder auf der Vovida Homepage [VOVIDA].
Die Installation wird unter Root-Rechten mit den Befehlen **rpm -i vocalbin-1.5.0-20.i386.rpm** und **rpm -i sipset-1.5.0.i386.rpm** gestartet und sollte ohne Fehler ablaufen.

Zur Grundkonfiguration ist auch hier nach der Installation das Konfigurationsscript **allinoneconfigure** zu starten.

4 Bedienung und Konfiguration

4.1 Der grafischen User-Agent SIPSet

Der Grafische User-Agent SIPSet ist ein bequem zu bedienendes Softphone, um Anrufe zu tätigen und entgegenzunehmen.

4.1.1 Start und Bedienung

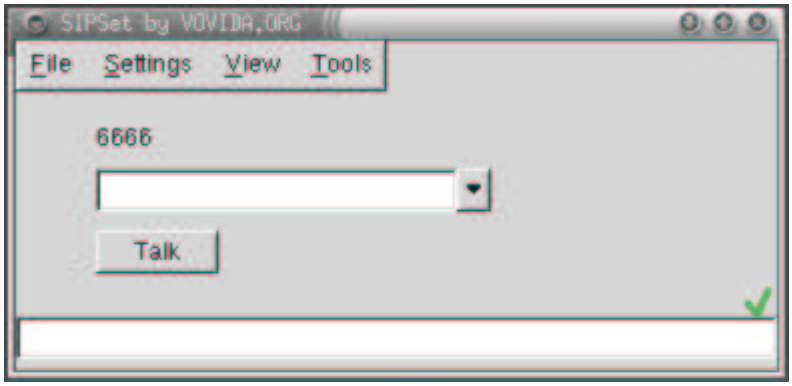


Abbildung 6 SIPSet

Das SIPSet ist unter dem Pfad **/usr/local/vocal/bin/sipset** zu finden. Die grafische Oberfläche des SIPSet ist in Abbildung 6 dargestellt. Es wird beim Start automatisch der zuletzt aktive Benutzer beim Proxy registriert. In der Abbildung ist der Benutzer „6666“ aktiv und erfolgreich beim SIP Proxy-Server angemeldet, was durch den grünen Haken signalisiert wird. Benutzerdaten lassen sich unter dem Menüpunkt Settings – Basic Configuration konfigurieren, indem die jeweiligen Felder für Benutzername, SIP Proxy-Server und Passwort angepasst werden. Nach der Bestätigung sollte wiederum am rechten unteren Rand des Fensters ein grüner Haken erscheinen. Falls statt diesem Haken ein rotes Kreuz erscheint, ist die Anmeldung beim Proxy fehlgeschlagen und die Einstellungen sollten überprüft werden.

Einen Anruf tätigen

Im ersten Textfeld kann nun die URI des gewünschten Gesprächspartners eingegeben werden. Beispiel: **hugo@dskt6621.zhwin.ch**. Mit dem Button „Talk“ wird die Verbindung initiiert und im untersten Textfeld wird der aktuellen Status der Verbindung angezeigt. Nach Beendigung des Gesprächs wird die Verbindung durch nochmaliges Betätigen des Button „Hang up“ wieder abgebaut.

Einen Anruf entgegennehmen

Wenn ein Anruf eingeht, erscheint auf der Arbeitsfläche eine entsprechende Nachricht mit Informationen über den Anrufenden. Es kann nun der Anruf mit den Button „Accept“ angenommen oder mit „Reject“ abgelehnt werden.

4.1.2 Konfiguration

Um weitere Einstellungen an SIPSet vorzunehmen wird der Menüpunkt Settings – Advanced Configuration gewählt (Siehe Abbildung 7) Unter dem Abschnitt „General“ kann neben dem

angezeigten Benutzernamen das Protokoll der Signalisierung gewählt werden. Zur Auswahl stehen UDP und TCP.

Im Abschnitt „User Registration“ kann neben den normalen Benutzereinstellungen unter Expires die Gültigkeitsdauer der Registrierung angegeben werden. Läuft diese ab, muss sich der UA neu beim Proxy registrieren.

Unter „Media“ kann das Gerät zur Sprach-Ein- und Ausgabe gewählt werden. Sowohl die UDP-Ports, die für SIP reserviert sind, als auch eine eventuell vorhandene öffentliche IP-Adresse, falls der Proxy hinter einem NAT-Gateway betrieben wird.

Im Weiteren können noch die Logging-Informationen konfiguriert werden. Es kann ein Logfile, bzw. Logfile-Pfad vorgegeben und ein Log-Level angeben werden. Für normalen Betrieb wird der Modus LOG_ERR empfohlen.

Es existiert im Weiteren noch eine Checkbox zum aktivieren eines Videostreams. Bei unserem Versuchsaufbau wurde diese Option nicht getestet.

Die Logging-Informationen (Logfile) können unter dem Menüpunkte „View“ betrachtet werden. Und unter dem Menüpunkt „Tools“ kann das Logfile wieder gelöscht werden.

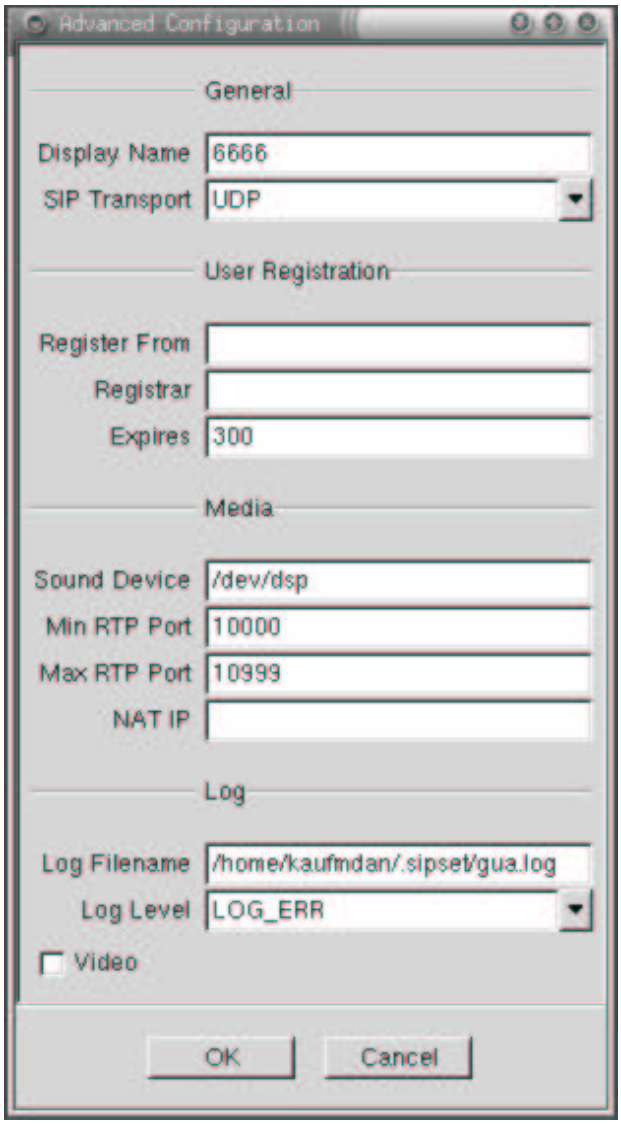


Abbildung 7 SIPSet Advanced Configuration

4.2 Kommandozeilen basierender User-Agent gua

Das kommandozeilenbasierende Softphone ist ein einfacher User-Agent, dessen Funktionsvielfalt aber weiter geht als jene von SIPSet. Die Bedienung ist natürlich nicht so intuitiv wie jene des grafischen User-Agents. Das gua Softphone ist unter dem Pfad **`/usr/local/vocal/bin/gua`** zu finden.

4.2.1 Konfiguration

Das gua Softphone wird mittels einer Konfigurationsdatei konfiguriert. Um die Konfiguration möglichst einfach zu gestalten ist eine gut kommentierte Beispiel-Konfigurationsdatei unter dem Pfad **`/usr/local/vocal/etc/ua.cfg`** erreichbar. Zur Illustration sind hier Teile einer Konfigurationsdatei aus unserem Versuchsaufbau abgebildet. Zur Vereinfachung wurden einige Kommentarzeilen weggelassen.

```
# Your phone number
User_Name                   string              5555
```

User_Name bezeichnet die Nummer oder den Namen unter der der Teilnehmer erreichbar sein soll. Der Username ist Teil der URI.

```
# IP address of SIP Proxy
Proxy_Server                string              160.85.162.81
```

Proxy_Server ist die IP-Adresse, unter welcher der Proxy-Server erreichbar ist, um den SIP-Teilnehmer anzumelden.

```
# Your password
Pass_Word                   string              hugo

# Name to display instead of User Name
Display_Name                string              hugo
```

Pass_Word ist das Benutzerpasswort, abgespeichert im Klartext (Konfigurationsdatei nur durch Benutzer lesbar).

```
# Transport can be either TCP or UDP
SIP_Transport               string              UDP
```

Das verwendete Transport-Protokoll zur Signalisierung. Zur Auswahl stehen TCP und UDP.

```
# IP of machine this user agent is registering from
Register_From               string              160.85.170.138
```

Register_From enthält die IP-Adresse des Clients, welcher sich beim Proxy registriert. (IP-Adresse des Client-Rechners)

```
# IP of registrar, if different from SIP Proxy
Register_To                 string              160.85.162.81
```

Register_To ist die IP-Adresse unter welcher der Registrar erreichbar ist. Diese Adresse ist nur anzugeben, falls die Registrierung über einen anderen Server als den Proxy läuft.

```
# Time (in seconds) until registration expires
Register_Expires   int                 60000
```

Register_Expires bezeichnet die Gültikeitsdauer der Registrierung in Sekunden. Läuft diese ab, muss sich der User-Agent neu beim Proxy anmelden.

```
# Name of sound card (ie /dev/dsp)
Device_Name                 string              /dev/dsp
```

Device_Name ist die Gerätedatei, unter der die Soundkarte (/dev/dsp) oder das Telefongerät (/dev/phone) zu erreichen ist.

```
# Lowest port number in range of RTP ports
Min_RTP_Port                string              10000
# Highest port number in range of RTP ports
Max_RTP_Port                string              10999
```

Min_RTP_Port und **Max_RTP_Port** legen den Bereich der UDP-Ports fest, in der die Sprachkanäle aufgebaut werden. Die Ports zwischen 10000 und 10999 sind bei den meisten SIP-User-Agents für die Sprachübertragung vorgesehen.

```
# Public IP where RTP should be routed
# NATAddress_IP    string              <put value here>
```

Bei **NATAddress_IP** kann die IP-Adresse für ein eventuelles NAT-Gateway angegeben werden, falls der Proxy nicht direkt über eine öffentliche IP-Adresse erreichbar ist.

```
# Name of file for log messages
LogFilename                 string    /home/hugo/.sip/gua.log
# Use LOG_ERR for normal operation, LOG_DEBUG_STACK for debugging
LogLevel                    string              LOG_ERR
```

Die Logging-Informationen werden mit **LogFilename** und **LogLevel** konfiguriert. **LogFilename** gibt dabei den vollständigen Pfad zur Logdatei an und **LogLevel** den Logging-Modus. Für den normalen Betrieb sollte der Wert **LOG_ERR** gewählt werden.

Es existieren noch weitere Punkte, die zu dieser Basiskonfiguration hinzugefügt werden können. So besteht die Möglichkeit Videotelefonie zu betreiben, sich an einem Konferenzserver anzumelden usw. Die abgebildeten Felder genügen, um den gua für eine einfache Sprachverbindung zu konfigurieren. Weitere Konfigurationsmöglichkeiten sind aus dem Kommentar der Beispiel-Konfigurationsdatei oder der Manpage (**man gua**) zu entnehmen.

4.2.2 Start und Bedienung

Beim Start des gua muss der Pfad der Konfigurationsdatei angeführt von der Option -f angegeben werden. Der gua User-Agent wird beispielsweise mit **./gua -f /usr/local/vocal/etc/hugo.cfg** gestartet. Nach dem Start versucht sich der UA beim Proxy anzumelden. Bei erfolgreicher Registrierung erscheint der Prompt **gua>** und signalisiert Bereitschaft. Mit der Taste **?** wird eine kurze Hilfe mit den möglichen Befehlen angezeigt.

Einen Anruf tätigen

Die Verbindung mit einem Gesprächspartner wird mit dem Kommando **call**, gefolgt von der URI des gewünschten Teilnehmers aufgebaut. Bsp: **gua> call hugo@atlanta.com**. Der aktuelle Status der Verbindung wird nun angezeigt.

Einen Anruf entgegennehmen

Ist der UA erfolgreich gestartet und beim Proxy registriert, erscheint bei eingehendem Anruf die Meldung: **gua> ringing !!**. Es wird beim gua nicht angezeigt, welcher Teilnehmer das Gespräch wünscht. Die Verbindung kann entweder mit der Taste **z** (Hang up) zurückgewiesen oder mit **a** (Off Hook) akzeptiert werden, wobei der Sprachkanal sofort aktiviert wird. Nach Beendigung des Gesprächs kann die Verbindung mit **z** wieder abgebaut werden.

5 SIP Sicherheitsmechanismen

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

5.1 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.

5.2 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`**.

Beispiel einer Digest Authentification

Im Folgenden ist das Beispiel aus 2.3.3 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 2.3.3 weiter. Die folgenden Pakete sind nicht mehr durch eine Authentifikation geschützt.

5.3 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.

5.4 S/MIME

In die Fussstapfen von PGP tritt mit Version 2 von SIP 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. Jedoch verlangt S/MIME nach einer Public-Key Infrastruktur um mittels Zertifikaten die jeweiligen Partner zu identifizieren.

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 FAX oder der Tausch in einem vorausgehenden Treffen.

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

Abbildung 4 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 Übertragungsprotokollen 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 des 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 4 Einige MIME-Typen die üblicherweise in SIP mit S/MIME verwendet werden

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.

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.

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.

5.5 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 2.3.2.

6 Medien-Transportprotokolle

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

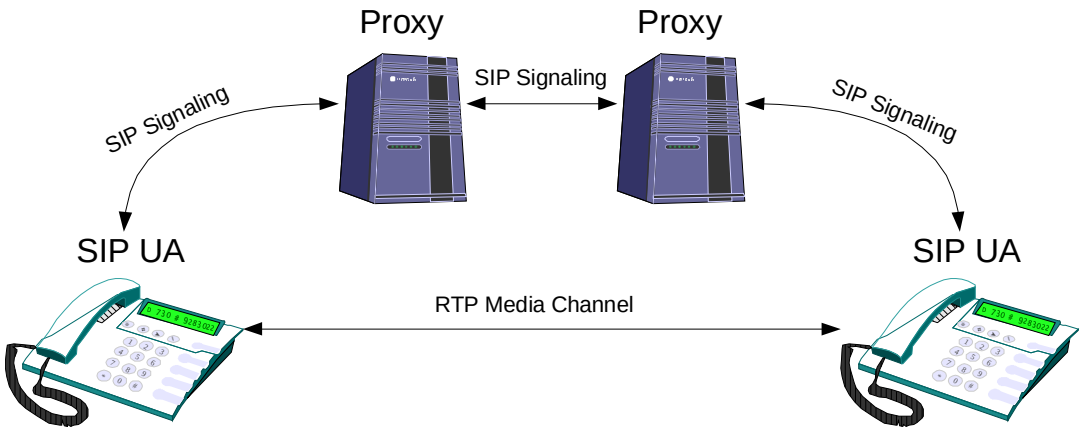


Abbildung 8 Bedeutung des Medien-Kanals

6.1 Real-Time-Protocol (RTP)

Internet-Telefonie ist eine 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. Das Real-Time-Protokoll, wird üblicherweise über UDP transportiert, kann aber auch auf anderen Transportprotokollen aufsetzten.

Die RTP-Übertragung setzt sich üblicherweise aus zwei Protokollen zusammen. Zum eigentlichen Nutzdatentransport wird das Real-Time-Transportprotokoll (RTP) verwendet. Zum Transport von Status-, Monitoring- und Qualitätsmeldungen wird auf das Real-Time-Controll-Protocoll (RTCP) zurückgegriffen.

6.1.1 RTP-Paketaufbau

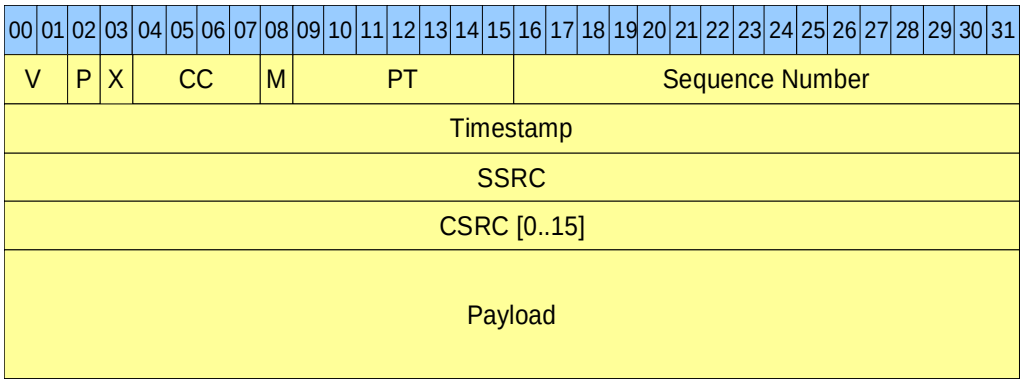


Abbildung 9 RTP-Header Version 2

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.)
Sequence Number:	Eine Nummer, deren Startwert zufällig gewählt wird, die bei jedem RTP-Paket in einen Datenstrom inkrementiert wird.
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, die beteiligten Quellen an einem Datenstrom identifiziert. Die Anzahl Elemente in dieser Liste wird durch das CC Feld gegeben.

Der Payload besteht nun aus den Daten, in dem Format wie es im Headerfeld PT definiert wird. Bei SIP Internet-Telefonie ist dies z.B. ein Audiostrom in G.711 µLaw codiert.

6.1.2 RTCP-Protokoll

Das Real-Time-Controll-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 10) und einem oder mehreren Report-Blocks. Bei diesen Report-Blocks wird wiederum zwischen Sender-Report SR (Abbildung 11) und Receiver-Blocks RR (Abbildung 12) unterschieden, wobei der Sender-Report vom Sender an den Empfänger und der Receiver-Report vom Empfänger and den Sender geschickt wird.

RTCP-Header

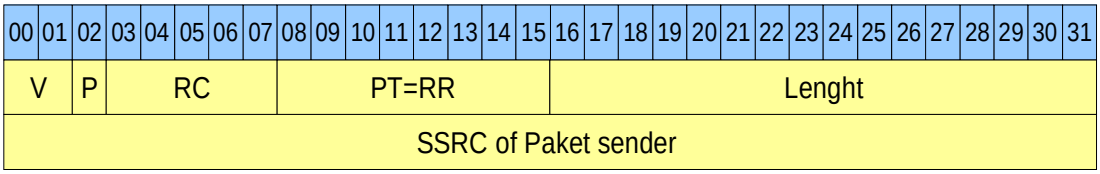


Abbildung 10 RTCP-Header

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

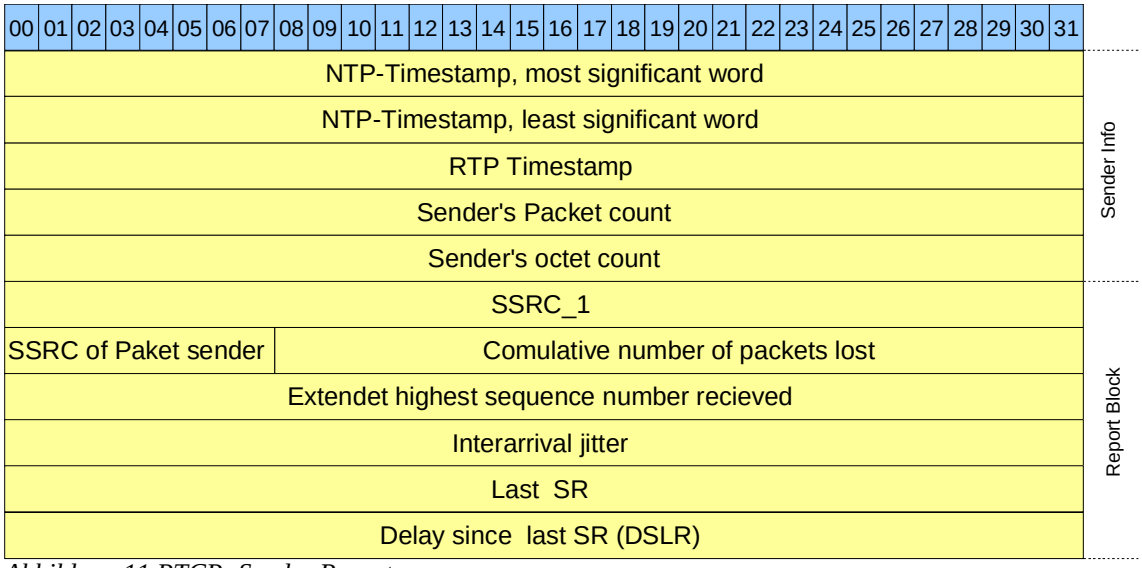


Abbildung 11 RTCP- Sender Report

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-Octets 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-Paketen, die seit dem letzten RTCP-Paket verloren gegangen sind.

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

Extd. Highest seq. Numer:
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.

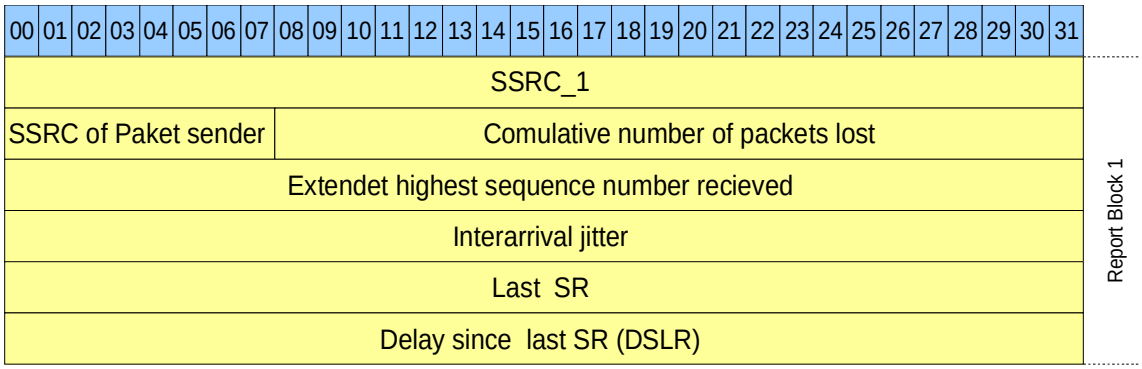


Abbildung 12 SRTCP- Receiver Report

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

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

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

Extd. Highest seq. Numer: 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.

6.2 Secure Real Time Protocol SRTP

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-Protocol (SRTP). Dieses Transportprotokoll bietet die gleichen Eigenschaften wie RTP, bringt aber Mechanismen zur Verschlüsselung, sowie einen Integritätsschutz mit sich. 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. Leider ist bei der Vovida Vocal Implementation SRTP noch nicht integriert.

6.2.1 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 13).

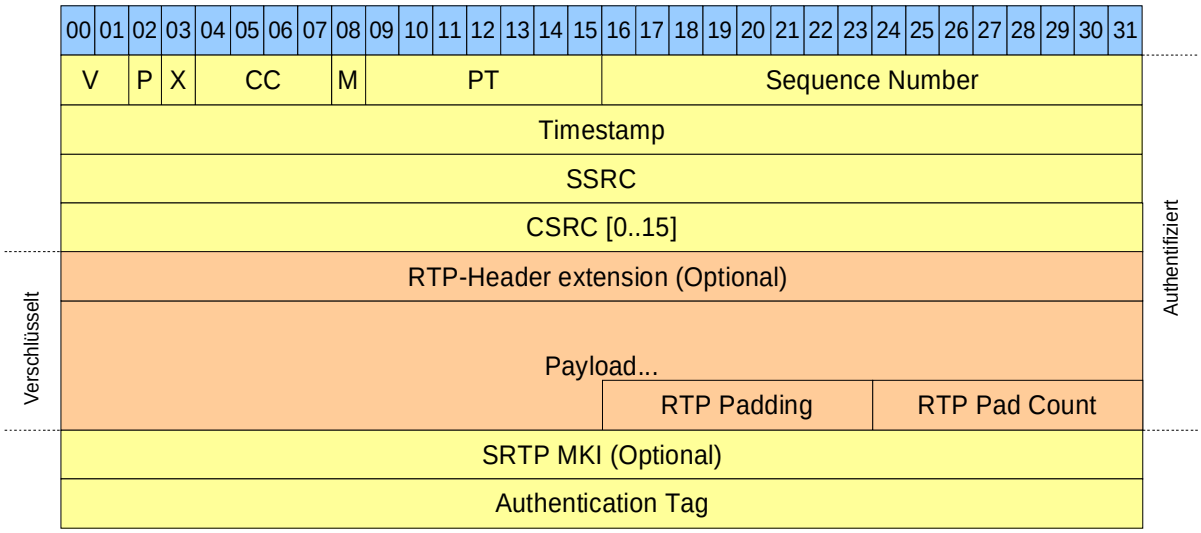


Abbildung 13 SRTP-Paketaufbau

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

Dem verschlüsselten Teil des Paketes folgt ein optionales Feld zur Identifizierung des Master-Keys und ein Feld für eine Paket-Signatur (Authentication Tag).

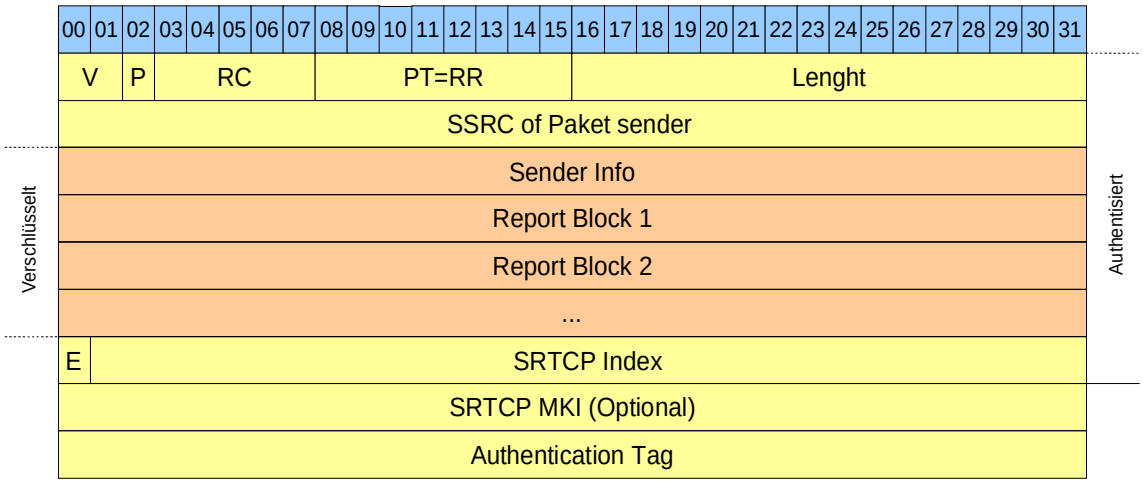


Abbildung 14 RTCP-Paket

6.2.2 SRTCP-Protokoll

Das Secure-Real-Time-Control-Protocol (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. Als Ergänzung ist das „Encryption Bit“ (Feld E in Abbildung 14) bei verschlüsseltem RTCP-Inhalt gesetzt. Weiter ist ein SRTCP Index eingefügt, ein Zähler der bei jedem SRTCP-Paket inkrementiert wird. Zusätzlich kann ein optionaler Key-Identifier, gefolgt vom obligatorischen Integritäts-Feld (Authentications-Tag) folgen.

6.2.3 SRTP-Sicherheitsmechanismen

Das SRTP-Protokoll benötigt einen einzelnen, symmetrischen Master-Key. Die Verteilung und die Erneuerung dieses Master-Keys ist nicht spezifiziert [SRTP]. Eine Möglichkeit zur Verteilung dieses Master-Keys ist MIKEY, beschrieben in Kapitel 6.2.4.

Schlüsselbildung

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].

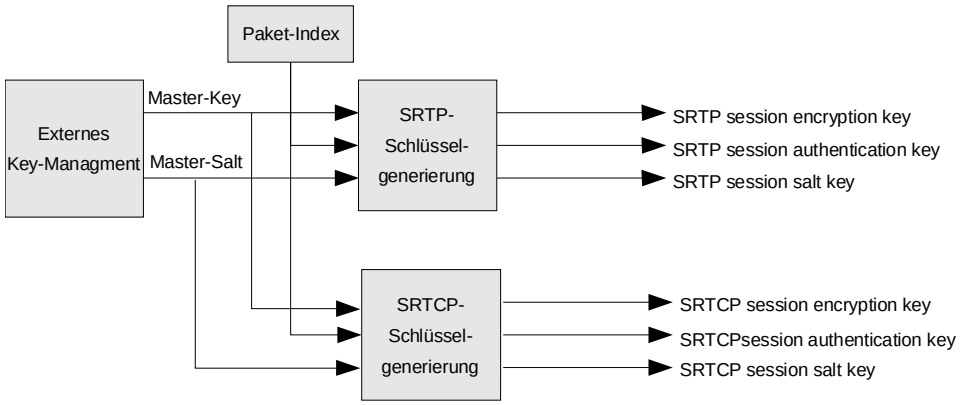


Abbildung 15 Schlüsselbildung

Verschlüsselung

Die Verschlüsselung erfolgt nun mit dem schnellen und modernen Advanced Encryption Standard (AES) im Counter-(AES-CTR) 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-Schlüssel gewählt werden, wobei dann die Daten unverändert durch den Verschlüsselungsalgorithmus gelangen 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 eine Signatur (Digest) der Daten mittels SHA1 gebildet. Diese Signatur ist aber immer noch im „Klartext“ d.h. eine unverschlüsselte Nachricht könnte von einem Angreifer verändert und die Signatur, mit dem bekannten Signaturverfahren neu signiert werden. Um die gefälschte Signatur zu entdecken, muss der Benutzer die korrekte Signatur bereits kennen. Um dies zu umgehen wird die Signatur noch per HMAC-Verfahren gesichert. Zum verschlüsseln der Signatur wird der Session-Authentcation-Key verwendet. Als Resultat bleibt eine 32-Bit Signatur, die nun in das Authentication-Tag des SRTP, bzw. SRTCP-Protokolls eingefügt wird.

6.2.4 MIKEY

Wie erwähnt sieht SRTP kein Protokoll für den Austausch des Master-Secret vor. In der Spezifikation (Draft) wird jedoch auf MIKEY verwiesen – ein weiteres Draft. Die MIKEY Spezifikation setzt sich aus drei Drafts zusammen, die jeweils einen Teil des Schlüsselaustauschs regeln. In Abbildung 5 ist eine Übersicht dieser Drafts zusammengestellt. Vermutlich haben sich die Nummern unterdessen schon geändert oder die Drafts wurden bereits als RFC freigegeben.

MIKEY Draft		Inhalt
draft-ietf-msec-mikey-06	[MIKEY]	MIKEY: Multimedia Internet KEYing
draft-ietf-mmusic-sdescriptions-00	[MMSD]	SDP Security Descriptions for Media Streams
draft-ietf-mmusic-kmgmt-ext-07	[KMGMT]	Key Management Extensions for SDP and RTSP

Tabelle 5 Übersicht der MIKEY Spezifikation

MIKEY bietet eine ähnliche Funktionalität wie IKE bei IPSec. MIKEY unterscheidet sich von IKE dadurch, dass es speziell für Multimedia-Sessions optimiert wurde. Das Protokoll wurde so einfach wie möglich gehalten und es wurde darauf geachtet, dass es effizient implementiert werden kann.

MIKEY kann über SDP getunnelt werden, so dass es sich hervorragend in das Gespann SIP/SDP/SRTP einfügt.

Der Key-Austausch kann entweder über ein Preshared-Secret[i], über eine Public-Key Verfahren oder Diffie-Hellman (DH) erfolgen.

7 IPsec

In der Spezifikation wird die Möglichkeit, IPsec anzuwenden nicht erwähnt. Doch es lohnt sich die Betrachtung, ob sich IPsec für das Absichern von VoIP eignet. Durch die folgenden Faktoren wird die Anwendung begrenzt:

- Für die Absicherung von SIP muss jeder Teil der Kommunikations-Strecke IPsec einsetzen: Es handelt sich wie bei SSL/TLS um eine Hop-by-Hop Lösung.
- Alle beteiligten Komponenten müssen über einen IPsec Stack verfügen, was insbesondere bei den Clienten – den IP Phones – nur in wenigen Fällen gegeben ist.

Während mittels einer Road-warrior-Infrastruktur die SIP-Signalisierung zum Proxy recht einfach in den Griff zu kriegen ist (Abbildung 16), kann der Verbindungsaufbau für den Medienkanal nicht so simpel gelöst werden: Zwei UAs mit wechselnder Adresse, die zuvor noch nie in Verbindung standen, müssen einen IPsec-Tunnel aufbauen. Wenn die beiden UAs nicht innerhalb der gleichen Organisation liegen, unterliegen sie vermutlich jeweils unabhängigen Zertifikat-Hierarchien.

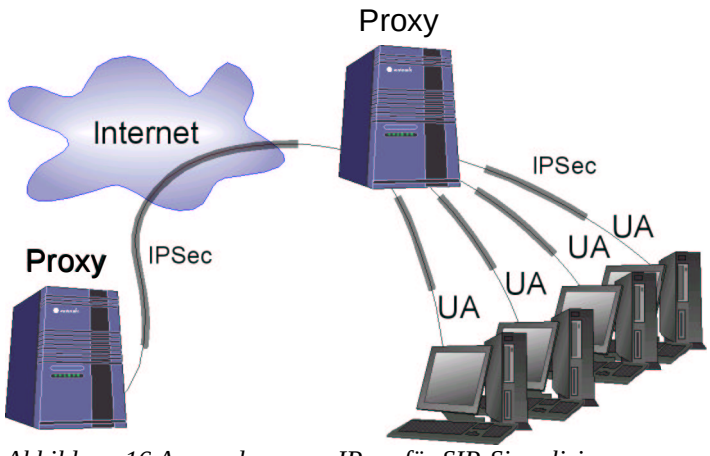


Abbildung 16 Anwendung von IPsec für SIP-Signalisierung

IPsec eignet sich gut für das Zusammenkoppeln von Organisations-Abteilungen über ein öffentliches Netz (Abbildung 16). Hier wird für jede Organisations-Abteilung ein Proxy vorgesehen, die jeweils einen IPsec-Tunnel zu den anderen Abteilungen aufbauen.

i Vorher ausgetauschtes Geheimniss: ein zuvor abgemachtes Passwort

Wenn der Medienkanal über IPsec getunnelt wird, und gleichzeitig QoS eingesetzt werden soll, wird die Konfiguration schwieriger. Auf dieses Problem werden wir hier nicht weiter eingehen.

Unsere Versuche mit IPsec erfolgten wie erwartet ohne Überraschungen. Die Verzögerungen nahmen, wenn überhaupt, nicht merkbar zu. Die Roadwarrior-Konfiguration mit dem Proxy als Gateway und den Clients als Roadwarrior funktionierte ohne Anpassung des SIP UA. Die Verschlüsselung des Medienkanals war schwieriger: Wir beschränkten uns auf die statische Konfiguration des Kanals, da sonst eine Anpassung des SIP UA nötig gewesen wäre, welche versucht eine IPsec Verbindung zum Gegenüber aufzubauen, bevor der Medienkanal eingerichtet wird. Unsere Versuche erfolgten mit FreeS/WAN 2.0 [FSWAN].

8 Gegenüberstellung

Die im vorangehenden Kapitel vorgestellten Sicherheitsmechanismen haben alle unterschiedliche, spezifische Eigenschaften. Wir haben in Abbildung 6 die wichtigsten Eigenschaften all dieser Mechanismen zusammenfassend dargestellt.

	UA↔Proxy	UA↔UA	Proxy↔Proxy	Hop-by-Hop	Auth	Encryption	SIP-Feature
HTTP-Digest	✓	✓	✗	✗	✓	✗	✓
SSL/TLS	✓	✓ ✗ *	✓	✓	✓	✓	✓
S/MIME	✗	✓	✗	✗	✓	✓	✓
SRTP	✗	✓	✗	✗	✗	✓	✗
MIKEY	–	✓	–	✗	✓	✓	✗
IPsec	✓	✓	✓	✓ ✗ **	✓	✓	✗

Table 6 Gegenüberstellung der für SIP anwendbaren Sicherheitsmechanismen
✓ Vorhanden, ✗ Nicht Vorhanden, – Nicht bewertbar

* Nur bei direkter Verbindung
** Hop-by-Hop wenn die SIP-Signalisierung involviert ist. Beim Medienkanal wird direkt zwischen den UAs verschlüsselt.

9 Kombination von Sicherheitsmechanismen

Erste Priorität in der gesicherten SIP-Telefonie hat die Identifikation des UAs gegenüber dem Proxy. Dieser Sicherheitsmechanismus ist leicht auch in SIP-Phones (Hardware) zu implementieren. Bei Hardware SIP-Phones stellt sich die Frage, ob diese die gewünschten Sicherheitsmechanismen überhaupt unterstützen. Stehen mit einem SIP-Softphone mehr Rechenleistung und mehr Softwareressourcen zur Verfügung, können weitere Sicherheits-Mechanismen implementiert werden.

Im folgenden Abschnitt haben wir eine praxisnahe Zusammenstellung von Sicherheits-Mechanismen vorgenommen.

9.1 Zusammenstellung

9.1.1 HTTP-Digest

Verfahren

Mit HTTP-Digest ist eine Authentifikation aller beteiligen UAs möglich. Ein Integritätsschutz bzw. Verschlüsselung ist nicht vorgesehen. Im Wesentlichen werden zwei Betriebsmodi unterschieden. Die eine Variante ermöglicht es dem Proxy den UA eindeutig zu identifizieren, während die andere Variante die Authentisierung in die Gegenrichtung erlaubt.

Bewertung

Ein Einfaches Verfahren zu Authentifikation von UA und Proxy, das bei den meisten SIP-Implementationen integriert sein sollte. HTTP-Digest sollte auch in Hardware SIP-Phones zum Einsatz kommen. Zur sicheren Kommunikation über ein unsicheres Netz sind weitere Sicherheitsmechanismen notwendig. Ein möglicher Anwendungsfall sehen wir in einem Privat- oder Firmennetz, in dem die interne Telefonie über SIP abgewickelt wird, aber der Zugang von aussen in dieses Netz anderweitig abgesichert ist.

9.1.2 SSL/TLS

Verfahren

Bei dieser Variante können sich UA und Proxy gegenseitig über Zertifikate identifizieren und gleichzeitig wird die Kommunikation (SIP) verschlüsselt. Eine Verschlüsselung des Medien-kanals muss bei Bedarf mittels weiteren Mechanismen vorgenommen werden.

Bewertung

Dieses Verfahren bringt den Vorteil, die Teilnehmer mit Hilfe einer Public-Key Infrastruktur (ohne Passwort) zu identifizieren. Weiter wird die SIP-Signalisierung Hop-by-Hop gesichert. Nachteilig wirkt sich aus, dass beim Hop-by-Hop Verfahren jedem beteiligten Knoten vertraut werden muss und es möglich, ist mit einem komprimitierten Knoten die ganze Sicherheit zu gefährdet. SSL/TLS erlaubt keine gegenseitige Identifikation der beiden beteiligten UAs. Die Sicherung des Medienkanals wurde hier nicht betrachtet.

9.1.3 S/MIME mit SRTP

Verfahren

Für den gesicherten Verbindungsaufbau wird S/MIME verwendet. Der Transport erfolgt über SRTP. Der Master-Key der SRTP Verbindung kann durch SDP wie in Kapitel 5.4 beschrieben übertragen werden. Dieses SDP-Paket wird durch S/MIME vor den Augen Dritter geschützt.

Bewertung

Eine sehr praxisnahe Lösung, bei der das Problem des Schlüsselaustausches einfach gelöst wird. Sowohl der Signalisierungs- als auch der Medienkanal werden bei dieser Lösung gesichert. Der Mechanismus funktioniert unabhängig von den beteiligten Proxys, es ist sogar eine sichere Kommunikation über unsichere Proxys möglich. Mit dem Schlüsselaustausch ist aber der ganze Verbindungsaufbau unflexibel.

9.1.4 SRTP mit MIKEY

Verfahren

Diese Kombination regelt die Sicherheit des Medienkanals. Zum Verbindungsaufbau (SIP) kann ein beliebiger, anderer Sicherheitsmechanismus eingesetzt werden. MIKEY übernimmt den Schlüsselaustausch für die gesicherte SRTP Verbindung.

Bewertung

Eine gute und flexible Möglichkeit um Schlüssel zu verteilen. Die Schlüssel können dabei von Punkt-zu-Punkt als auch von Punkt-zu-Multipunkt verteilt werden. SRTP kann nun mit den sicher übertragenen Schlüsseln eine verschlüsselte und integritätsgesicherte Medienverbindung herstellen. Das Problem des sicheren Verbindungsaufbau via SIP ist bei diesem Vorschlag noch nicht geklärt.

9.1.5 SIP über IPsec

Verfahren

Zum gesicherten Verbindungsaufbau wird ein statischer IPsec-Tunnel zum Proxy aufgebaut. Die Absicherung des Medienkanals wird hier nicht betrachtet.

Bewertung

Der Vorteil dieser Lösung besteht darin, dass die Signalisierung über ein sicherer, statischer Tunnel an den Proxy gelangt. Sowohl UA als auch Proxy können sich identifizieren und verschlüsselt kommunizieren. IPsec ist auch eine gute Möglichkeit um mehrere Proxys untereinander statisch über einen Tunnel zu verbinden. Es kann als Nachteil gewertet werden, dass die Kommunikation nur über vertrauenswürdige Proxys abgewickelt werden kann. Zudem hängt der Einsatz davon ab, ob alle SIP IP-Phones[i] über einen IPsec-Stack verfügen.

i Hardware SIP IP Phone

9.1.6 Medienkanal über IPsec

Verfahren

Bei dieser Variante wird der Medienkanal über ein IPsec Tunnel aufgebaut. Dadurch kann auf das SRTP-Transportprotokoll verzichtet und RTP eingesetzt werden. Die Signalisierung ist bei dieser Variante nicht abgesichert und könnte beispielsweise auch mit IPsec oder einem anderen Verfahren geschehen.

Bewertung

Wenn der Medienkanal über ein IPsec Tunnel geführt wird, kann als Transportprotokoll RTP statt SRTP verwendet werden. Damit müssen keine Schlüssel zur Verschlüsselung ausgetauscht werden. Es werden auch beide Endteilnehmer (UA) unabhängig von der Signalisierung identifiziert. Der Nachteil an diesem Verfahren ist, dass der IPsec-Tunnel bei jeder Verbindungsaufnahme dynamisch aufgebaut werden muss. Dazu ist es notwendig, dass ein SIP-UA ein IPsec-Stack integriert hat, oder den Aufbau an eine weitere Applikation delegieren kann.

9.2 Fazit

Es stehen für SIP mehrere mögliche Sicherheitsmechanismen zur Verfügung. Die einen sind sicherer, die anderen dafür weit verbreitet und einfach einzusetzen. Grundsätzlich gibt es keinen besten Mechanismus. Für jede Anwendung muss die Frage nach Aufwand und Schutz individuell gestellt werden.

Für einen typischen Fall von IP-Telefonie über ein unsicheres Netz scheint uns eine kombinierte Lösung von S/MIME und SRTP als günstig. Dieses Verfahren bietet Authentifizierung, Integrationsschutz und Verschlüsselung des Signalisations- und Medienkanals. Der Aufwand für diese Sicherung hält sich in vertretbarem Rahmen und der Benutzer wird nicht eingeschränkt. Nachteilig ist an dieser Lösung nur, dass häufig weder S/MIME noch SRTP integriert ist. Eine entsprechende Implementation dieser Sicherheitsmechanismen ist wünschenswert.

10 Zusammenfassung

Einführung

Dass Internet-zentrierte Session-Initiation-Protocol (SIP) löst immer mehr das Telefonie-zentrierte H.323 Voice-over-IP Protokoll ab. SIP ist ein Signalisierungs-Protokoll für VoIP, welches den Verbindungsaufbau und den geordneten Verbindungsabbau regelt. Die eigentlichen Sprachinformationen werden über das Real-Time-Protokoll (RTP) transportiert. Bei Telefongesprächen über das öffentliche Internet sollte jedoch auch der Sicherheit genügend Beachtung geschenkt werden.

Ziel

In dieser Projektarbeit sollen Sicherheitsmechanismen für das Session-Initiation-Protocol (SIP) ausgearbeitet, nach Möglichkeit praktisch aufgebaut und getestet werden. Weiter sollen auch Sicherheitslösungen für das Transportprotokoll RTP erarbeitet werden.

Vorgehen

Nach dem gründlichen Studium der Grundlagen von SIP sowie RTP wurde eine kleine VoIP Infrastruktur aufgebaut. Dieser Versuchsaufbau basierte auf dem Produkt Vocal der Firma Vovida. Diese SIP-Implementation ist eine OpenSource Lösung für Linux. Das Versuchsnetz besteht aus zwei Softphones (UA) und einem Proxy der als Zentrale fungiert. Bei dieser Arbeit sind auch nützliche Installationsscrips entstanden, die eine weitere Installation vereinfachen.

Zuerst wurde der in Vocal integrierte Sicherheitsmechanismus HTTP-Digest aktiviert und getestet. Dieser Mechanismuss bietet eine Authentifizierung des Teilnehmers gegenüber dem Proxy und dem Gesprächspartner. Eine weitere Verschlüsselung ist nicht vorgesehen.

Ein Sicherheitsmechanismus auf Basis von PGP fiel schon sehr früh aus dem Rennen, da PGP von SIP nicht mehr unterstützt wird.

Ein Sicherheitsmechanismus der von SIP unterstützt wird ist S/MIME. Bei dieser Lösung ist eine Authentisierung und Verschlüsselung, sogar eine Integritätssicherung des SIP-Protokolls möglich. Durch das tunneln von SIP über SIP mit S/MIME ist eine sichere Kommunikation über unsichere Proxys möglich. Dieser Sicherheitsmechanismus konnte, mangels Implementation, nicht in Betrieb genommen und praktisch geprüft werden.

Die Hop-by-Hop Sicherheit, die eine SSL/TLS Lösung für SIP bieten würde, konnte ebenfalls mangels fehlender Implementation nicht in der Praxis getestet werden. Dieser Sicherheitsmechanismus bietet eine Authentisierung der Gesprächsteilnehmer und kann einen möglichen Schlüssel, zum verschlüsseln des Medienkanals, sicher austauschen. Diese Lösung bringt aber den Nachteil mit sich, dass allen beteiligten Knoten vertraut werden muss.

Eine weitere Methode zur Absicherung von SIP ist IPsec. Diese Methode wurde in unserem Versuchsnetz praktisch aufgebaut und getestet. IPsec bietet einerseits die Möglichkeit ein statisches Tunnel zum Proxy aufzubauen und den gesamten SIP-Verkehr dadurch zu sichern, sowie der dynamische Aufbau eines Tunnels, um den Medienkanal abzusichern.

Während dieser Arbeit haben wir uns auch mit dem Transportprotokoll auseinendergesetzt und haben einen Ersatz von RTP durch das Secure-Real-Time-Protokoll geprüft. Dieses Protokoll erscheint uns als geeignet um Sprachdaten sicher unter den Gesprächsteilnehmern zu transportieren. Praktische Erfahrungen mit Sprachübermittlung konnen allerdings nicht gesammelt werden.

Resultate

Das Resultat dieser Arbeit ist eine Zusammenstallung von Sicherheitsmechanismen, die zum Teil für den Vovida SIP-Stack erhältlich sind, aber grösstenteils noch nicht für SIP implementiert worden sind.

Diese Sicherheitsmechanismen wurden bewertet und unsere Empfehlung abgegeben.

11 Schlusswort

Das Resultat dieser Projektarbeit ist eine Sammlung von Sicherheitslösungen für SIP basierende VoIP-Systeme, die teilweise von uns praktisch erprobt werden konnten. Doch die wenigsten dieser Lösungen sind bereits in den erhältlichen Implementationen enthalten, obschon genug geeigente Lösungen vorhanden sind.

In dieser Projektarbeit konnte der Bogen zwischen der Kommunikationstechnik und der sicheren Netzwerkommunikation geschlagen werden. Wir hatten die Möglichkeit eine VoIP-Infrastruktur von Grund auf kennen zu lernen und in Betrieb zu nehmen. Diese Arbeit gab uns auch die Möglichkeit, uns intensiv mit verschiedenen Sicherheitsmechanismen auseinander zu setzten, in einer Form, die weit über den regulären Unterricht hinausgeht.

Leider konnten nur wenige der betrachteten Sicherheitsmechanismen auch tatsächlich aufgebaut und praktisch getestet werden. Die Zeit war zu knapp bemessen, um selber eine dieser Funktionen zu implementieren. Schliesslich handelt es sich um ein komplexes Zusammenspiel vieler, einzelner und – für sich selber gesehen – einfacher Protokolle.

Während den letzten Wochen arbeiteten wir Seitenweise RFCs, zwei Bücher und diverse Dokumentationen durch. Die Herausforderung bestand darin, das Wesentliche in diese Dokumentation zu packen und verständlich zu erklären.

Anhang A: Aufgabenstellung

Projektarbeiten im SS 2003 - PA2 Sna02

SIP Security

Studierende:

- Daniel Kaufmann, IT3
- Andreas Stricker, IT3

Termine:

- Ausgabe: Dienstag, 20.05.2003 im E523
- Abgabe: Freitag, 4.07.2003

Beschreibung:

Dass Internet-zentrierte Session Initiation Protocol (SIP) löst immer mehr das Telefonie-zentrierte H.323 Voice-over-IP Protokoll ab. Bei Telefongesprächen über das öffentliche Internet sollte jedoch auch der Sicherheit genügend Beachtung geschenkt werden.
Im Rahmen der ausgeschriebenen Projektarbeit soll eine Übersicht über die für SIP vorgesehenen Sicherheitsmechanismen (Verschlüsselung und Authentisierung) erarbeitet werden. Geeignete Methoden sollen praktisch aufgebaut und erprobt werden.

Aufgaben:

- Studium der in RFC 3261, Abschnitt 26, vorgeschlagenen Sicherheitsansätze:
  - Layer 7: HTTP Digest, S/MIME, PGP
  - Layer 4: TLS auf Layer 4
  - Layer 3: IPsec auf Layer 3
- Studium des Secure Real-time Transport Protokolls (SRTP)
- Implementation eines oder mehrerer Sicherheitsansätze z.Bsp. auf der Basis des OpenSource SIP Stacks und User Clients von Vovida.
- Bewertung der Sicherheitsansätze

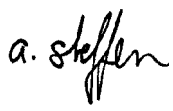
Infrastruktur / Tools:

- Raum: **E523**
- Rechner: 2 PCs mit Windows/Linux
- SW-Tools: Linux OpenSource Software

Literatur / Links:

- Henning Schulzrinne, SIP Security
  www.softarmor.com/sipwg/meets/ietf52/slides/pres-schulzrinne-SIP-Security-ietf52.ppt
- 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-06.txt>
  The Secure Real-time Transport Protocol
- Vovida SIP Implementation
  http://www.vovida.org
- OpenSource Library implementing SRTP
  http://srtp.sourceforge.net/spec.html

Winterthur, 20. Mai 2003



Prof. Dr. Andreas Steffen

Anhang B: Projektplan und Aufgabenteilung

Projektplan

Soll

Task	Kalenderwoche						
	21	22	23	24	25	26	27
Einarbeitung Theorie							
Einarbeitung System							
Grundsystem Client							
Grundsystem Proxy							
Aufbau HTTP-Digest							
Aufbau S/MIME. PGP							
Aufbau SSL/TLS							
Aufbau Ipsec							
Aufbau SRTP							
Unvorhergesehenes							
Auswertung, Bewertung							
Dokumentation							

Ist

Task	Kalenderwoche						
	21	22	23	24	25	26	27
Einarbeitung Theorie							
Einarbeitung System							
Grundsystem Client							
Grundsystem Proxy							
Aufbau HTTP-Digest							
Aufbau S/MIME. PGP							
Aufbau SSL/TLS							
Aufbau Ipsec							
Aufbau SRTP							
Auswertung, Bewertung							

Aufgabenteilung

Task	Wer
Einarbeiten SIP	Beide
Einarbeiten RTP/SRTP	Beide
Einarbeiten Vovida Vocal	Beide
Installation Linux Clients (RedHat / Knoppix)	Dani
Aufsetzten Server Debian	Andy
Installation Vocal- Server	Andy
Installation Vocal- Client -(Server) RedHat	Dani
Installation Client Knoppix	Andy
Konfiguration / Inbetriebnahme SIP-Infrastruktur	Beide
Inbetriebnahme HTTP-Digest	Andy
Ausarbeitung Sicherheitslösungen	Beide
Installation IPsec Server	Andy
Installation IPsec Clients	Dani
Konfiguration / Inbetriebnahme IPsec	Beide
Dokumentation	Beide

Anhang C: Installationsscript

```
#!/bin/bash
# Make Vovida Vocal on systems without X

# change follow lines if needed
MAKELOG='./make.log'
VOCALDIR='./vocal'
TARGETS='contrib
heartbeat
sip
sdp2
pslib
cdrlib
openssl
fs
rs
hbs
ps
ms
cdr
b2bua
vme
vmcp
vmserver
mail
sipset
contrib_imap
base64encoder
radius_test
staging_notar
java'

# do not edit lines below
CURDIR=$(pwd)
SCRIPTNAME="$0"
WITHOPENSSL='false'
WITHJAVA='false'
LOGFILE="${CURDIR}${VOCALDIR}"
ERR_COMP_MSG="configure failed: All header files and \
compiler tools installed?"

function logger() {
    if [ "x$1" = "xt" ]; then
       echo "+ $2 successful compilated" >> ${CURDIR}/${MAKELOG}
    else
       echo "- $2 failed compilation" >> ${CURDIR}/${MAKELOG}
    fi
}

function error() {
    echo "ERROR: $1" 1>&2
    echo "ERROR: $1" >> ${CURDIR}/${MAKELOG}
    exit -1
}

while [ "x$1" != "x" ]; do
    case $1 in
        --with-openssl)
            WITHOPENSSL='true'
        ;;
        --with-java)
            WITHJAVA='true'
        ;;
```

```
        --vocal-dir)
            VOCALDIR="$2"
            shift
        ;;
        *)
            echo "
Usage: $SCRIPTNAME [--with-openssl] [--with-java]
                   [--vocal-dir DIR]
        --with-openssl       compile with openssl support
        --with-java          compile with javaadministration
                              interface (needs sdk)
        --vocal-dir DIR      specify directory to change into
                              (default DIR=vocal)
"
            exit -1
        ;;
    esac
    shift
done

LOGFILE="${CURDIR}${VOCALDIR}"

if [ "x$WITHOPENSSL" != "xtrue" ]; then
    TARGETS=$(echo -en "$TARGETS" | grep -v "openssl")
fi

if [ "x$WITHJAVA" != "xtrue" ]; then
    TARGETS=$(echo -en "$TARGETS" | grep -v "java")
fi

echo "Logfile from Vocal compilation" > ${LOGFILE}
echo "Successful compiled targets" >> ${LOGFILE}

cd ${CURDIR}/${VOCALDIR} \
|| error "Can''t cd to $VOCALDIR, exiting"

if [ "x$WITHOPENSSL" == "xtrue" ]; then
    ./configure --with-openssl || error "${ERR_COMP_MSG}"
    echo "+ Configure successfull with openssl" >> ${LOGFILE}
else
    ./configure || error "${ERR_COMP_MSG}"
    echo "+ Configure successfull without openssl" >> ${LOGFILE}
fi

for target in $TARGETS; do
    make $target && logger t $target || logger f $target
done

cd $CURDIR
cat $MAKELOG
echo "Done, check $MAKELOG for further informations"

exit 0
```

Anhang C: Übersicht der OpenSource Software für SIP

Wir setzten vorwiegend das Vovida Vocal-System ein. Dennoch blickten wir über den Tellerrand und schauten uns Alternativen an.

Kommerzielle Systeme gibt es eine ganze Fülle: Die Hersteller wissen wie sie diese verkaufen müssen und so findet sich schnell ein entsprechendes Produkt. Da es bei diesen Produkten mehrheitlich schwer ist, den Quelltext zu erhalten und auch entsprechende Lizenzgebüren fällig werden, konzentrierten wir uns auf die OpenSource Lösungen, welche unseren Anforderungen durchwegs gerecht wurden.

Nun folgt ein kurzer Überblick von OpenSource SIP-Software-Projekten.

Vovida

SIP-Stack	Der Stack basiert noch auf RFC 2543, auch wenn einige Features von RFC 3261 hinzugefügt worden sind. Der Stack läuft sehr stabil und wurde auf Performance optimiert. Er ist komplett in C++ geschrieben, was sich vor allem bei der Übersetzung äussert: Viel Platzbedarf für die Objectfiles und lange Compilier-Zeit.
Vocal	Das Vocal System besteht aus diversen Server, unter anderem: Marshall MS (Proxy), Register-Server RS, Provisioning Server PS (Verwaltung), ein MGCP und ein H.323 Gateway.
SIPset und gua	Simpler UA, eher gedacht für Testzwecke.

Alle Vovida Produkte stehen unter der BSD-ähnlichen Vovida Software Lizenz.

reSIProcate

Bei reSIProcate handelt es sich ursprünglich um den Vovida SIP-Stack, der in grossen Teilen neu geschrieben wurde, mit dem Ziel den neuen Standard nach RFC 3261 vollständig zu unterstützen. Dieses Projekt bietet auch einen kleinen Clienten an, der Meldungen über SIP verschicken kann. Der C++ Quelltext wurde optimiert, was sich insbesondere bei der Übersetzung durch weniger Platzbedarf auszeichnet.

Auch diese Stack steht unter der Vovida Software Lizenz. Deshalb existieren auch kommerzielle Produkte, welche diesen Stack verwenden.

oSIP

Dieser unter der GNU Public Lizenz (GPL) stehende Stack unterstützt ebenfalls den neuen Standard (RFC 3261). Der Stack wurde in C geschrieben und hängt nur von der libc ab. Er ist klein und daher auch für embedded Systeme geeignet. Für oSIP gibt es einen kleinen Test-Clienten namens Josua, der nur die Signalisierung beherrscht.

osr

osr bietet die Funktionalität eines Registrars und eines Proxys.

SIP Phones

Aufbauend auf den oben genannten Stacks sind mindestens vier ernst zu nehmende Softphones vorhanden. Die Unterstützung von verschiedenen Codecs und der Funktions-Umfang variieren.

Übersicht

SIP Stacks:

Projekt	Referenz
ReSIProcate	http://www.resiprocate.org/, http://sourceforge.net/projects/resiprocate/
libSIP BSD-Project	http://sourceforge.net/projects/libsip/
osr - open source SIP registrar/redirect proxy	http://osip.atosc.org/osr.html
osip	http://www.fsf.org/software/osip

Sip Phones:

Projekt	Referenz
Kphone	http://www.div8.net/kphone
SIPHON	http://siphon.sourceforge.net
Sippo	http://sippo.hotsip.com/download/index.html
linphone	http://www.linphone.org

Anhang D: Inhalt CD-ROM

/README.TXT	Beschreibung des Inhalts und Ergänzungen zu dieser Inhaltsangabe
/docu	Dokumentation, PDF, PS und OpenOffice
/docu/HOWTO	Installations-Howto
/config	Einige Beispiel-Konfigurations-Dateien
/software	Verwendete Software-Pakete
/software/vocal	Vovida Vocal System (Proxy, Registration Server, SIPSet, und mehr)
/software/reSIProcate	SIP RFC3261 Stack
/software/SRTP	SRTP Stack
/software/freeswan	FreeS/WAN 2.0
/rfc/	Die von uns angeschauten RFCs und Drafts

Beachten Sie die README.TXT im Hauptverzeichnis der CD-ROM. Diese enthält Ergänzungen zu dieser Inhaltsangabe.

Literatur

Quellenverzeichnis

[RFC3261] SIP: Session Initiation Protocol, Rosenberg, et al., IETF 2002, www.ietf.org/rfc/rfc2543.txt

[RFC3263] Session Initiation Protocol (SIP): Locating SIP Servers, Rosenber & Schulzrinne, IETF 2002, www.ietf.org/rfc/rfc3263.txt

[RFC3264] An Offer/Answer Model with the Session Description Protocol (SDP), Rosenberg & Schulzrinne, IETF 2002, www.ietf.org/rfc/rfc3264.txt

[RFC2976] The SIP INFO Method, Donovan, IETF 2000, www.ietf.org/rfc/rfc2976.txt

[RFC2327] SDP: Session Description Protocol, Handley & Jacobson, IETF 1998, www.ietf.org/rfc/rfc2327.txt

[RFC2616] Hypertext Transfer Protocol -- HTTP/1.1, Fielding, et al., IETF 1999, www.ietf.org/rfc/rfc2616.txt

[VoIPGW] Voice over IP Gateway - ISDN2SIP Gateway, Ernst Till & Gallizzi Ulrich, 2002

[VOVIDA] Vocal Installation Guide, , Vovida 2003, http://www.vovida.org/document/

[RFC2543] SIP: Session Initiation Protocol, Handley, et al., IETF 1999, www.ietf.org/rfc/rfc2543.txt

[RFC2069] An Extension to HTTP: Digest Access Authentication, , IETF 1997, www.ietf.org/rfc/rfc2069.txt

[RFC2617] HTTP Authentication: Basic and Digest Access Authentication, Franks, et al., IETF 1999, www.ietf.org/rfc/rfc2617.txt

[RFC2633] S/MIME Version 3 Message Specification, Ramsdell, IETF 1999, www.ietf.org/rfc/rfc2633.txt

[RFC2246] The TLS Protocol Version 1.0, Dierks & Allen, IETF 1999, www.ietf.org/rfc/rfc2246.txt

[RFC1889] A Transport Protocol for Real-Time Applications, Schulzrinne, et al., IETF 1996, www.ietf.org/rfc/rfc1889.txt

[SRTP] The Secure Real Time Protocol, Baugher, et al., 2003, www.ietf.org/internet-drafts/

[FIPS-197] Announcing the ADVANCED ENCRYPTION STANDARD (AES), Federal Information Processing Standards, 2001,

[MIKEY] MIKEY: Multimedia Internet KEYing, Arkko, et al., 2003,

[MMSD] SDP Security Descriptions for Media Streams, Baugher & Wing, 2003,

[KMGMT] Key Management Extensions for SDP and RTSP, Arkko, et al., 2003,

[FSWAN] FreeS/WAN Project, John Gilmore, et al., FreeS/WAN 2003,

Abkürzungen

MiM	Man-in-the-Middle
QoS	Quality of Service
RTP	Realtime Transport Protocol
RTSP	Realt Time Streaming Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SRTP	Secure Realtime Transport Protocol
S/MIME	Secure/Multipurpose Internet Mail Extensions
UA	User Agent: SIP IP Telefon oder Softphone
VoIP	Voice over IP, Telefonie über IP-Netze

Glossar

Replay Attack
: Attacke durch wiederholen eines Paketes. Kann durch Seriennummern verhindert werden.

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).

Downgrade Attack
: Der Angreifer versucht die Gesprächspartner zu einem unsicheren oder keinem Verschlüsselungs-Algorithmus zu zwingen.

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

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.

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.