IT-Security

Instant Messaging und Presence Security

Analyse von Maßnahmen für sichere und anonyme Kommunikation

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science in Engineering (BSc)

eingereicht von

Christoph Leo Mahrl

is121018

im Rahmen des

Studienganges IT-Security an der Fachhochschule St. Pölten

Betreuung

Betreuer/in: Dipl. Ing. Daniel Haslinger,BSc

Mitwirkung: -

St. Pölten, 17. Mai 2015 _____ _____

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

*

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Bachelorarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.

- ich dieses Bachelorarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

- diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

Der Studierende räumt der FH St. Pölten das Recht ein, die Bachelorarbeit für Lehre- und Forschungstätigkeiten zu verwenden und damit zu werben (z.B. bei der Projektevernissage, in Publikationen, auf der Homepage), wobei der Absolvent als Urheber zu nennen ist. Jegliche kommerzielle Verwertung/Nutzung bedarf einer weiteren Vereinbarung zwischen dem Studierenden/Absolventen und der FH St. Pölten.

St. Pölten, 17. Mai 2015

(Unterschrift Verfasser/in)

Danksagung

Hiermit möchte ich mich bei allen Personen herzlich bedanken, die mich während der Anfertigung dieser wissenschaftlichen Arbeit unterstützt haben.

Mein besonderer Dank gilt:

- Dipl. Ing. Daniel Haslinger für die angenehme Betreuung
- meinen Korrekturlesern Philipp Schneider und meiner Lebensgefährtin Nicole Moser
- meiner Mutter Anneliese Mahrl und ihren Lebensgefährten Franz Schabschneider für die stetige Unterstützung während meines Studiums

St. Pölten, 17. Mai 2015

Christoph Leo Mahrl

Kurzfassung

Täglich werden weltweit Milliarden von Nachrichten mit Instant Messaging Clients unter Personen ausgetauscht. Vor allem Mobile-Messenger wie WhatsApp erfreuen sich großer Beliebtheit unter der breiten Masse. Neben Textnachrichten werden auch vielfach Bild-, Video- und Sprachaufnahmen versendet. Diese anfallenden Daten stellen nur einen kleinen Prozentanteil des enormen globalen Informationsflusses dar.

Die Enthüllungen globaler Spionageprogramme staatlicher Organisationen erregten internationales Aufsehen, zumal diese Organisationen sich eigenmächtig Zugriff auf eine enorme Menge an sensiblen Daten beschaffen konnten. Dies bewegte viele Menschen zu einem verstärkten Sicherheitsbewusstsein und die Nachfrage nach verschlüsselter Kommunikation nahm zu.

Diese wissenschaftliche Arbeit beschäftigt sich mit den Gefahren, die mit Instant Messaging einhergehen und evaluiert Maßnahmen, um sicher und anonym über das Internet zu kommunizieren. Dazu werden verschiedene Technologien vorgestellt und auf ihre Praxistauglichkeit überprüft. Weiters wird eine Übersicht über aktuelle Instant Messaging Clients erstellt, welche die Sicherheit von Nachrichten und die Anonymität des/der Benutzers/Benutzerin wahren.

Im praktischen Teil dieser Arbeit wird ein eigenständiger Secure-Messenger entwickelt, welcher auf einem dezentralen Peer-to-Peer Modell basiert und mittels Onion Routing anonyme Kommunikation ermöglicht.

Abstract

Worldwide every day people exchange billions of messages using instant messaging clients. Especially mobile clients like WhatsApp enjoy great popularity among the masses. Adjacent to text messages people also send pictures, videos and voice recordings. These data represent only a small percentage of the immense global flow of information.

The revelations of global espionage programs of governmental organizations created international stir since those organizations have had access to an enormous amount of sensitive data. This incident led to increased security awareness of many people. and a rising demand for encrypted communication.

This bachelor thesis deals with the dangers that go along with instant messaging and evaluates measures to communicate securely and anonymously over the Internet. To do this, different technologies will be presented and checked for their suitability. Furthermore a general overview of current secure instant messaging clients will be given.

In the practical part of this thesis a prototype of a secure messenger is developed from the ground up. This messenger is based on a decentralized peer-to-peer network and facilitates anonymous communication by using onion routing.

Inhaltsverzeichnis

1 Einleitung

Im Rahmen dieser Bachelorarbeit werden verschiedene Technologien für sichere und anonyme Kommunikation von Instant Messengern im Internet erläutert. Im einleitenden Teil dieser Arbeit wird die derzeitige Situation beleuchtet und auf die Gefahren, die mit unsicherer Kommunikation einhergehen, näher eingegangen. Die forschungsleitenden Fragestellungen, als auch die Beschreibung des Aufbaus dieser Arbeit, bilden den Abschluss dieses Kapitels.

1.1 Motivation für das Thema

Die Schnelllebigkeit und das große Wachstum des Internets, als auch neuartiger Technologien, verschaffen Internetbenutzern und -benutzerinnen nicht nur Vorteile, sondern bergen auch unbekannte und unvorhergesehene Gefahren.
Im Juni 2013 veröffentlichten die amerikanische Zeitung *The Washington Post*, beziehungsweise die in Großbritannien ansässige Tageszeitung *The Guardian*, geheime NSA[1]-Dokumente. Diese brachten ein umfassendes und globales Überwachungsprogramm namens *PRISM* ans Tageslicht.[1]
Den Stein ins Rollen gebracht hat dabei der ehemalige NSA-Mitarbeiter Edward Snowden, der sich für die Enthüllungen rund um den Skandal verantwortlich zeichnet und seither auf der Flucht vor der US-amerikanischen Justiz ist. Seit diesem Zeitpunkt werden fast täglich neue Enthüllungen der Öffentlichkeit bekannt gemacht.[1][2]
Die Aufdeckung des PRISM Überwachungsprogramms hat gezeigt, dass die NSA Zugriff auf Unmengen an Daten vieler weltweiter Unternehmen hatte. Microsoft, Google, Yahoo, Facebook und Apple sind nur einige Beispiele von Unternehmen, die von permanenter Überwachung betroffen waren. Sensible Daten, die durch PRISM unautorisiert abgegriffen wurden, umfassten neben E-Mails und Chat-Nachrichten auch Videos, Fotos, VoIP Daten, Dateiübertragungen und vieles mehr.[2]
Die Enthüllungen dieses globalen Spionageprogramms als auch die technologischen Maßnahmen, die zur Prävention ergriffen werden hätten können, waren ausschlaggebende Faktoren, um über dieses Thema zu schreiben.

[1] National Security Agency

1.2 Problemstellung

Wie wichtig maßgebliche, technologische Schutzmaßnahmen für sichere Kommunikation im Internet sind, zeigen aktuelle Statistiken und Prognosen zum Thema Instant Messaging.
Einer Statistik zufolge wurden im Jahr 2012 weltweit 14,7 Billionen Instant Messaging Nachrichten versendet. Juniper Research prognostizierte Anfang 2014, dass sich bis 2017 die Anzahl gesendeter Nachrichten auf 28.2 Bill. beinahe verdoppeln wird.[3]
Weitere Statistiken zeigen, dass täglich weltweit 600 Millionen Fotos, 200 Millionen Sprachnachrichten und 100 Millionen Videonachrichten alleine über den Instant Messenger *WhatsApp* verschickt werden.[4]
Wenn nun jemand Zugriff auf all diese Daten erhält, ist eine zielgerichtete Erstellung von Profilen einzelner Personen möglich. Eventuell besteht durch einen solchen Zugriff auch die Möglichkeit sehr sensible Daten wie zum Beispiel politische oder religiöse Ansichten auf einzelne Personen rückzuschließen.
In Anbetracht der täglich anfallenden Daten geben daher unter anderem die Überwachungsprogramme staatlicher Organisationen Grund zur Sorge, wie Informationen sicher und anonym untereinander ausgetauscht werden können.

1.3 Forschungsleitende Fragestellungen

An diese Problemstellung stellen sich daher folgende grundlegende forschungsleitende Fragestellungen an:

- Welche Angriffsvektoren und Defensivmaßnahmen sind bei Instant Messaging für sichere Kommunikation im Internet zu berücksichtigen?
- Wie sieht die praktische Umsetzung der Schnittstellendefinition von Tor in einer Secure-Messenger Implementierung aus?

1.4 Aufbau dieser Arbeit

Die folgende Auflistung gibt einen kurzen Überblick über den Inhalt der anschließenden Kapitel:

- **Kapitel 2 - Instant Messaging und Presence**
  In diesem Kapitel wird Instant Messaging allgemein erläutert und geklärt, was unter Presence verstanden wird. Darüber hinaus werden verschiedene technologische Architekturen, als auch aktuelle Instant Messaging Transportprotokolle vorgestellt.

- **Kapitel 3 - Security und Privacy**

  In Kapitel „Security und Privacy“ werden diverse Gefahrenquellen, als auch reelle Angriffsszenarien vorgestellt, die mit Instant Messaging und Presence einhergehen. Zudem werden verschiedene Sicherheitsmetriken und technologische Maßnahmen zur Mitigation derartiger Risiken beschrieben.

- **Kapitel 4 - Tor-basierte Instant Messaging Clients**

  Dieses Kapitel stellt verschiedene auf dem Markt erhältliche Instant Messaging Clients vor, die dediziert das Tor Netzwerk als Transportmittel für Sofortnachrichten und Präsenzinformationen verwenden.

- **Kapitel 5 - DChat - Eigenentwicklung eines IM Clients**

  In diesem Kapitel wird ein Prototyp eines Instant Messaging Clients entwickelt, welcher ausschließlich als Hidden Service über das Tor Netzwerk erreichbar ist. Es werden dabei verschiedene Voraussetzungen als auch technische Schnittstellen evaluiert, um die Entwicklung eines solchen Prototyps zu ermöglichen. Im Anschluss daran wird erläutert, inwiefern sich DChat von den in Kapitel 4 vorgestellten Clients differenziert.

2 Instant Messaging und Presence

„*I would rather be without a state than without a voice.*“ , Edward Snowden.

In diesem Kapitel werden verschiedene Termini rund um Instant Messaging und Presence erläutert. Anschließend wird kurz auf die geschichtliche Entwicklung von Instant Messaging eingegangen und es werden sowohl technologische Architekturen, als auch verschiedene aktuelle Transportprotokolle vorgestellt.

2.1 Terminologie

Die Internet Engineering Task Force definiert Instant Messaging und Presence im RFC[1] 2778 wie folgt:

> „*A presence and instant messaging system allows users to subscribe to each other and be notified of changes in state, and for users to send each other short instant messages.*“.[5, S. 1]

Aus dieser Definition heben sich zwei wichtige Eigenschaften hervor. Zum einen ist das die Presence (*deutsch:* Präsenz) Information, die zu erkennen gibt, ob und welche Benutzer/innen gerade verfügbar sind. Zum anderen ist das die Eigenschaft, dass Nachrichten zwischen Benutzern und Benutzerinnen in beinahe Echtzeit ausgetauscht werden. Aus der Möglichkeit „Instant Messages“ (*deutsch:* Sofortnachrichten) untereinander auszutauschen, leitet sich daher der Name Instant Messaging (IM) ab.[6, S. 2] Tabelle 2.1 gibt eine kurze Übersicht über Eigenschaften und Funktionen, die oft mit Instant Messaging in Verbindung gebracht werden[7, S. 71]:

Online-User	Ein/e auf einem IM Server erfolgreich angemeldeter/angemeldete Benutzer/in.
Sofortnachricht	Eine von einem/einer Benutzer/in verfasste Nachricht, die ein/e Empfänger/in erhält, sobald dieser/diese am IM Server angemeldet ist.

[1] Request for Comments

Präsenz	Informiert darüber, ob ein/e bestimmter/bestimmte Benutzer/in online, beziehungsweise offline ist.
Verfügbarkeit	Eine von einem/einer Benutzer/in festgelegte Statusinformation, die seine/ihre Verfügbarkeit angibt (z.B. “away“, “do not disturb“, etc.).
Kontaktliste	Eine Liste von Benutzern/Benutzerinnen, deren Präsenz- und Verfügbarkeitsinformationen abonniert sind.
Block-Liste	Diese Liste enthält Benutzer/innen, die explizit blockiert sind. Von Benutzern/-Benutzerinnen dieser Liste werden keine Sofortnachrichten, Präsenz- und Verfügbarkeitsinformationen mehr empfangen.
Allow-Liste	Im Gegensatz zur Block-Liste ist der Empfang von Sofortnachrichten, Präsenz- und Verfügbarkeitsinformationen von Benutzern/Benutzerinnen dieser Liste erlaubt. Diese ist bei den meisten IM Clients deckungsgleich mit der Kontaktliste
One-to-One Chat	Eine Unterhaltung zwischen zwei Benutzern/Benutzerinnen, deren Nachrichten mit Hilfe eines oder mehreren IM Servern transportiert wird.
Gruppenchat	Eine virtuelle Gruppe, bestehend aus mehreren Benutzern/Benutzerinnen, die untereinander eine Unterhaltung führen.
Chatraum	Ein virtueller Raum mit einem bestimmten Thema, in welchem Benutzer/innen Nachrichten untereinander austauschen können.

Tabelle 2.1: Instant Messaging Begriffe[7, S. 71]

Instant Messaging bezeichnet somit eine Kommunikationsform, in welcher zwei oder mehrere Personen involviert sind, die Nachrichten synchron untereinander austauschen. Diese Form der Nachrichtenübermittlung unterscheidet sich maßgeblich von asynchronen Formen, wie zum Beispiel Email. Denn im Gegensatz zum synchronen Nachrichtentransport, kann ein/e Verfasser/in einer Nachricht bei asynchroner Kommunikation nicht mit einer unmittelbaren Antwort des/der Empfängers/Empfängerin rechnen. [8, S. 135]

2.2 Geschichtliche Entwicklung

Instant Messaging hat seinen Ursprung in den 1980er Jahren. Zu dieser Zeit waren Nachrichtensysteme noch sehr einfache und rudimentär implementierte One-to-One Chat Applikationen. Mit der Entwicklung von *talk*, *ytalk* und *ntalk* fanden seinerseits die ersten Nachrichtendienste in die UNIX-Welt ihren Einzug.[8, S. 135]

Zu Beginn war es ausschließlich möglich Nachrichten mit Personen auf demselben Computer auszutauschen und jeder Tastenanschlag wurde unmittelbar versendet. Mit der Zeit änderte sich dies, sodass jeweils nur vollständig verfasste Nachrichten verschickt wurden. Diese Nachrichten konnten später auch an Personen auf anderen Computern versendet werden.[8, S. 135]

Über die Jahre hinweg entwickelten sich One-to-One Nachrichtendienste weiter, sodass sich bis Ende der 1990er Jahre derartige Applikationen an großer Beliebtheit erfreuten. Verbreitete Anwendungen[8, S. 135f.] auf dem IM Markt waren damals ICQ, AIM, Yahoo! Messenger, MSN Messenger und Google Talk. Diese Applikationen wiesen bereits typische Instant Messaging Eigenschaften auf, wie es heute üblich und verbreitet ist. AIM und MSN Messenger hatten beispielsweise virtuelle Gruppen, VoIP[2] und Videoübertragung als Features implementiert.[8, S. 136]

In den letzten Jahren drang Instant Messaging immer mehr in den Bereich Webapplikationen und Mobile Messaging vor, sodass diese für viele Personen kaum von ihrem Alltag wegzudenken sind. Abbildung 2.1 veranschaulicht die zehn weltweit meist genutzten Instant Messenger im Jahr 2013 (in Millionen).[9]

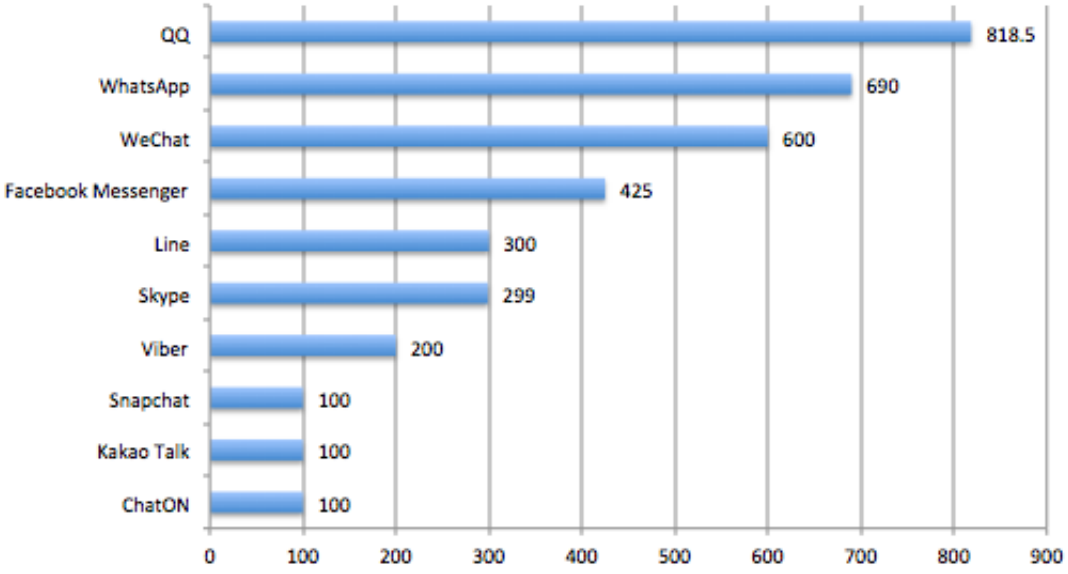


Abbildung 2.1: Top 10 Instant Messenger[9]

Dieser Statistik kann entnommen werden, dass *WhatsApp*, *WeChat* und *Facebook Messenger* mit 425 Millionen bis 690 Millionen Benutzer/innen die beliebtesten Mobile-Messenger sind. Im Bereich Desktop-Anwendungen führen dagegen die Instant Messenger *Skype*, *Viber* und *QQ*, das vor allem im asiatischen Raum äußerst beliebt ist, mit Nutzerzahlen zwischen 200 und 818,5 Millionen aktiven Nutzern/Nutzerinnen weltweit.[9]

[2] Voice over IP

2.3 Technologische Modelle

RFC 2778 definiert ein abstraktes Modell, das die notwendigen Komponenten zur Implementierung eines Instant Messaging Systems beschreibt. Dieses Modell legt zwei grundlegende Dienste fest[5, S. 2]:

- **Präsenzdienst**

  Dieser Dienst ist dafür zuständig, Präsenzinformationen anzunehmen, zu speichern und gegebenenfalls zu verteilen. Es existieren zwei verschiedene Typen von Clients, die vom Präsenzdienst Gebrauch machen. Einerseits sind das die sogenannten *Presentities*, die dem Dienst Präsenzinformationen übermitteln. Andererseits sind das die sogenannten *Watchers*, die sich Präsenzinformationen von diesem Dienst beschaffen.[5, S. 2f.] Die Gruppe der *Watchers* unterteilt sich wiederum in[5, S. 3]:

  - **Fetcher**

    Ein Client vom Typ *Fetcher* fordert lediglich aktuelle Präsenzinformationen von einer bestimmten *Presentity* vom Präsenzdienst an. Ein spezielle Ausprägung dieses Typs ist der sogenannte *Poller*, welcher regelmäßig um diese Informationen anfragt.

  - **Subscriber**

    Im Gegensatz zum *Fetcher* verlangt ein *Subscriber* Benachrichtigungen über zukünftige Änderungen der Präsenzinformationen einer bestimmten *Presentity* vom Präsenzdienst.

- **Sofortnachrichtendienst**

  Die Annahme und Verteilung von Sofortnachrichten erledigt dieser Dienst. Der Sofortnachrichtendienst unterscheidet ebenfalls zwei verschiedene Client-Typen. Dabei handelt es sich einerseits um *Sender*, die Sofortnachrichten zur Verteilung zur Verfügung stellen und andererseits um *Instant Inboxes*, an welche Nachrichten zugestellt werden. Letztere fungieren somit als Empfänger.[5, S. 4f.]

In den nächsten Abschnitten werden die Optionen beschrieben, wie diese beiden Dienste in Form eines IM Servers und die dazugehörigen Clients architektonisch aufgebaut sein können. Anschließend werden verschiedene aktuelle Transportprotokolle erläutert, die zwischen Präsenzdienst, Sofortnachrichtendienst und den jeweiligen Clients interagieren.

2.3.1 Client-Server Modell

Die einfachste aller Architekturen ist das häufig anzutreffende Client-Server Modell. Bei diesem Modell gibt es eine wichtige Kontrollinstanz, den *Server*. Der *Server* dient als Service Provider und bietet Dienste an, die von Clients in Anspruch genommen werden können.[7, S. 71]
Im Falle von Instant Messaging dient der *Server* als Service Provider für Präsenz- und Sofortnachrichtendienste.[7, S. 71] Abbildung 2.2 und Abbildung 2.3 veranschaulichen den Kommunikationsweg zwischen den beiden Clients, *Client A* und *Client B*. Wenn beide Clients auf dem IM Server angemeldet sind, können diese unter Verwendung der Dienste des IM Servers untereinander Nachrichten austauschen. Die jeweiligen Clients sind dabei direkt mit dem IM Server verbunden und die Kommunikation mit dem jeweiligen anderen Client verläuft daher indirekt.[7, S. 71]
Es besteht die Möglichkeit, dass mehrere IM Server im Hintergrund einen Cluster bilden, nach außen hin den Clients jedoch als eine Entität gegenüber treten (siehe Abbildung 2.3). Für einen Client ist dies unbemerkbar, da dieser die erstellte Nachricht an einen IM Server übermittelt. Die Nachricht durchläuft den Cluster und wird dabei von Server zu Server weitergereicht. Schließlich stellt ein IM Server die Nachricht dem/der entsprechenden Empfänger/in zu.[7, S. 71])

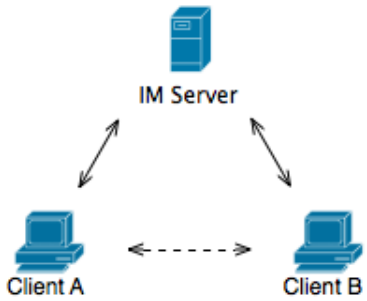


Abbildung 2.2: Einfach-Server Modell[7, S. 71]

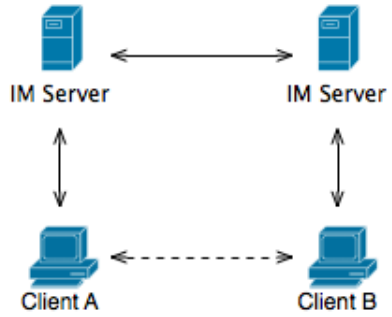


Abbildung 2.3: Mehrfach-Server Modell[7, S. 71]

2.3.2 Peer-to-Peer Modell

Im Gegensatz zum Client-Server Modell existieren beim Peer-to-Peer Modell keine speziellen Kontrollinstanzen. Es besteht daher keine Abhängigkeit zu Servern, die Nachrichten zwischen Clients vermitteln. Die Architektur dieses Modells ist auf völlige Dezentralität ausgelegt. Alle Clients sind auf einer Ebene gleichgestellt und kommunizieren direkt miteinander. Daher rührt auch der Name dieses

Modells, *Peer-to-Peer* (*deutsch:* Gleichgestellter zu Gleichgestellter). Abbildung 2.4 illustriert ein solches Netzwerkmodell.[10, S. 2][11, S. 6]

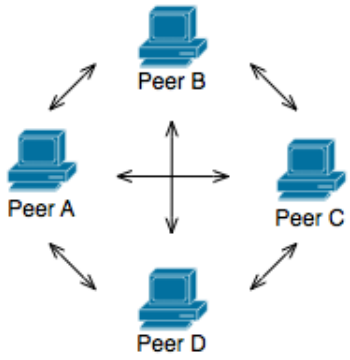


Abbildung 2.4: Peer-to-Peer Modell[10, S. 2]

In Kapitel 5 wird ein Nachrichtensystem inklusive Transportprotokoll entwickelt, das auf dem Peer-to-Peer Modell und infolgedessen auf einem dezentralisierten Netzwerkmodell basiert.

2.3.3 Hybrides / Super Peer-to-Peer Modell

Das Hybride-Peer-to-Peer Modell bzw. dessen Weiterentwicklung, das Super-Peer-to-Peer Modell, basieren auf einem semi-zentralen Netzwerk. Ein Netzwerk dieser Art ist eine Mischung aus Client-Server Modell und Peer-to-Peer Modell (siehe Abbildung 2.5).[10, S. 2][11, S. 6]

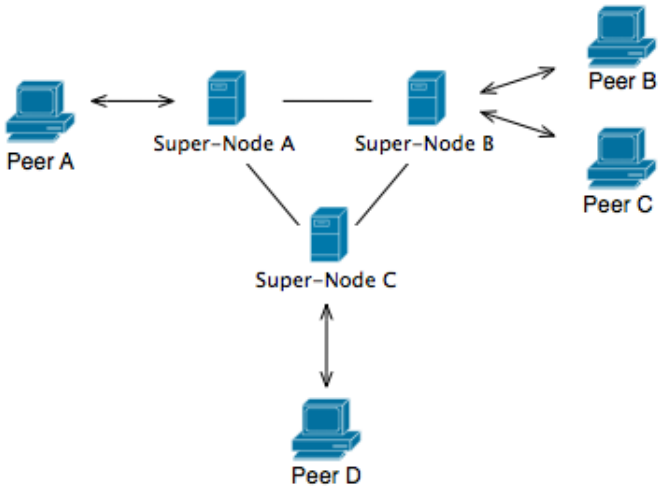


Abbildung 2.5: Hybrides Peer-to-Peer Modell[10, S. 2]

Es besteht aus zwei unterschiedlichen Typen von Netzwerkknoten. Zentrale Knotenpunkte, auch Super-Nodes genannt, übernehmen mit Präsenz- und Sofortnachrichtendienst die Serverrolle. Diese Super-

Nodes sind direkt zueinander verbunden und bilden durch ihre hierarchische Struktur ein klassisches Peer-to-Peer Netzwerk. Die übrigen Knoten, hier als Peers bezeichnet, interagieren mit den Super-Nodes in einer gewöhnlichen Client-Server Beziehung (siehe Abbildung 2.5).[10, S. 2][11, S. 6]

2.4 Überblick über aktuelle Instant Messaging Protokolle

Nachfolgend werden einige aktuelle Netzwerkprotokolle erläutert, die für die Interaktion zwischen Server und Clients bzw. Peers benötigt werden. Sie transportieren Informationen, wie Präsenzdaten und Sofortnachrichten, und regeln die Interaktion zwischen Präsenzdienst, Presentities und Watchers als auch Sofortnachrichtendienst, Senders und Instant Inboxes.[5, S. 5]

2.4.1 XMPP

Das *Extensible Messaging and Presence Protocol* (*ehemals:* Jabber) ist ein seit Jahren ausgereiftes und etabliertes Protokoll. Jabber wurde 1998 von Jeremie Miller entwickelt und im Oktober 2004 im RFC 3920 und RFC 3921 als XMPP standardisiert.[12, S. 7f.]
XMPP basiert auf einem dezentralisierten Client-Server Netzwerkmodell (siehe Kapitel 2.3.1), welches der Netzwerkstruktur für den Transport von E-Mails ähnelt. Der Grundgedanke für dieses Modell war die Trennung von Verantwortlichkeiten. Entwickler/innen von Client-Anwendungen sollten den Fokus auf User-Experience legen, wohingegen Entwickler/innen von Server-Anwendungen ihr Hauptaugenmerk auf Stabilität und Zuverlässigkeit legen sollten.[12, S. 11]
Im Gegensatz zur E-Mail, die möglicherweise mehrere Hops (E-Mail Server) durchläuft, wird eine XMPP Nachricht vom XMPP Server eines Clients direkt an den XMPP Server des/der Empfängers/Empfängerin übermittelt. Dieses Netzwerkmodell wird auch als *Direct Federation* Modell bezeichnet.[12, S. 13]
Um kommunizieren zu können, benötigt jede XMPP Entität eine eindeutige ID. Diese wird JabberID (JID) genannt und entspricht dem Format von E-Mail Adressen, z.B. *christoph@jabber.org*. Vorteil dieser Vorgehensweise ist, dass das Format auf einer existierenden DNS[3]-Infrastruktur und dem dazugehörigen Namensraum aufbaut. Die virtuelle Identitäten verwaltet ein XMPP Server. Die XMPP Netzwerkinfrastruktur basiert hauptsächlich auf öffentliche XMPP Server (z.B. *jabber.org*). Auf diesen öffentlichen Servern können jederzeit neue virtuelle Identitäten registriert werden. Es ist jedoch auch möglich, selbst

[3] Domain Name System

einen privaten XMPP Service zu betreuen, dessen JabberIDs den bei der Server-Installation angegebenen Domänennamen verwenden. Eine Liste von öffentlichen XMPP Server findet man auf folgender Website: `https://xmpp.net/directory.php`.[12, S. 13]

Zur eindeutigen Unterscheidung von unterschiedlichen Ressourcen desselben/derselben Benutzers/Benutzerin, wie Smartphone, Notebook, etc. wird der JID beim Verbindungsaufbau zu einem XMPP Server eine *Resource-ID* hinzugefügt. Die JID hat dann folgendes Format: *christoph@jabber.org/mobile*, wobei */mobile* den *Resource-Identifier* angibt.[12, S. 14f.]

Nachrichten und Präsenzinformationen werden in einem fortlaufenden TCP[4]-Stream als XML[5]-Snippets transportiert. Dieser Stream wird beim Verbindungsaufbau zu einem XMPP Server geöffnet und transportiert über ihre gesamte Laufzeit die folgenden drei XML-Snippets (auch *Stanza* genannt)[12, S. 16-20]:

- **<message/>**

  Dieses XMPP Stanza wird verwendet, um Nachrichten/Informationen zu übermitteln. Es findet Anwendung bei Instant Messaging, Gruppenchats und Benachrichtigungen.[12, S. 18]

- **<presence/>**

  Dieses Snippet dient zur Übertragung von Präsenzinformationen. Der Transport dieser Informationen wird dabei nach dem *Publish-Subscribe* Modell gehandhabt (siehe Kapitel 2.3). Das bedeutet, dass bei Änderungen der Präsenzinformationen einer XMPP-Entität dessen Kontakte über die Änderung benachrichtigt werden.[12, S. 19f.]

- **<iq/>**

  Das *Info/Query* XMPP-Snippet dient für Request/Response Interaktionen. Mit den Attributen *get* und *set* ähnelt es den Übertragungsmethoden von HTTP[6]. Im Gegensatz zum *Message*-Stanza erwartet das *IQ*-Stanza immer eine Antwort, sodass ein zuverlässiger Transport von Daten gewährleistet ist. Dieses Snippet findet hauptsächlich Verwendung bei Video-, Audio- und Dateiübertagungen.[12, S. 20f.]

2.4.2 MTProto

MTProto ist noch ein sehr junges Protokoll, welches seinen Einsatz im Telegram Messenger findet und im Jahr 2013 von Pavel und Nikolai Durov entwickelt wurde. Das Protokoll kommt in einer Client-Server

[4] Transmission Control Protocol
[5] Extensible Markup Language
[6] Hyper Text Transfer Protocol

Netzwerkstruktur zum Einsatz und ist in drei voneinander unabhängigen Komponenten unterteilt[13]:

1. **Transport**

   MTProto Nachrichten können über drei verschiedene Transportprotokolle vom Client zum Server und umgekehrt übermittelt werden[13]:

   - HTTP
   - TCP
   - UDP[7]

2. **API Query Language**

   Server und Client tauschen innerhalb einer Sitzung Nachrichten aus, wobei eine Sitzung jeweils an ein Client-Gerät gebunden ist. Dies wird durch eine *User-Key-ID* bewerkstelligt, welche auch für Autorisierungen verwendet wird. Nachrichten, die zwischen Server und Clients ausgetauscht werden, sind unter anderem[13]:

   - RPC[8]-Aufrufe (Client zu Server)
   - RPC-Antworten (Server zu Client)
   - Bestätigungen von erhaltenen Nachrichten
   - Status von ausstehenden Anfragen
   - Multipart Nachrichten bzw. Container, die mehrere Nachrichten beinhalten

3. **Cryptographic Layer**

   Bevor eine Nachricht über das Netzwerk vom Client zum Server bzw. vice versa transportiert wird, wird diese in einer bestimmten Art und Weise verschlüsselt. Der zu versendenden Nachricht wird ein Header am Anfang der Nachricht beigefügt, welcher jeweils eine 64-Bit lange User-Key-ID (für die eindeutige Identifizierung der Client bzw. Server Authorization-Keys) als auch einen 128-Bit langen Message-Key beinhaltet. Dies ergibt insgesamt einen 256-Bit langen Schlüssel. Die Nachricht wird schließlich mit diesem Schlüssel mittels AES[9]-256 verschlüsselt.[13]

[7] User Datagram Protocol
[8] Remote Procedure Call
[9] Advanced Encryption Standard

2.4.3 SIMPLE

Die IETF hat viele Spezifikationen bezüglich Erweiterungen des SIP[10][14]-Protokolls um Instant Messaging und Presence Funktionen veröffentlicht. *SIMPLE*, was für SIP for Instant Messaging and Presence Leveraging Extensions steht, ist eine Sammlung von Erweiterungen des für IP-Telefonie gedachten SIP-Protokolls.[14, S. 2]

Im Gegensatz zu den zuvor beschriebenen Protokollen unterstützt *SIMPLE*, neben typischen IM Eigenschaften, auch Sprach- und Videoübertragungen. Um Präsenzdienste zu ermöglichen und die Handhabung entsprechender Präsenzinformationen zu unterstützen, wurde die Präsenzspezifikation in folgende Punkte gegliedert[14, S. 3-8]:

- **Core Protocol Machinery**

  Dieser Teil definiert die eigentlichen SIP Erweiterungen im Bezug auf Subscriptions, Notifications und Publications. Präsenzinformationen werden nach dem Publish-Subscribe Modell gehandhabt (siehe Kapitel 2.3).[14, S. 4]

- **Presence Documents**

  Presence Documents sind XML-Dokumente, die Präsenzinformationen beinhalten und als solche in SIP Nachrichten transportiert werden. Ein/e Benutzer/in erhält diese Dokumente, sobald er/sie Präsenzinformationen einer SIP-Entität abonniert.[14, S. 5]

- **Privacy and Policy Documents**

  Diese XML-Dokumente beschreiben, ob und wie Präsenzinformationen einer SIP-Entität bei anderen SIP-Entitäten dargestellt werden sollen. SIMPLE ermöglicht es, dass jeder/jede Benutzer/in bestimmen kann, ob er/sie Subscriptions von seiner/ihrer Präsenz zulässt bzw. verhindert und welche Informationen konkret übertragen werden dürfen.[14, S. 6]

- **Provisioning**

  Die korrekte Funktionsweise von SIMPLE setzt voraus, dass gewisse Benutzerdaten entsprechend verwaltet und schließlich im System vorgehalten werden. Diese Daten inkludieren Kontaktlisten, Privacy/Policy Dokumente und vieles mehr. Die Verwaltung dieser Daten geschieht mit Hilfe von XCAP[11]. User Agents verwenden dieses Protokoll, um die auf dem Server gespeicherten XML-Dokumente zu manipulieren.[14, S. 2]

[10] Session Initiation Protocol
[11] XML Configuration Access Protocol

- **Optimizations**

  Der Austausch von Präsenzinformationen ist eine kostspielige Operation. Um die Performance von SIMPLE zu verbessern, wurden verschiedene Maßnahmen spezifiziert. Beispielsweise werden bei Notifications von Präsenzinformationen anstatt der vollständigen Informationen ausschließlich die Änderungen übertragen.[14, S. 8]

SIMPLE unterstützt neben Presence auch zwei Modi von Instant Messaging. Diese Modi gliedern sich in[14, S. 9f.]:

- **Page Mode**

  In diesem Modus werden Sofortnachrichten als SIP-Requests über den SIP-Server transportiert. Diese Requests beinhalten Nachrichteninhalte.[14, S. 10]

- **Session Mode**

  Im Session Mode wird IM, neben Video und Audio, als eigener Medientyp betrachtet. *INVITE* Requests bauen dabei direkt Sitzungen zu den gewünschten SIP-Entitäten auf, mit welchen man sich schließlich via IM unterhalten kann.[14, S. 10]

Wenn nur wenige Sofortnachrichten untereinander ausgetauscht werden, ist der Page Mode effizienter als der Session Mode. Falls jedoch längere Konversationen geführt werden, ist der Session Mode dem Page Mode bezüglich Performance überlegen. Dies hat den Grund, dass beim Session Mode die Nachrichten direkt an die entsprechende Entität gesendet werden. Der Umweg über den SIP-Server bleibt somit erspart.[14, S. 10]

2.4.4 RetroShare

RetroShare ist ein IM Protokoll, das auf ein dezentralisiertes, Peer-to-Peer basierendes Netzwerkmodell setzt. Im Gegensatz zu herkömmlichen Peer-to-Peer Anwendungen bzw. Protokollen, erlaubt *RetroShare* ausschließlich die Verbindung zu bekannten Kontakten. Das Peer-to-Peer Netzwerk einer RetroShare-Entität besteht somit einzig und allein aus Freund/inn/en, die man mehr oder weniger persönlich kennt. Diese Art von Netzwerk wird auch Friend-to-Friend (F2F) Netzwerk genannt.[15]
Features, die *RetroShare* unterstützt, sind:

- Private Konversationen
- Private und öffentliche Gruppenchats

- RetroShare Foreneinträge
- Filesharing
- VoIP

Um ein sicheres F2F Netzwerk zu etablieren, setzt RetroShare sehr stark auf PGP[12]. PGP ist eine asymmetrische Verschlüsselungstechnologie, die den Aufbau eines sogenannten *Web of Trust* Netzwerks ermöglicht. Ziel eines solchen Netzwerks ist es, mittels digitaler Signaturen Man-In-The-Middle Attacken zu verhindern. Neben Sofortnachrichten werden in *RetroShare* auch lokale Daten mit Hilfe von PGP-Schlüssel verschlüsselt. Diese umfassen beispielsweise Kontaktlisten, Forenposts, Nachrichtenhistorien, uvm. Zum besseren Verständnis wird PGP in Kapitel 3.4.1 näher erläutert.[16]
Ein zusätzliches Sicherheitsfeature von *RetroShare* bildet das anonyme Turtle Routing Modell. Dieses gewährt Personen Anonymität bei Dateiübertragungen über das dezentralisierte Netzwerk mit Hilfe von Tunneln. Der Routing Algorithmus, der in diesem Modell angewandt wird, erlaubt den Austausch von Daten mit „Non-Direct Friends“, also Kontakten, die einem nicht unmittelbar bekannt sind (z.B. Freunde/Freundinnen von Kontakten). Nähere Informationen zur Funktionsweise dieses Protokolls sind auf folgender Website zu finden: `http://www.turtle4privacy.org/`[17]

[12] Pretty Good Privacy

3 Security und Privacy

> „*Those who surrender freedom for security will not have, nor do they deserve, either one.*“ ,
> Benjamin Franklin.

In diesem Kapitel werden verschiedene Gefahrenquellen und Angriffsszenarien erläutert, die mit Instant Messaging und Presence einhergehen. Außerdem wird sowohl auf unterschiedliche Sicherheitsmetriken, als auch auf Technologien, wie PGP, OTR und Tor zur Prävention bzw. zur Mitigation derartiger Risiken näher eingegangen.

3.1 Sicherheitsaspekte und -metriken

Instant Messaging ist für potentielle Angreifer/innen eine sehr gute Quelle um sensible und schützenswerte Daten von anderen Personen zu erlangen. Mittels Malware, Social Engineering oder anderen durchdachten Methoden können umfangreiche Informationen über eine Person gewonnen werden. Solche Daten sind unter anderem[18, S. 2]:

- Nachrichten, die untereinander ausgetauscht werden
- Verbindungsdaten, die angeben mit welchen Personen kommuniziert wird
- Präsenzinformationen, die den Onlinestatus einer Person bestimmen
- Kontaktlisten, die Informationen zu bekannten Personen beinhalten
- vCards, die Daten zur eigenen Person (Name, Wohnort, E-Mail Adresse, etc.) speichern

Die Electronic Frontier Foundation hat zum Schutz sensibler Informationen eine Scorecard für Secure Messaging veröffentlicht. In dieser wurden verschiedene Sicherheitsmetriken aufgestellt und die bekanntesten Instant Messenger darauf überprüft und bewertet (Weitere Informationen unter: `https://www.eff.org/de/secure-messaging-scorecard`). In der folgenden Auflistung

werden die Sicherheitsmetriken der Scorecard näher betrachtet, wobei diese teilweise vom Autor dieser Arbeit modifiziert und ergänzt wurden [19]:

- **Ende-zu-Ende Verschlüsselung**

  Eine ausreichend sichere Ende-zu-Ende Verschlüsselung soll sicherstellen, dass Nachrichten vertraulich transportiert werden und ausschließlich Verfasser/innen und Empfänger/innen die übermittelten Nachrichten lesen können.

- **Authentizität und Bestreitbarkeit von Nachrichten**

  Ein/e Empfänger/in einer Nachricht soll verifizieren können, ob die Nachricht tatsächlich von dem/der vermeintlichen Verfasser/in stammt. Dennoch muss der Schutz des/der Verfassers/Verfasserin gegeben sein, sodass die Authentizität gegenüber einen Dritten nicht bewiesen werden kann.

- **Sicherheit vorhergehender Konversationen**

  Die Sicherheit von privaten Konversationen zwischen Personen soll auch bei Entwendung des geheimen Schlüssels durch einen/eine Angreifer/in gegeben sein (Stichwort: Perfect Forward Secrecy).

- **Dokumentation des Security Designs**

  Eine gute und aussagekräftige Dokumentation des Security Designs des Systems ist eine essentielle Voraussetzung, um das Konzept auf etwaige Sicherheitsmängel zu untersuchen. Die Dokumentation sollte eine detaillierte Beschreibung von angewandten Verfahren, Methoden und Algorithmen beinhalten.

- **Einsicht in Security Implementierungen**

  Neben der Dokumentation des Security Designs ist auch die Einsicht in entsprechende Implementierungen wesentlich. Unabhängige Dritte erhalten dadurch die Möglichkeit, einzelne Security-spezifische Implementierungen jederzeit auf etwaige Schwachstellen überprüfen zu können.

- **Security-Auditing**

  Die regelmäßige Durchführung von Security Audits durch Dritte sollen mögliche Vulnerabilities aufdecken. Die Audits umfassen sowohl die Analyse des Security Konzepts, als auch die genaue Überprüfung von Sourcecode-Ausschnitten entsprechender Security Implementierungen.

3.2 Gefahrenquellen

Jegliche internetfähige Applikationen stellen potentielle Risiken für Angriffe aus dem Internet auf das System dar. Klassische Gefahren, die bei Instant Messaging zu berücksichtigen sind, sind vor allem[20, S. 2][7, S. 73f.]:

- Malware (Wümer, Trojaner, Viren, etc.)
- Vulnerabilities in der Software (IM Client, IM Server, etc.)
- Offenlegung von Informationen, die nicht für die Öffentlichkeit bestimmt sind und
- Diebstahl von Passwörtern

In den folgenden Abschnitten wird auf die jeweiligen Gefahren näher eingegangen.

3.2.1 Malware

Mit Funktionen wie Filesharing ist Instant Messaging ein geeignetes Übertragungsmedium für schädliche Software. Um sich auf eine Vielzahl von Computern und anderen elektronischen Geräten zu verbreiten, werden bei Malware durchdachte Herangehensweisen angewandt. Würmer, die sich in IM Netzwerken verbreiten, greifen oft zu folgenden Methoden[20, S. 7f.][7, S. 74]:

1. **Verwendung von Hersteller APIs**[1]

   Mit Hilfe von legitimen und dokumentierten APIs von Herstellern, die für die Entwicklung eigener Clients bzw. Erweiterungen gedacht sind, können Malware Entwickler/innen sehr leicht eine schädliche, sich selbst verbreitende Software schreiben. Ein Wurm kann zum Beispiel benachrichtigt werden, wenn der Client eine Nachricht erhalten hat, oder wenn ein neuer Kontakt der Kontaktliste hinzugefügt wurde. Die APIs ermöglichen zudem, dass sich ein Wurm sehr einfach im Netzwerk verbreiten kann, indem er sich selbst an alle Kontakte schickt.[20, S. 7]

2. **Verwendung von APIs, um interaktiv Dateien zu versenden**

   Häufig öffnet sich beim Versenden von Dateien bei IM Clients ein Dialog, mit welchem der Dateitransfer bestätigt werden soll. Dies bedeutet, dass schädliche Software Eingaben des/der Benutzers/Benutzerin emulieren muss, um sich selbst zu verbreiten. Moderne Betriebssysteme bieten

[1] Application Programming Interface

sowohl zur Emulation von Benutzereingaben, als auch zur Enumeration von geöffneten Anwendungsfenstern, entsprechende APIs. Beim Versenden von Dateien verwenden Würmer diese APIs, um auf das Sendedialogfenster Zugriff zu erhalten und anschließend den Klick auf die „Senden"-Schaltfläche zu emulieren.[20, S. 7]

3. **Versenden von URL[2] Links**

   Eine noch einfachere Möglichkeit, Schadsoftware zu verbreiten, ist durch das Versenden von URL Links gegeben. Links werden vor allem dann versendet, wenn die Verwendung von APIs zu komplex ist, um Dateien interaktiv zu transferieren, oder die Hersteller APIs das Versenden von Dateien generell nicht anbieten. Durch den Klick auf einen solchen URL Link wird ein Download jener Schadsoftware gestartet, die schließlich das System kompromittiert.[20, S. 8]

4. **Patchen von Client DLLs**

   Eine fortgeschrittenere Methode zur Verbreitung von Malware ist das Patchen von Client DLLs[3]. Dabei kann es sich sowohl um spezielle Instant Messaging DLLs handeln, aber auch um Betriebssystem-spezifische DLLs wie *Winsock*. Mittels Modifizierung bestimmter DLLs kann erreicht werden, dass beispielsweise eine infizierte Datei beim Versenden einer Nachricht mitgesendet wird. Ein weiteres Beispiel ist, dass bei Dateiübertragungen anstatt der originalen Dateien schädliche Dateien verschickt werden.[20, S. 8]

Neben Würmern stellen auch Trojaner potentielle Gefahrenquellen in IM Netzwerken dar. Trojaner wenden wie Würmer ebenfalls die zuvor beschriebenen Methoden an, um sich auf beliebige Systeme einzuschleusen. Im Unterschied zu Würmern zielt ein Trojaner aber nicht darauf ab, sich möglichst oft zu verbreiten. Vielmehr ist es sein Ziel, ein Backdoor zum infizierten System zu schaffen. Über dieses Backdoor hat der/die Angreifer/in die Möglichkeit, den Trojaner von der Ferne aus zu steuern. Die Malware kann somit instruiert werden, sensible Informationen zu übermitteln oder willkürliche Kommandos auf dem infizierten System zu exekutieren.[20, S. 9]

3.2.2 Vulnerabilities in Software

Vulnerabilities (*deutsch:* Schwachstellen) in Applikationen, die aufgrund von Programmierfehlern entstehen, können schwerwiegende Konsequenzen mit sich bringen. Je nach Art der Vulnerability könnte

[2] Uniform Resource Locator
[3] Dynamic Link Libraries

ein/e Angreifer/in mit einem speziellen Exploit ein Denial of Service (DoS) erreichen oder im schlimmsten Fall sogar Zugriff auf das System erhalten, um beliebigen Code auszuführen. Typische Schwachstellen sind, aufgrund von fehlender Längenüberprüfung des Puffers, Bufferoverflows im Stack- oder Heapsegment einer Anwendung. [20, S. 10] Vulnerabilities in Instant Messaging Clients sind besonders gefährlich, da für den/die Angreifer/in nicht die Notwendigkeit besteht, nach gefährdeten Geräten im Internet zu suchen. Der/Die Angreifer/in kann davon ausgehen, dass die Schwachstelle auf allen Geräten ausgenutzt werden kann, auf welchen dieser IM Client installiert ist. Die Schadsoftware wird daher versuchen, sich über die Kontakte eines/einer Benutzers/Benutzerin zu verbreiten, anschließend über deren Kontakte, usw. Dies hat zur Folge, dass sich Malware innerhalb kürzester Zeit sehr schnell auf Millionen Geräten verbreiten kann.[20, S. 12]

3.2.3 Information Disclosure

Information Disclosure, also die ungewollte Veröffentlichung von privaten Informationen, stellt ein großes Risikopotential bei Instant Messaging dar. Das Risiko einer solchen Publikation ist deshalb erhöht, da Daten gestohlen werden könnten, ohne das System jemals mit Schadsoftware zu kompromittieren. Dazu kommen folgende Angriffsmethoden zum Einsatz[20, S. 14][7, S. 73f.]:

- **Session Hijacking**

  Nachdem ein/e Benutzer/in sich gegenüber dem IM System authentifiziert hat, stellt der IM Server diesem/dieser üblicherweise ein Session Cookie aus. Beim Aufruf von Serverfunktionen (z.B. zum Versenden von Sofortnachrichten) wird dieses Cookie mitgesendet. Der IM Server kann anschließend feststellen, ob die aktuelle Sitzung des/der Benutzers/Benutzerin noch gültig ist oder ob die Sitzung bereits abgelaufen und daher eine erneute Anmeldung erforderlich ist. Sollte es einem/einer Angreifer/in gelingen dieses Session Cookie zu stehlen, beispielsweise mit Hilfe eines Netzwerksniffers, kann er/sie das Session Cookie dazu missbrauchen, um die aktive Sitzung des Opfers zu kapern. Der/Die Angreifer/in kann sich dann als die betreffende Person ausgeben und so etwaige sensible Daten erbeuten oder anderweitigen Schaden anrichten. Diese Art des Angriffs wird als *Session Hijacking* bezeichnet.

- **Man-In-The-Middle (MITM) Attacken**

  Bei MITM-Attacken befindet sich ein/e Angreifer/in zwischen zwei oder mehreren Kommunikationsendpunkten und fungiert als Vermittler des Netzwerkverkehrs. Die Kommunikationsendpunk-

te, z.B. IM Server und IM Client, merken von dem/der Angreifer/in nichts und nehmen an, dass sie direkt mit dem jeweiligen anderen Kommunikationsendpunkt kommunizieren. Da die gesamte Kommunikation zwischen Server und Client über den/die Angreifer/in läuft, kann dieser/diese unbemerkt wertvolle Informationen aus dem Netzwerkverkehr erbeuten.

- **Data Sniffing / Data Export**

  Unverschlüsselter Netzwerkverkehr kann ebenfalls zu Datendiebstahl führen. Ein/e Angreifer/in benötigt lediglich ein Tool zur Analyse des Netzwerkverkehrs, um etwaige sensible Daten herauszufiltern. Extrahierte Daten können anschließend von einem/einer Angreifer/in über den IM Client exportiert werden. Dazu ist bloß ein schlichtes Backdoor auf dem System notwendig.

3.2.4 Passwortdiebstahl

Die meisten Instant Messaging Clients speichern das Passwort des/der Benutzers/in in einen Zwischencache, sodass beim Starten des Clients die Eingabeaufforderung nach Benutzernamen und Passwort erspart bleibt. Üblicherweise werden diese Passwörter obfuscated bzw. als Hashwert in der Registry oder in einer Datei im Filesystem gespeichert. Dies hat zur Folge, dass ein etwaiger Trojaner diese sensiblen Informationen an einen/eine Angreifer/in übermitteln könnte. Für den/die Angreifer/in ergibt sich schließlich die Möglichkeit, das Plaintext-Passwort zu ermitteln oder den erlangten Hashwert in einer Replay-Attacke zu verwenden. Die Gefahr des Passwortdiebstahls besteht nicht nur am lokalen System, sondern Passwörter könnten auch mit Hilfe eines Sniffers aus dem Netzwerkwerkverkehr abgegriffen werden. Dieses Risiko ist zum Beispiel dann gegeben, wenn das IM System den Authentifizierungsmechanismus fehlerhaft implementiert (Challenge-Response, Verschlüsselung, etc.), wodurch Angriffe wie Brute-Force-, Wörterbuch- oder andere Attacken möglich sind.[20, S. 16][7, S. 74]

3.3 Angriffsvektoren am Beispiel XMPP

Am Beispiel XMPP werden nachfolgend verschiedene Angriffsvektoren veranschaulicht, die zu Information Disclosure führen können. Zudem werden verschiedene Verschlüsselungsmöglichkeiten erläutert, um entsprechende Angriffsflächen einzudämmen. Die verschiedenen Verschlüsselungsmethoden werden am Ende dieses Kapitels näher vorgestellt.

3.3.1 Aufbau der Verbindung

In folgender Abbildung wird der Aufbau der Verbindung zweier miteinander kommunizierender XMPP-Entitäten gezeigt[18, S. 2]:

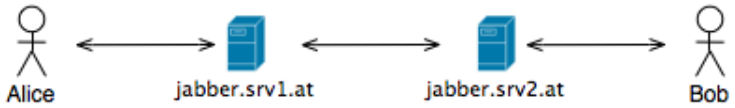


Abbildung 3.1: Verbindung zweier XMPP-Entitäten[18, S. 2]

Die XMPP-Server *jabber.srv1.at* und *jabber.srv2.at* sind die Homeserver von Alice (JID: *alice@jabber.srv1.at*) und Bob (JID: *bob@jabber.srv2.at*). Möchte Alice an Bob eine Nachricht versenden, so schickt sie ihre Nachricht an ihren Homeserver. Dieser übermittelt Alices Nachricht an Bobs Homeserver, der diese wiederum an Bob zustellt.

3.3.2 Relevante Sicherheitseigenschaften

Die nachfolgende Auflistung[18, S. 2f.] beschreibt Eigenschaften sicherer Kommunikation, die mit Hilfe kryptographischer Verschlüsselung gewährleistet werden können. In den anschließenden Szenarien wird überprüft, ob die jeweiligen Eigenschaften gegeben sind bzw. mit welchen Maßnahmen diese erreicht werden hätten können.[18, S. 2]

- **Vertraulichkeit**

  Nachrichten können nur von dem/der Verfasser/in bzw. dem/der Empfänger/in gelesen werden.

- **Integrität**

  Nachrichten können auf ihrem Transportweg zum/zur Empfänger/in nicht unbemerkt verändert werden.

- **Authentizität**

  Der/Die Empfänger/in einer Nachricht kann feststellen, ob die Nachricht tatsächlich von dem/der angegebenen Verfasser/in stammt.

- **Verbindlichkeit**

  Der/Die Verfasser/in einer Nachricht kann nicht dementieren, diese versendet zu haben.

3.3.3 Szenario 1 - vertrauenswürdige Server

In Szenario 1 werden verschiedene auftretende Angriffsvektoren zwischen der Kommunikation von Alice und Bob betrachtet. Es wird davon ausgegangen, dass sich ein/e Angreifer/in im lokalen Netz befindet. Der/Die Angreifer/in hat jedoch keine Möglichkeit auf die XMPP-Server zuzugreifen. Benutzer/innen können diesen Servern somit vollständig vertrauen. Sinn und Zweck dieses Szenarios ist, die Sicherheit der Client-Server Beziehung zu überprüfen.[18, S. 2]

Unzureichende Validierung von Zertifikaten

Vertraulichkeit, Integrität und Verbindlichkeit der Client-Server Beziehung kann in XMPP mit Einsatz von TLS[4] erreicht werden. Je nach Ciphersuite werden symmetrische Schlüssel (z.B. für AES) mittels asymmetrischen, kryptographischen Verfahren wie RSA ausgetauscht. Die Datenintegrität stellen Hashalgorithmen wie SHA1 sicher. Bevor jedoch Schlüssel ausgetauscht werden und über einen verschlüsselten Kanal kommuniziert wird, müssen Clients erhaltende Serverzertifikate auf Gültigkeit validieren.[18, S. 3]

Sollte das vom Server erhaltende Zertifikat nicht ausreichend überprüft werden, sind MITM-Attacken möglich. Ein/e Angreifer/in, nachfolgend als Eve bezeichnet, kann somit die gesamte Kommunikation verfolgen, diese nach Belieben ändern oder filtern. Bei Klartext Authentifizierungen würde Eve sogar Benutzernamen und Passwort der authentifizierenden Person erhalten. Eve könnte sich dann als die entsprechende Person ausgeben. Dieser Vorgang wird als *Impersonation* bezeichnet. Um MITM-Attacken entgegenzuwirken, müssen daher Zertifikate vom Client immer auf ihre Gültigkeit validiert bzw. zum Vergleich zwischengespeichert werden.[18, S. 4]

Mangelhafte Authentifizierungsmechanismen

Weitere Angriffsvektoren bilden mangelhaft implementierte Authentifizierungsmethoden. Ein Beispiel dafür sind schlecht implementierte Pseudo-Zufallsgeneratoren in Challenge-Response Verfahren. XMPP spezifiziert in RFC 3920 die Verwendung von SASL[5]. SASL ermöglicht es, Authentifizierungsparameter zwischen Client und Server auszutauschen. Möchte sich ein Client gegenüber einem Server authentifizieren, so sendet der Client eine Authentifizierungsanfrage mit der gewünschten Authentifizierungsmethode an diesen Server. Unterstützt der Server die gewünschte Methode, wird der Authentifizierungsmechanis-

[4] Transport Layer Security
[5] Simple Authentication and Security Layer

mus der entsprechenden Methode initiiert.[21, S. 3][18, S. 7] SASL spezifiziert verschiedenste Authentifizierungsmechanismen, darunter Kerberos, GSSAPI, S/Key, DIGEST-MD5, usw.[21, S. 8-12][18, S. 7] Wählt der Client ein typisches Challenge-Response Verfahren wie DIGEST-MD5 zur Authentifizierung, dann ist darauf zu achten, dass die Zufallszahlen (Challenge) des Servers kryptographisch zufällig sein müssen. Sind sie das nicht, kann ein/e Angreifer/in mit Hilfe aufgezeichneter Authentifizierungsvorgängen eine Wörterbuch-Attacke durchführen und so möglicherweise das Passwort des/der Benutzers/Benutzerin herausfinden.[18, S. 7]

Nicht vorhandene Ende-zu-Ende-Verschlüsselung

Selbst wenn die Verbindung zum Server verschlüsselt sein sollte, kann ein/e Benutzer/in nicht überprüfen, ob die Server-Server Verbindung bzw. die Client-Server Verbindung des/der Kommunikationspartners/Kommunikationspartnerin ebenfalls verschlüsselt ist. Die Gefahr, dass jemand die Kommunikation belauscht, besteht also nicht nur im lokalen Netz. Das bedeutet, dass jeder Rechner, der Zugang zu den Netzen der Server *jabber.srv1.at* und *jabber.srv2.at* besitzt oder sich im Netz von Bob befindet, eine potentielle Gefahr darstellt.[18, S. 8f.]

3.3.4 Szenario 2 - nicht vertrauenswürdige Server

In Szenario 2 werden die XMPP-Server als nicht vertrauenswürdig eingestuft. Alice und Bob möchten nach wie vor miteinander kommunizieren. Diesmal hat Eve jedoch Zugriff auf einen der beiden Server. Ziel dieses Szenarios ist es, Gefahren aufzuzeigen, wenn ein/e Angreifer/in Zugang zu einem Server hat und dadurch jegliche Kommunikation belauschen kann.[18, S. 9]

Unverschlüsselte Metadaten

Ende-zu-Ende Verschlüsselung ist eine unerlässliche Maßnahme, die getroffen werden muss, um eine sichere Kommunikation zwischen Alice und Bob zu gewährleisten. Bei PGP wird zur Verschlüsselung der Nachricht der öffentliche Schlüssel vom Empfänger Bob herangezogen, der diese mit seinem privaten Schlüssel wieder entschlüsseln kann. Diese Vorgehensweise verhindert, dass die Server die Inhalte untereinander ausgetauschter Nachrichten lesen können. Eve kann folglich nicht bestimmen, was zwischen Alice und Bob kommuniziert wird.[18, S. 9f.]

Das Problem, das in diesem Fall besteht ist, dass nur ein Teil der schützenswerten Informationen gesichert ist. Die Nachrichten an sich sind zwar verschlüsselt, aber nicht deren Metadaten (Verbindungsda-

ten, Zeit, etc.). Die XMPP-Server benötigen Metadaten, um Nachrichten entsprechend weiterleiten zu können. Auch Präsenzinformationen sind nach wie vor ungeschützt, da sie an alle Kontake verschickt werden. Präsenzinformationen sind nur dann geschützt, wenn sichergestellt ist, dass mit allen Kontakten verschlüsselt kommuniziert wird. Eve besitzt daher Zugriff auf umfangreiche Informationen, mit denen festgestellt werden kann[18, S. 10] :

- wann Alice online ist,
- welche XMPP-Client Version Alice benutzt,
- auf welchem Betriebssystem dieser Client läuft,
- wann sich Alice am IM Server eingeloggt hat,
- wann Alice die letzte Nachricht versendet hat,
- wo sich Alice befindet (Geolocation),
- mit wem Alice kommuniziert,
- uvm.

Auch persönliche Daten, die in vCards am Server gespeichert sind, sind vor Eve ungeschützt und könnten jederzeit entwendet werden.[18, S. 10]

Abhandenkommen des Private-Keys

Selbst verschlüsselte Nachrichten sind mitunter nicht vor Eve sicher. Mit PGP verschlüsselte Nachrichten könnten von Eve gelesen werden. Die einzige Voraussetzung dafür ist, dass Eve den Private-Key des Empfängers, also von Bob, besitzt. Dies ist einer der Schwachpunkte von PGP. Jede verschlüsselte Nachricht kann mit dem entsprechenden Private-Key zu einem beliebigen späteren Zeitpunkt wieder entschlüsselt werden. Folglich gelten verschlüsselte Nachrichten nur in jener Zeit als privat und sicher, in welcher der Private-Key vor Dritten geschützt ist. Diesem Problem haben sich einige Wissenschaftler angenommen und mit OTR[6] haben sie ein Protokoll für private Konversationen entwickelt.[18, S. 10]
OTR bietet gegenüber von PGP zwei wichtige Eigenschaften[18, S. 10f.]:

- **Perfect Forward Secrecy**

  Bei OTR werden spezielle kurzlebige Sitzungsschlüssel für die Verschlüsselung von Konversationen eingesetzt. Diese leiten sich vom öffentlichen Schlüssel ab und sind vollkommen voneinander unabhängig. Sollte Eve an einen Sitzungsschlüssel gelangen, kann sie im schlimmsten Fall nur

[6] Off-The-Record

jene Nachrichten entschlüsseln, die mit diesem Sitzungsschlüssel verschlüsselt wurden. Im Gegensatz zu PGP ist es bei OTR nicht mehr möglich mit einem Schlüssel sämtliche aufgezeichnete Nachrichten zu entschlüsseln.[18, S. 10]

- **Malleable Encryption**

  Malleable Encryption hat zur Folge, dass jede Änderung eines Bits im verschlüsselten Text, dieselbe Auswirkung im unverschlüsselten Text bewirkt. Diese Eigenschaft wird dadurch erreicht, indem der Klartext mit einem Schlüsselstrom via XOR-Operationen verschlüsselt wird. Wüsste Eve den Klartext einer verschlüsselten Nachricht, könnte sie den chiffrierten Text dementsprechend ändern, um jeden beliebigen Klartext zu erhalten. Den chiffrierten Text kann Eve ohne den dazugehörigen Sitzungsschlüssel dennoch nicht entschlüsseln. Sinn und Zweck dieser Eigenschaft ist die sogenannte „Plausible Deniability “. Das bedeutet, dass beispielsweise Bob zwar die Authentizität einer Nachricht feststellen, jedoch nicht gegenüber von Dritten beweisen kann.[22, S. 79f.][18, S. 11]

3.4 Defensivmaßnahmen am Beispiel XMPP

Nachfolgend werden verschiedene Defensivmaßnahmen gegen die zuvor vorgestellten Angriffsvektoren beschrieben. Ziel ist es, Maßnahmen zu evaluieren, um Gefahrenquellen wie Information Disclosure zu minimieren bzw. zu verhindern. Zudem wird geklärt ob und wie diese Defensivmaßnahmen am Beispiel XMPP eingesetzt werden können.

3.4.1 Pretty Good Privacy (PGP)

PGP („Pretty Good Privacy“) ist ein von Phil Zimmermann entwickeltes hybrides, kryptographisches System zur Ver- und Entschlüsselung von Daten wie E-Mails.[23, S. 33]
Hybride Verschlüsselungssysteme kombinieren die Vorteile asymmetrischer und symmetrischer Verschlüsselungsmethoden. Asymmetrische Kryptosysteme werden aufgrund ihrer Eigenschaften für die Schlüsselverteilung eingesetzt, wohingegen symmetrische Systeme hinsichtlich ihrer Geschwindigkeitsvorteile zur Verschlüsselung von Daten Anwendung finden.[23, S. 9]

Funktionsweise

Die Funktionsweise von PGP läuft prinzipiell nach folgenden Schemata ab[23, S. 8]:

- **Verschlüsselung**
  1. Klartext wird zum Schutz kryptographischer Analysen komprimiert.
  2. Symmetrischer Sitzungsschlüssel zum einmaligen Gebrauch wird erstellt.
  3. Klartext wird mit diesem Schlüssel und einem schnellen und sicheren Verschlüsselungsalgorithmus (z.B. AES) chiffriert.
  4. Symmetrischer Sitzungsschlüssel wird mit dem öffentlichen, asymmetrischen Schlüssel des/der Empfängers/Empfängerin verschlüsselt.
  5. Der symmetrisch chiffrierte Klartext, als auch der asymmetrisch verschlüsselte Sitzungsschlüssel wird nun an den/die Empfänger/in übertragen.
- **Entschlüsselung**
  1. Mit dem privaten Schlüssel des/der Empfängers/Empfängerin wird der einmalige Sitzungsschlüssel wiederhergestellt.
  2. Der Sitzungsschlüssel wird nun verwendet, um die symmetrisch verschlüsselten Daten zu entschlüsseln und dadurch den Klartext zu erhalten.
  3. Der entschlüsselte Klartext wird zuletzt nur noch dekomprimiert.

Die Authentizität von öffentlichen Schlüsseln und somit die Gültigkeit von Zertifikaten wird bei PGP mit Hilfe des „Web of Trust“ Vertrauensmodell erreicht. Im Gegensatz zu hierarchisch angeordneten PKIs[7], ist das Web-of-Trust Modell dezentral aufgebaut. Ein/e PGP Benutzer/in kann mittels digitaler Signatur den öffentlichen Schlüssel einer anderen Person unterzeichnen. Damit bestätigt er/sie zum einen, dass er/sie dieser Person vertraut und zum anderen wird er/sie zur Certificate Authority des unterzeichneten Schlüssels.[23, S. 24-26]
Jeder/Jede PGP Benutzer/in hat daher die Möglichkeit als Zertifizierungsstelle zu agieren und die Gültigkeit von Zertifikaten anderer Benutzer/innen zu bestätigen. Zertifikate sind allerdings nur dann für andere PGP Anwender/innen gültig, wenn sie die überprüfende Person als autorisierten Schlüsselverwalter akzeptieren. Zu diesem Zweck speichert PGP öffentliche Schlüssel inklusive Vertrauens- und Gül-

[7] Public Key Infrastructure

tigkeitsstufen in den Schlüsselbund eines/einer Anwenders/Anwenderin ab. Durch diesen Ansatz wird allmählich ein Netz aufgebaut, das Web of Trust.[23, S. 26]
In PGP kann ein Schlüssel als gültig, zweitrangig gültig oder als ungültig erachtet werden. Zusätzlich gibt es folgende vier verschiedene Vertrauensstufen eines öffentlichen Schlüssels [23, S. 26]:

- **implizites Vertrauen**

  Die höchste Vertrauensstufe, die einem öffentlichen Schlüssel in PGP verliehen werden kann, gilt dem öffentlichen Schlüssel eines/einer Anwenders/Anwenderin selbst. Jeder Schlüssel, der mit dieser Vertrauensstufe signierte wurde, wird als gültig anerkannt.

- **volles Vertrauen**

  Wenn einem gültigen Schlüssel eines/einer Benutzers/Benutzerin volles Vertrauen zugewiesen wird, gilt der/die Benutzer/in als Certificate Authority. Die von dieser CA digital signierten Schlüssel, werden ebenfalls als gültige Schlüssel akzeptiert.

- **eingeschränktes Vertrauen**

  Ein Schlüssel, der von einer CA mit eingeschränktem Vertrauen signiert wurde, gilt nicht sofort als gültig, sondern als zweitrangig gültig. Der/Die PGP Anwender/in erkennt diesen Schlüssel erst dann als gültig an, wenn er zumindest von einer zweiten CA mit eingeschränktem Vertrauen unterzeichnet wurde.

- **kein Vertrauen**

  Schlüssel von Personen, denen nicht vertraut wird, werden folglich auch nicht als autorisierte Schlüsselverwalter angesehen. Folglich bedeutet das, dass Schlüssel, die mit dieser Vertrauensstufe signiert wurden, nicht verifiziert werden können.

Abbildung 3.2 gibt einen guten Überblick über die vorgestellten Konzepte eines Web-of-Trust Vertrauensnetzes[24]. Unter anderem wird veranschaulicht, wie Personen sich untereinander vertrauen, welche Stellung diese Vertrauensbeziehungen haben und ob und wie die Authentizität verifiziert werden kann.

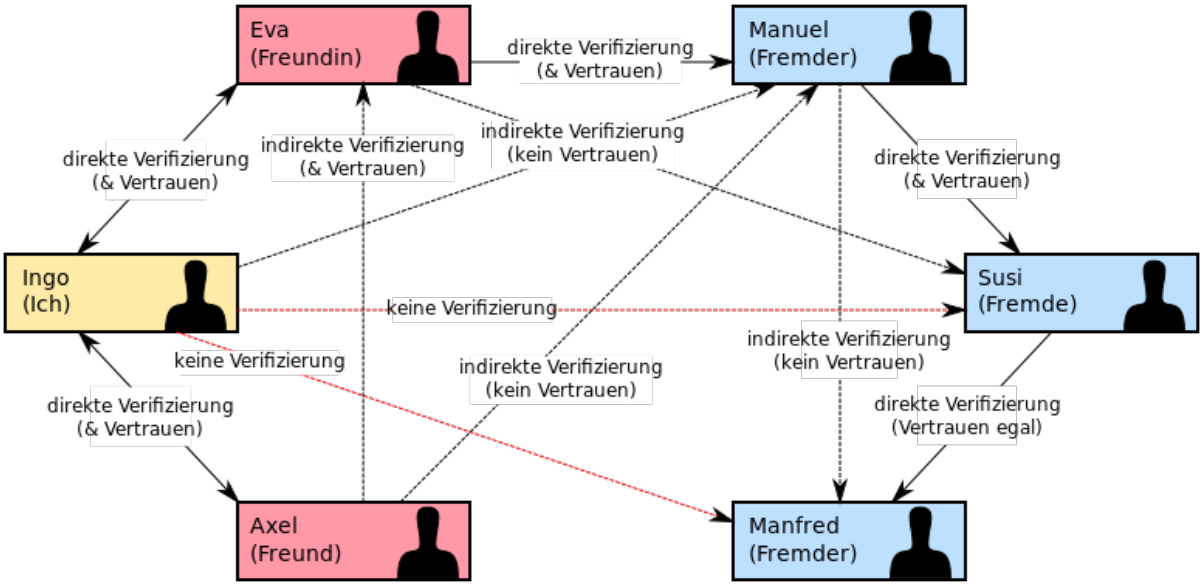


Abbildung 3.2: Web of Trust[24]

PGP in XMPP

PGP wurde als XMPP-Erweiterung in XEP[8] 27 definiert. Um verschlüsselt kommunizieren zu können, besitzt jede XMPP-Entität einen privaten Schlüssel und den dazugehörigen öffentlichen Schlüssel. Die Schlüsselverteilung öffentlicher Schlüssel geschieht über eigens dafür vorgesehene Keyserver. Auf diesen Servern werden die öffentlichen Schlüssel, zusammen mit den digitalen Signaturen, der XMPP-Entitäten gespeichert. Die Authentizität und somit die Vertrauenswürdigkeit eines Schlüssels wird mit dem zugrundeliegenden Vertrauensmodell von PGP, dem Web-of-Trust, verifiziert.[18, S. 11]

Bei Verwendung von PGP in XMPP werden Presence-Nachrichten digital signiert. Einerseits wird dadurch erreicht, dass andere Personen über die Möglichkeit verschlüsselte Nachrichten zu senden informiert werden. Andererseits kann überprüft werden, ob die entsprechende Signatur tatsächlich von der vermeintlichen Person stammt. Es ist allerdings nicht möglich, Presence-Nachrichten zu signieren und zu verschlüsseln. Darüber hinaus sind Replay-Angriffe möglich. Das bedeutet, dass ein/e Angreifer/in den Onlinestatus einer Person vortäuschen kann, indem er/sie aufgezeichnete Presence-Nachrichten einer anderen Person versendet. Die Voraussetzung dafür ist, dass der/die Angreifer/in das Passwort der entsprechenden Person kennt oder Zugriff auf den XMPP-Server bzw. auf das Servernetzwerk hat.[18, S. 12]

Gewöhnliche Sofortnachrichten werden nach XEP 27 zwar verschlüsselt, dafür aber nicht signiert. Für

[8] XMPP Extension Protocol

die Verschlüsselung wird der öffentliche Schlüssel des/der Empfängers/Empfängerin verwendet. Aufgrund der fehlenden digitalen Signatur kann ein/e Empfänger/in die Authentizität des/der Verfassers/-Verfasserin einer Nachricht nicht überprüfen. Ein/e Angreifer/in hätte daher auch hier die Möglichkeit, Nachrichten einzuschleusen. Dazu benötigt er/sie lediglich Zugriff auf einen XMPP-Server und den öffentlichen Schlüssel des/der Empfängers/Empfängerin.[18, S. 12f.]
PGP für XMPP bietet zwar ein gewisses Maß an Sicherheit, in Hinblick auf Sicherheit von Instant Messaging und Presence kann es jedoch nicht empfohlen werden. Für diesen speziellen Einsatzzweck ist PGP nicht gedacht. Dies wird durch folgende Sicherheitsbedenken bekräftigt[18, S. 13]:

- Sowohl Replay-Attacken aufgezeichneter Presence-Nachrichten, als auch die Einschleusung nicht signierter Sofortnachrichten sind möglich.
- Alle aufgezeichneten Nachrichten können mit dem entsprechenden privaten Schlüssel entschlüsselt werden.
- Die Authentizität bzw. der Ursprung einer Nachricht kann nicht überprüft werden.

3.4.2 Off-the-Record (OTR)

Das Off-the-Record (OTR) Messaging Protokoll wurde von Nikita Borisov, Ian Goldberg und Eric Brewer entwickelt und im Jahr 2004 in Form eines wissenschaftlichen Papers veröffentlicht. Die Wissenschaftler hatten sich zum Ziel gesetzt ein Protokoll zu entwickeln, das sichere und private Konversationen über eine Kommunikationsform mit niedriger Latenz wie Instant Messaging ermöglicht.[22, S. 77]

Relevante Eigenschaften

In OTR wurde das Hauptaugenmerk auf drei grundlegende Elemente gerichtet, um private Konversationen zu gewährleisten[22, S. 78]:

- **Perfect Forward Secrecy**

  Einer der größten Probleme von PGP wurde mit dieser Eigenschaft in OTR behoben. Zur Verschlüsselung von Nachrichten kommen kurzlebige Schlüssel zum Einsatz, die nach einmaligen Gebrauch verworfen und wieder neu erzeugt werden. Diese Vorgehensweise garantiert, dass vergangene, derzeitige und zukünftige Nachrichten geschützt sind. Dazu wird jede Nachricht mit einem anderen Schlüssel chiffriert, wodurch niemand komplette Konversationen entschlüsseln kann.

Die Schlüssel werden mit dem Diffie-Hellman Protokoll erzeugt. Dieses beruht darauf, dass sich zwei Kommunikationspartner/innen auf ein gemeinsames Geheimnis einigen. Dieses gemeinsame Geheimnis ermöglicht beiden Kommunikationspartner/innen, denselben symmetrischen Schlüssel zu generieren (z.B. mit einer Hashfunktion wie SHA-1).[22, S. 79]

- **Message Authentication Codes (MAC)**

  Indem langlebige asymmetrische Schlüssel verwendet werden, ermöglichen digitale Signaturen, die Authentizität eines/einer Verfassers/Verfasserin einer Nachricht zu verifizieren, . OTR verzichtet dennoch auf digitale Signaturen. Dies hat den Grund, dass mit digitalen Signaturen die Eigenschaft der Bestreitbarkeit verloren geht.[22, S. 79]

  Ein Beispiel: Alice möchte Bob eine Nachricht schicken und signiert diese mit ihrem privaten Schlüssel. Nach Erhalt der Nachricht kann Bob mit dem öffentlichen Schlüssel von Alice überprüfen, ob diese tatsächlich von Alice stammt. Bob kann somit die Authentizität von Alice feststellen. Bob kann nun auch gegenüber Dritten beweisen, dass diese Nachricht von Alice stammt, da ausschließlich Alice den privaten Schlüssel besitzt.[22, S. 79]

  Dieses Problem wurde in OTR berücksichtigt und daher werden MACs an Stelle von digitalen Signaturen verwendet. Im Fall von OTR teilen sich Alice und Bob ein zusätzliches gemeinsames Geheimnis, den MAC-Schlüssel. Dieser wird zur Generierung von MACs eingesetzt, die im einfachsten Fall symmetrisch verschlüsselte Hashwerte repräsentieren, wobei die Hashwerte wiederum von den zu übermittelnden Nachrichten berechnet werden.[22, S. 79]

  Wenn Alice an Bob nun eine Nachricht versenden möchte, sendet sie ihm neben der verschlüsselten Nachricht, zusätzlich den davon berechneten MAC. Dies hat nun mehrere Vorteile. Zum einen kann Bob die Integrität, als auch die Authentizität überprüfen. Schließlich weiß nur Alice über den MAC-Key Bescheid. Zum anderen kann Bob nicht beweisen, dass Alice diese Nachricht verfasst hat. Bob hätte nämlich ebenso diese Nachricht an Stelle von Alice verfassen können, da sie beide denselben MAC-Key kennen. Ein weiterer Vorteil von MACs ist, dass selbst wenn Nachrichten belauscht und entschlüsselt werden können, der/die Verfasser/in nicht festgestellt werden kann.[22, S. 79]

- **Malleable Encryption**

  Malleable Encryption ist eine Schwäche der Verschlüsselung, die es prinzipiell zu vermeiden gilt. In OTR wird diese jedoch bewusst ausgenützt, um die Eigenschaft der „Plausible Deniability“ also

der Bestreitbarkeit von Nachrichten nochmals zu stärken. Dies wird erreicht, indem Nachrichten mit einer einfachen Stromverschlüsselung (AES in CTR Modus) verschlüsselt werden. Das bedeutet, dass Nachrichten mit dem Schlüsselstrom mit XOR-Operationen ver- und entschlüsselt werden. Jede Bit-Änderung in einem derart verschlüsselten Text wirkt sich gleich auf den entschlüsselten Text aus. Prinzipiell ist dies äußerst unsicher, da eine Angreiferin wie Eve gezielt einen verschlüsselten Text ändern kann, sodass dies einen von ihr bestimmten entschlüsselten Text ergibt. Voraussetzung dafür ist, dass Eve den Inhalt einer Nachricht kennt. Doch genau diese unsichere Eigenschaft mach sich OTR zu Nutze. Eve sowie auch andere Personen haben jederzeit die Möglichkeit verschlüsselte Nachrichten zu ändern, was der Eigenschaft der Bestreitbarkeit zu Gute kommt. Eine mit einer Stromverschlüsselung chiffrierte Nachricht garantiert daher keine gültige Integrität bzw. Authentizität. Aus diesem Grund kommen in OTR schließlich MACs zum Einsatz.[22, S. 80]

Funktionsweise

Die folgenden vier Schritte beschreiben die Funktionsweise von OTR[22, S. 80f.]:

1. **Verschlüsselung von Nachrichten**

   Nachrichten werden mit dem symmetrischen Verschlüsselungsalgorithmus AES im CTR Modus verschlüsselt, um einerseits Vertraulichkeit und anderseits Bestreitbarkeit zu gewährleisten. Jede Nachricht wird mit einem eigenen symmetrischen Schlüssel verschlüsselt. Zu diesem Zweck einigen sich Alice und Bob mit Hilfe des Diffie-Hellman Protokolls bei jeder Nachricht auf einen gemeinsamen Schlüssel.[22, S. 80]

2. **Verwerfen von Schlüsseln**

   Um Perfect Forward Secrecy zu ermöglichen, müssen alte, bereits verwendete Schlüssel verworfen werden. Dies geschieht nachdem sich Alice und Bob auf neue Schlüssel geeinigt haben. Der Nachrichtenaustausch erfolgt jedoch meist asynchron. Alice müsste sich daher für alle Nachrichten, die sie an Bob versendetet hat, die entsprechenden Schlüssel merken. Dies hat den Grund, dass eine bestimmte Nachricht von Bob möglicherweise noch nicht bei Alice angekommen ist. Erst wenn Bob eine Nachricht, die mit dem aktuellsten Schlüssel verschlüsselt wurde, an Alice als Antwort versendet hat, kann Alice die Schlüssel bedenkenlos verwerfen.[22, S. 80f.]

   Aufgrund dieser Tatsache werden neue Schlüssel in OTR nur dann generiert, wenn ein/e Kommu-

nikationspartner/in eine Antwort auf vorhergehende Nachrichten erhalten hat. Dies impliziert, dass sich Alice maximal zwei Schlüssel merken muss. Einen Schlüssel für Nachrichten, die sich möglicherweise noch im Umlauf befinden und einen zweiten Schlüssel, der von Bob für eine Antwort auf aktuelle Nachrichten von Alice verwendet werden kann. Damit Alice nicht ständig denselben Schlüssel für ihre Nachrichten verwendet, wird Bob regelmäßig leere Nachrichten verschicken, um sich auf neue Schlüssel zu einigen.[22, S. 80f.]

3. **Überprüfung der Authentizität**

   OTR Benutzer/innen besitzen langlebige asymmetrische Schlüssel für die initiale Schlüsselvereinbarung. Zu Beginn einer Sitzung werden die untereinander ausgetauschten Diffie-Hellman Nachrichten, die zur Generierung eines Schlüssels benötigt werden, digital signiert. Alice bzw. Bob können sich somit sicher sein, dass die erhaltenen Nachrichten von der jeweiligen Person stammen. Voraussetzung dafür ist allerdings, dass die öffentlichen Schlüsseln auf einem sicheren Kommunikationskanal ausgetauscht wurden bzw. die Echtheit dieser Schlüssel verifiziert werden können (z.B. über PGP Fingerprints). Wenn sichergestellt ist, dass die ersten Diffie-Hellman Nachrichten richtig verifiziert wurden, sind somit auch die nachfolgenden Schlüsselvereinbarungen sicher. Von dem gemeinsamen Geheimnis, auf das man sich mit dem Diffie-Hellman Protokoll geeinigt hat, wird ein Hashwert berechnet. Dieser Hashwert ergibt den MAC-Schlüssel, welcher für die Erzeugung von MACs verwendet wird. MACs garantieren wiederum die Integrität, als auch die Authentizität ohne dabei die Eigenschaft der Bestreitbarkeit zu verlieren.[22, S. 81]

4. **Veröffentlichung der MAC-Schlüssel**

   Eine zusätzliche Sicherheitsfunktion von OTR ist, dass MAC-Schlüssel nach Gebrauch veröffentlicht werden. Das bedeutet, dass jede Person willkürliche Nachrichten, die von diesem MAC-Schlüssel Gebrauch machen, erstellt haben könnte. Niemand kann daher den Ursprung einer Nachricht feststellen bzw. behaupten.[22, S. 81]

   Ein Beispiel: Alice sendet Bob mehrere Nachrichten, deren MAC mit einem bestimmten MAC-Schlüssel erzeugt wurden. Wenn Bob alle Nachrichten erhalten hat, kann Alice ihren MAC-Schlüssel bedenkenlos veröffentlichen. Eve könnte nun ebenso eine Nachricht unter der Verwendung dieses MAC-Schlüssel versenden. Da dieser jedoch bereits veröffentlicht wurde und Bob alle empfangenen Nachrichten überprüft hat, ist dieser Schlüssel veraltet und daher wertlos. Dies hat den Grund, dass jegliche Person diese Nachricht verfassen hätte können.[22, S. 81]

OTR in XMPP

Viele IM Clients unterstützen OTR von Haus aus. Darüber hinaus werden auch sehr viele Third Party Plugins angeboten, die Clients um OTR Funktionalitäten erweitern. Das OTR Entwickler-Team selbst bietet zudem Funktionsbibliotheken für die verschiedensten Programmiersprachen an. Auf folgender Website findet man eine Übersicht über das Softwareangebot: `https://otr.cypherpunks.ca/software.php`.

OTR ist zwar eine sichere Alternative zu PGP, dennoch treten zeitweise Probleme in IM Clients auf. Dies ist zum Beispiel der Fall, wenn der Status einer Sitzung zweier unterschiedlicher Clients nicht auf demselben Stand ist. Ein Client denkt, dass bereits ein neuer Sitzungsschlüssel vereinbart wurde, während der andere Client hingegen noch einen alten Schlüssel verwendet. Ein weiteres, möglicherweise auftretendes Problem ist, wenn Nachrichten in unterschiedlicher Reihenfolge eintreffen. Diese können dann nicht mehr richtig entschlüsselt werden, da der falsche Sitzungsschlüssel verwendet wird. Auch das Versenden von Offline-Nachrichten ist problematisch. Wenn Alice an Bob Nachrichten verschickt, dieser jedoch genau zu diesem Moment den Client beendet, kann Bob später empfangene Nachrichten nicht mehr entschlüsseln. Sobald Bob den Client beendet, wird auch die OTR Sitzung beendet und somit der Schlüssel verworfen.[18, S. 14]

Versuche in einer Testumgebung haben gezeigt, dass es zu Problemen kommt, wenn ein/e Anwender/in auf mehren Clients mit demselben Account eingeloggt ist. In einem Testszenario wurden verschiedene XMPP Clients unter Android und OS X getestet. Darunter: Xabber, ChatSecure, Conversations, Adium und Pidgin. OTR Probleme traten immer dann auf, wenn eine Person mit dem mobilen XMPP Client bzw. mit dem Client auf dem Desktop-Rechner gleichzeitig online war und ein Kontakt eine verschlüsselte Verbindung initiieren wollte. Beide Clients antworteten in diesem Fall auf die Verschlüsselungsanfrage und führten folglich zu einem fehlerhaften Schlüsselaustausch.

Bis auf einige kleine Probleme ist OTR auf alle Fälle PGP vorzuziehen. Es wurde speziell für sichere und private Konversationen entwickelt und bietet mit Hilfe unterschiedlicher, kryptographischer Verfahren ein hohes Maß an Sicherheit.

3.4.3 The Onion Router (Tor)

Die zuvor vorgestellten Maßnahmen bieten zwar Schutz für den Inhalt und Transport von Nachrichten, allerdings sind Metadaten wie Verbindungsdaten nach wie vor ungeschützt. Verbindungsdaten können nicht Ende-zu-Ende verschlüsselt werden, da sie für die Zustellung zur richtigen Person auf dem Trans-

portweg gelesen werden müssen. Diese Daten könnten bereits vor Gericht für eine Entscheidungsgrundlage ausreichen. Eine weitere Voraussetzung für sichere und private Kommunikation im Internet ist daher die Sicherstellung der Anonymität. Die technologische Basis dafür bietet Tor.[25, S. 1]

Funktionsweise

Tor ist eine leitungsbasierende Routingtechnologie zur Wahrung der Anonymität von anfallenden Netzwerkverkehr von TCP-Anwendungen wie Webbrowsing, Secure Shell (ssh), Instant Messaging uvm. Client-Anwendungen kommunizieren dazu mit einem lokalen Proxy, um einen zufälligen Pfad durch das Tor-Netzwerk zu wählen und somit einen Circuit (*deutsch:* Leitung) aufzubauen. Der Pfad besteht aus mehreren Knoten, den sogenannten Onion Routern (OR). Jeder Knoten kennt dabei ausschließlich seinen Vorgänger und Nachfolger. Über diese aufgebaute Leitung wird schließlich Netzwerkverkehr in schichtförmig verschlüsselten Zellen fixer Größer geroutet, wobei jeder Knoten die äußerste Schicht der Zelle mit einem symmetrischen Schlüssel entschlüsselt. Die Zelle wird also während der Weiterleitung entlang des Pfades von jedem Knoten Schicht für Schicht entpackt, ähnlich wie die Schalen einer Zwiebel. Dieses Zwiebelschalenprinzip hat zur Folge, dass der Ursprung von TCP-Segmenten und somit die IP-Adresse des Geräts verschleiert wird. Die Anonymität einer Person ist damit sichergestellt.[25, S. 1]
Das Tor-Netzwerk besteht im Prinzip aus zwei zentralen Komponenten[25, S. 5f.]:

- **Onion Router**

  Onion Router sind für die Weiterleitung von Daten verantwortlich und managen den Aufbau, den Abbau als auch Änderungen von Circuits. Jeder Router besitzt einen langlebigen Schlüssel für digitale Signaturen (z.B. für TLS-Zertifikate, OR Deskriptoren, Directories, etc.) und einen kurzlebigen Schlüssel zum Entschlüsseln von Anfragen für die Erstellung neuer Circuits. Neben diesen Schlüsseln werden aufgrund des TLS Protokolls weitere kurzlebige Schlüssel für die Kommunikation zwischen Onion Routern eingesetzt.

- **Onion Proxy**

  Jeder/Jede Anwender/in, der/die seine/ihre Anonymität mit Hilfe des Tor-Netzwerks sicherstellen möchte, benötigt auf dem entsprechenden Gerät (z.B. PC) eine spezielle lokale Software, den Onion Proxy. Der Onion Proxy ist dafür zuständig Verzeichnisinformationen zu beziehen, Circuits aufzubauen und Verbindungen von Anwendungen zu managen. Die Proxy Software bildet für Anwendungen somit eine transparente Schnittstelle (via SOCKS4, SOCKS5 oder HTTP) zu Tor.

Für die nähere Beschreibung der Funktionsweise von Tor werden nun die Client- und Serverseite näher betrachtet.

Clientseitige Anonymität

In Abbildung 3.3 wird ein Beispiel veranschaulicht, das die Anonymität von Alice sicherstellt. Alice möchte dabei eine Website im Internet anonym abrufen. Alices Onion-Proxy wird zu diesem Zweck einen neuen Circuit erstellen, um den TCP-Stream zwischen Alices Webbrowser und dem Webserver der Website zu anonymisieren und zu verschlüsseln. Zur Vereinfachung des Beispiels besteht der Circuit nur aus zwei Onion Routern. Das Konzept, wie ein Circuit aufgebaut wird, kann allerdings auf beliebig viele Onion Router übertragen werden.[25, S. 6]
Die folgenden Punkte beschreiben, welche Schritte für den Aufbau des Circuits und die Weiterleitung von Zellen benötigt werden. Dabei wird die folgende Abbildung als Referenzbeispiel zur Veranschaulichung herangezogen, um eine Website via Tor anonym aufzurufen[25, S. 6f.]:

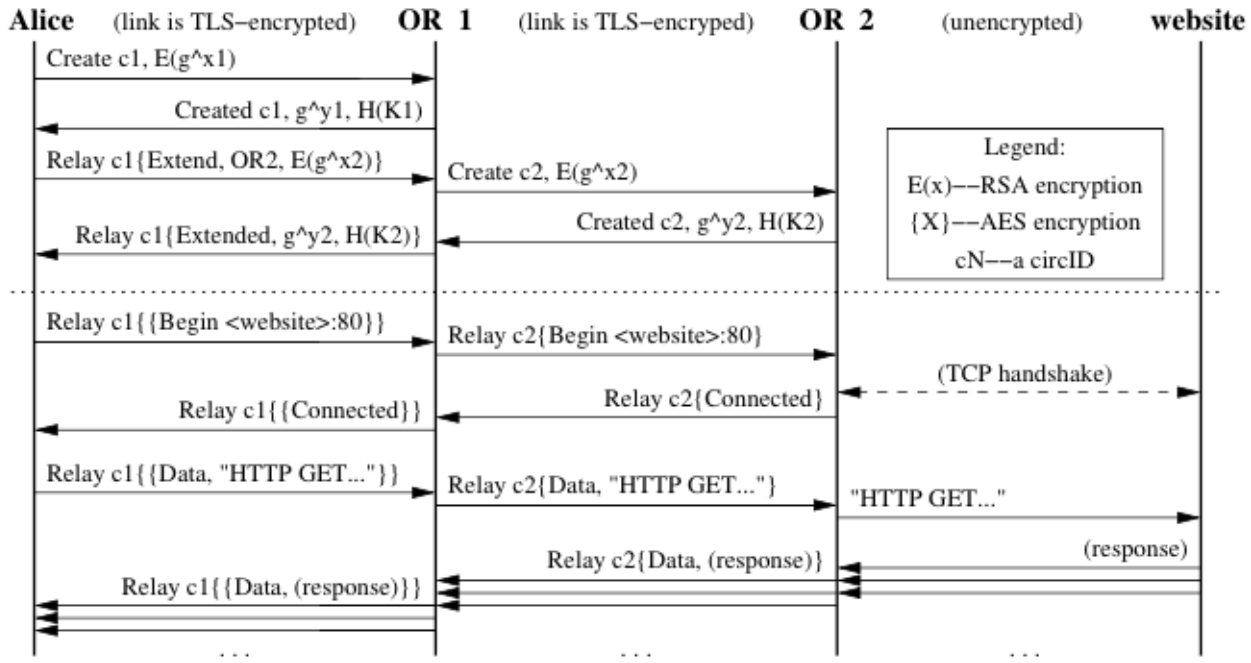


Abbildung 3.3: Clientseitige Anonymität via Tor[25, S. 6]

1. Schritt - Alice OP wählt zwei zufällige Knoten im Tor-Netzwerk, die einen Circuit bilden sollen. Sie schickt dem ersten Knoten dieses Pfades, in diesem Fall *OR 1*, eine Zelle mit dem Kommando *Create*. Diese Zelle beinhaltet den ersten Teil eines Diffie-Hellman Handshakes, welcher mit dem öffentlichen Schlüssel des ersten Knoten verschlüsselt wurde.[25, S. 6f.]

2. Schritt - Onion Router *OR 1* wird daraufhin eine *Created* Zelle als Antwort schicken. Diese beinhaltet als Payload den zweiten Teil des Diffie-Hellman Handshake. Darüber hinaus beinhaltet diese Zelle auch einen Hashwert, der von dem soeben vereinbarten symmetrischen Schlüssel *K1* berechnet wurde. *Alice* und *OR 1* besitzen nun mit dem Schlüssel *K1* ein gemeinsames Geheimnis und können fortan mit symmetrisch verschlüsselten Daten miteinander kommunizieren.[25, S. 6f.]

3. Schritt - *Alice* wird nun als nächstes den Circuit um einen Knoten *OR 2* erweitern. Zu diesem Zweck schickt sie eine *Relay* Zelle an *OR 1*. Die Payload dieser Zelle ist mit dem symmetrischen Schlüssel *K1* verschlüsselt, sodass ausschließlich *OR 1* den Inhalt entschlüsseln kann. Diese Zelle beinhaltet das Kommando *Extend*, die Adresse von *OR 2* und den ersten Teil eines Diffie-Hellman Handshake, der dieses mal jedoch mit dem öffentlichen Schlüssel von *OR 2* verschlüsselt wurde.[25, S. 6f.]

4. Schritt - *OR 1* entschlüsselt die erhaltene *Relay* Zelle und baut selbst einen neuen Circuit zu *OR 2* auf, indem eine *Create* Zelle an diesen Onion Router verschickt wird.[25, S. 6f.]

5. Schritt - *OR 2* antwortet wiederum mit einer *Created* Zelle (Siehe Schritt 2).[25, S. 6f.]

6. Schritt - *OR 1* antwortet nach Erhalt der *Created* Zelle mit einer *Relay Extended* Zelle auf *Alice* ursprüngliche Anfrage. Diese Zelle beinhaltet das Kommando *Extended*, den zweiten Teil des Diffie-Hellman Handshake von *OR 2* und den Hashwert des Schlüssels *K2*. Die Payload dieser Zelle ist wiederum mit dem Schlüssel *K1* verschlüsselt, sodass ausschließlich *Alice* den Inhalt dieser Zelle lesen kann.[25, S. 6f.]

7. Schritt - *Alice* Circuit besteht nun aus zwei Knoten (*OR 1* und *OR 2*) und sie besitzt für jeden Knoten einen symmetrischen Schlüssel, welche die Vertraulichkeit zwischen *Alice* und dem entsprechenden Knoten sicherstellt. Wenn *Alice* nun eine Website anonym abrufen möchte, erstellt sie eine *Relay* Zelle. Diese Zelle beinhaltet als Payload das Kommando *Begin* und die URL der Website. Das *Begin* Kommando veranlasst den Exit-Knoten (*OR 2*) eine TCP Verbindung zur angegebenen Adresse aufzubauen. Die äußerste Schicht der Relay Zelle ist mit dem Schlüssel *K1*

verschlüsselt, der Nachrichteninhalt mit dem Kommando und der URL jedoch mit dem Schlüssel *K2*. Damit wird erreicht, dass die Zelle Knoten für Knoten entschlüsselt wird, bis der letzte Knoten schließlich den unverschlüsselten Inhalt der Zelle lesen kann. Die Antwort des Webservers wird wiederum in umgekehrter Reihenfolge Knoten für Knoten verschlüsselt, sodass *Alice* am Ende alle symmetrischen Schlüssel ihres Circuits benötigt, um den Inhalt einer erhaltenen Zelle lesen zu können.[25, S. 6f.]

Serverseitige Anonymität

Mit sogenannten *Hidden Services* ermöglicht Tor auch die Sicherstellung der serverseitigen Anonymität. Hidden Services sind Dienste, die ausschließlich über das Tor Netzwerk erreichbar sind. In Abbildung 3.4 wird Schritt für Schritt veranschaulicht, wie Bob einen Dienst als Hidden Service bereitstellt und wie Alice auf diesen Dienst Zugriff erhält.[26]

1. Schritt (Siehe Abbildung 3.4(a)) - Bob möchte einen Hidden Service betreiben und wählt dazu eine beliebige Anzahl an *Introduction Points* (in diesem Fall sind es drei), zu welchen Bob jeweils einen eigenen Circuit aufbaut. Diesen *Introduction Points* wird der Public Key des Hidden Services übermittelt. Aufgrund der Tor-Circuits kann kein *Introduction Point* auf die öffentliche IP-Adresse von Bobs Hidden Server rückschließen.[26]

2. Schritt (Siehe Abbildung 3.4(b)) - Bob erstellt einen Hidden Service Deskriptor, der aus dem Public Key des Hidden Services und einer Beschreibung aller Introduction Points besteht. Der Deskriptor wird schließlich mit Bobs Private Key signiert und einer Distributed Hash Table bekannt gemacht. Der Hidden Service wurde hiermit erfolgreich eingerichtet. Mit der Onion Adresse des Hidden Services erhalten Clients wie Alice Zugriff auf den anonymen Dienst. Eine Onion Adresse wird vom Public Key des Hidden Service abgeleitet, besteht aus 16 Zeichen und hat die Form <*xyz...*>.*onion* (z.B. *my3uzkssaho2jli7.onion*).[26]

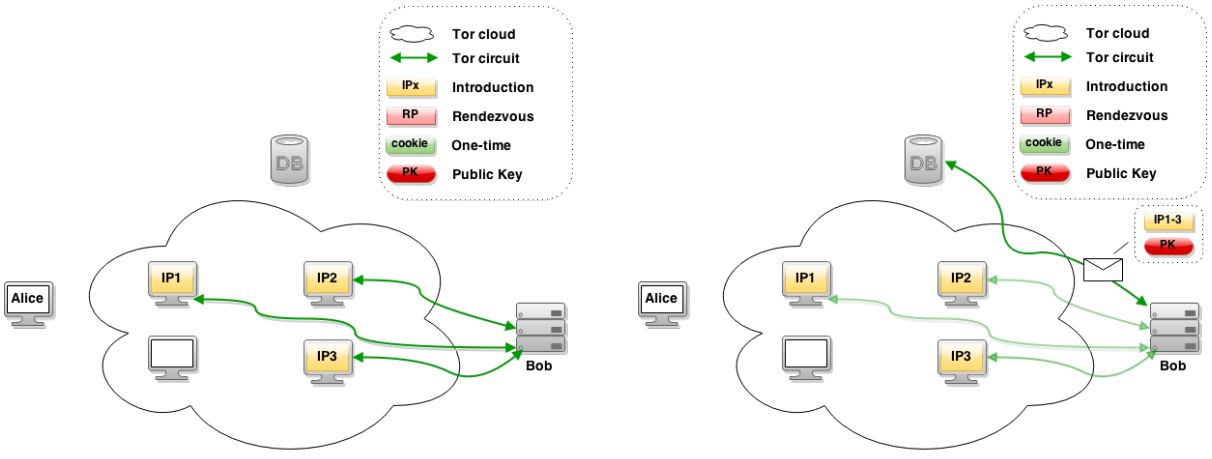
3. Schritt (Siehe Abbildung 3.4(c)) - Alice erfährt die Onion Adresse von Bobs Hidden Service über einen sicheren Kanal (z.B. via Telefon). Als nächstes baut Alice einen sicheren Circuit auf, um den Hidden Service Descriptor von einer Distributed Hash Table zu beziehen. Von diesem Deskriptor erfährt Alice, welche Introduction Points für diesen Hidden Service zur Verfügung stehen und welchen Public Key sie benutzen muss. Anschließend baut sie einen weiteren Circuit zu einem zufällig gewählten Knoten im Tor-Netzwerk auf, der als *Rendezvous Point* dient. Diesem übermittelt

sie ein einmaliges Geheimnis, das *Rendezvous Cookie*.[26]

4. Schritt (Siehe Abbildung 3.4(d)) - Alice erstellt eine Nachricht, welche die Adresse des *Rendezvous Points* und das *Rendezvous Cookie* beinhaltet. Die Nachricht ist mit dem öffentlichen Schlüssel des Hidden Services verschlüsselt. Danach baut Alice einen sicheren Circuit zu einem der angebotenen Introduction Points auf, dem die verschlüsselte Nachricht übermittelt wird. Der Introduction Point leitet diese Nachricht schließlich an Bob weiter. Da alle Nachrichten über Tor-Circuits weitergeleitet werden, bleiben sowohl Alice als auch Bob stets anonym.[26]

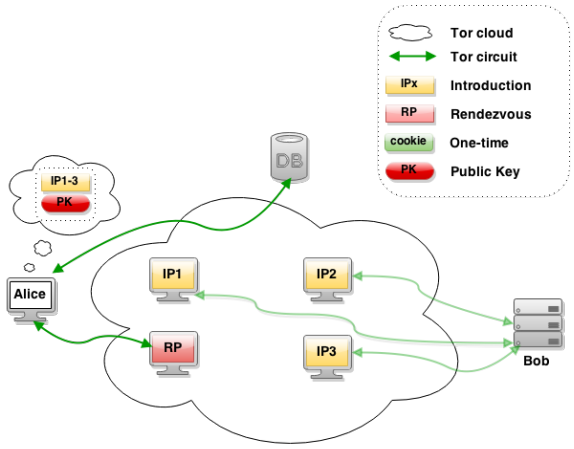
5. Schritt (Siehe Abbildung 3.4(e)) - Bob entschlüsselt die erhaltene Nachricht mit seinem Private Key und erfährt dadurch die Adresse des Rendezvous Points und das einmalige Geheimnis. Bob wird nun einen neuen Circuit zu dem Knoten, der als Rendezvous Point dient, aufbauen und diesem das Geheimnis übermitteln.[26]

6. Schritt (Siehe Abbildung 3.4(f)) - Wenn das von Bob übermittelte Geheimnis mit dem Geheimnis von Alice übereinstimmt, wurde eine erfolgreiche Verbindung aufgebaut. Der Rendezvous Point wird in diesem Fall Alice über die erfolgreich aufgebaute Verbindung benachrichtigen. Von diesem Zeitpunkt an können Alice und Bob mit deren Circuits anonym Ende-zu-Ende verschlüsselte Nachrichten miteinander austauschen. Der Rendezvous Point dient dabei lediglich als Relay für ankommende Nachrichten.[26]

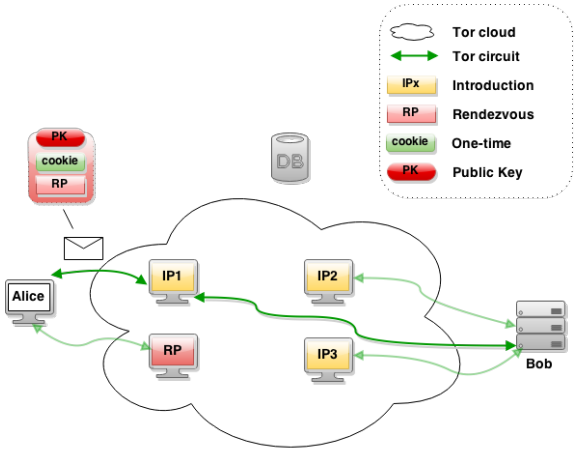


(a) Einrichtung eines neuen Hidden Services[26]

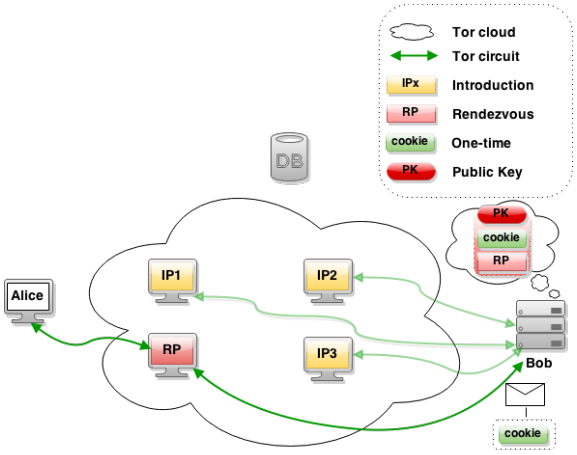
(b) Veröffentlichung der Onion Adresse[26]



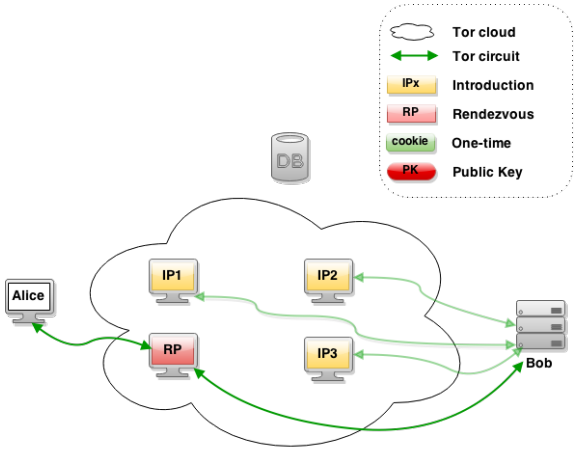
(c) Verbindungsaufbau zu einem Rendezvous Point[26]



(d) Verbindungsanfrage an einen Hidden Service[26]



(e) Verbindung zwischen Rendezvous Point und Hidden Service[26]



(f) Verschlüsselte und anonyme Kommunikation mit einem Hidden Service[26]

Abbildung 3.4: Serverseitige Anonymität via Tor[26]

Tor in XMPP

Die Einrichtung eines Hidden Services für einen XMPP Server ermöglicht anonyme und sichere Kommunikation aller auf diesem Server registrierter Clients. Die Clients müssen jedoch den Tor Onion Proxy installieren, damit der XMPP Client des/der jeweiligen Benutzers/Benutzerin Zugriff auf die Onion Adresse und somit auf den Hidden Service erhält. TLS wird für die Client-Server Beziehung nicht mehr benötigt, da die Verbindung zwischen Client und Hidden Service bereits verschlüsselt ist. Vorsicht ist geboten, wenn eine XMPP Entität auf einem anonymen XMPP Server mit einer XMPP Entität auf einem öffentlichen Server im Internet kommunizieren möchte. In diesem Fall ist die Anonymität des/der Benutzers/Benutzerin unter Umständen aufgehoben. Es wird daher empfohlen, dass anonyme XMPP Server ausschließlich mit anderen anonymen XMPP Servern kommunizieren sollten.[18, S. 15ff.]

Ein über Tor erreichbarer XMPP Server kann selbst sehr schnell und einfach eingerichtet werden. Der Autor dieser Arbeit hat beispielsweise innerhalb weniger Minuten einen XMPP Server aufgesetzt und die dafür notwendigen Hidden Services eingerichtet. Für die Testserverumgebung ist ein Raspberry Pi mit Raspbian zum Einsatz gekommen. Auf diesem System wurde die vielfach eingesetzte XMPP-Serversoftware *ejabberd* (`https://www.ejabberd.im/`) installiert und gestartet. Anschließend wurde in der Konfigurationsdatei des Tor Proxys die erreichbaren Ports des Hidden Services eingetragen. Dabei wurden die Standard Ports (*5222* für Client-Server Verbindungen und *5269* für Server-Server Verbindungen) des Servers angegeben. Zuletzt musste nur noch der Tor Proxy und ein beliebiger XMPP Client, der das SOCKS Protokoll zur Namensauflösung unterstützt, auf den Client Rechnern installiert werden. Als Serveradresse wurde für die XMPP Clients die Onion Adresse des Hidden Services konfiguriert. Mit diesem Setup ist es schließlich möglich vollständig über Tor zu kommunizieren.
Abschließend kann folgendes Fazit gezogen werden: Die Anonymität und der sichere Transport von Nachrichten kann bei Instant Messaging mit Hilfe von Tor realisiert werden. In Kombination mit Instant Messaging Clients, die OTR unterstützen, kann die Sicherheit weiter erhöht werden, da OTR Ende-zu-Ende verschlüsselte Kommunikation ermöglicht. Der Einsatz von OTR ist vor allem dann sinnvoll, wenn dem Server nicht vertraut wird (z.B. unbekannter Server) oder potentielle Sicherheitslücken im Tor Netzwerk bestehen.

4 Tor-basierte Instant Messaging Clients

„*Privacy - like eating and breathing - is one of life's basic requirements.*“ , Katherine Neville.

In diesem Kapitel werden verwandte Arbeiten erläutert und daher verschiedene Instant Messaging Clients vorgestellt, die Tor als zugrundeliegendes Netzwerk für den Transport von Nachrichten und Präsenzinformationen verwenden.

4.1 TorChat

TorChat ist ein von Bernd Kreuss entwickelter Open Source Instant Messenger, welcher auf einem dezentralisierten Design basiert und Tor als Transportnetzwerk nutzt. Der Client wurde ursprünglich in Python entwickelt, aufgrund von besserer Erweiterbarkeit jedoch völlig neu in Free Pascal implementiert. TorChat bedingt keinerlei spezielle Voraussetzungen, um den Client auszuführen. Dies hat den Grund, dass TorChat gemeinsam mit dem Tor Proxy Client ausgeliefert wird. Der Client muss lediglich installiert werden und konfiguriert automatisch ohne Zutun des/der Benutzers/Benutzerin einen Hidden Service, der als Identifier für die Kommunikation mit anderen Personen dient. Darüber hinaus bietet TorChat eine portable Lösung an, sodass der Client auf Speichermedien wie USB-Sticks transportiert und auf beliebigen Computern ausgeführt werden kann. In diesem Fall wird jedoch empfohlen, das Speichermedium zu verschlüsseln, sodass beim Verlust des Mediums der private Schlüssel des Hidden Services sichergestellt ist.[27]

Die Sicherheit des Transports und der Inhalte von Nachrichten wird ausschließlich mit Hilfe der Tor Technologie gewährleistet. TorChat verzichtet somit auf zusätzliche Sicherheitsschichten wie OTR. Tor hat keinen dedizierten Mechanismus zur Verifikation der Authentizität von eingehenden Verbindungen vorgesehen, weshalb TorChat eine eigens implementierte Call Back Methode verwendet. Ausgehenden Verbindungen zu Hidden Services kann immer vertraut werden, da dies vom Tor Protokoll sichergestellt wird. Um auch eingehende Verbindungen zu verifizieren, wird immer eine zusätzliche ausgehende

Verbindung zum vermeintlichen Kontakt aufgebaut. Beide Kontakte senden anschließend zufällige Cookies über die jeweiligen ausgehenden Verbindungen aus. Wenn ein Kontakt dasselbe Cookie über die eingehende Verbindung empfängt, wurde die Authentizität erfolgreich überprüft, weshalb nun dieser eingehenden Verbindung vertraut werden kann.[27]

Der Client ist sehr schlicht was den Funktionsumfang und die grafische Benutzeroberfläche (siehe Abbildung 4.1) betrifft. Prinzipiell können nur Nachrichten ausgetauscht werden und es gibt die Möglichkeit, Dateien über den Client und somit über das Tor Netzwerk zu transferieren. Gruppenchats, eine der wichtigsten Funktionen von Instant Messaging, werden hingegen nicht unterstützt.

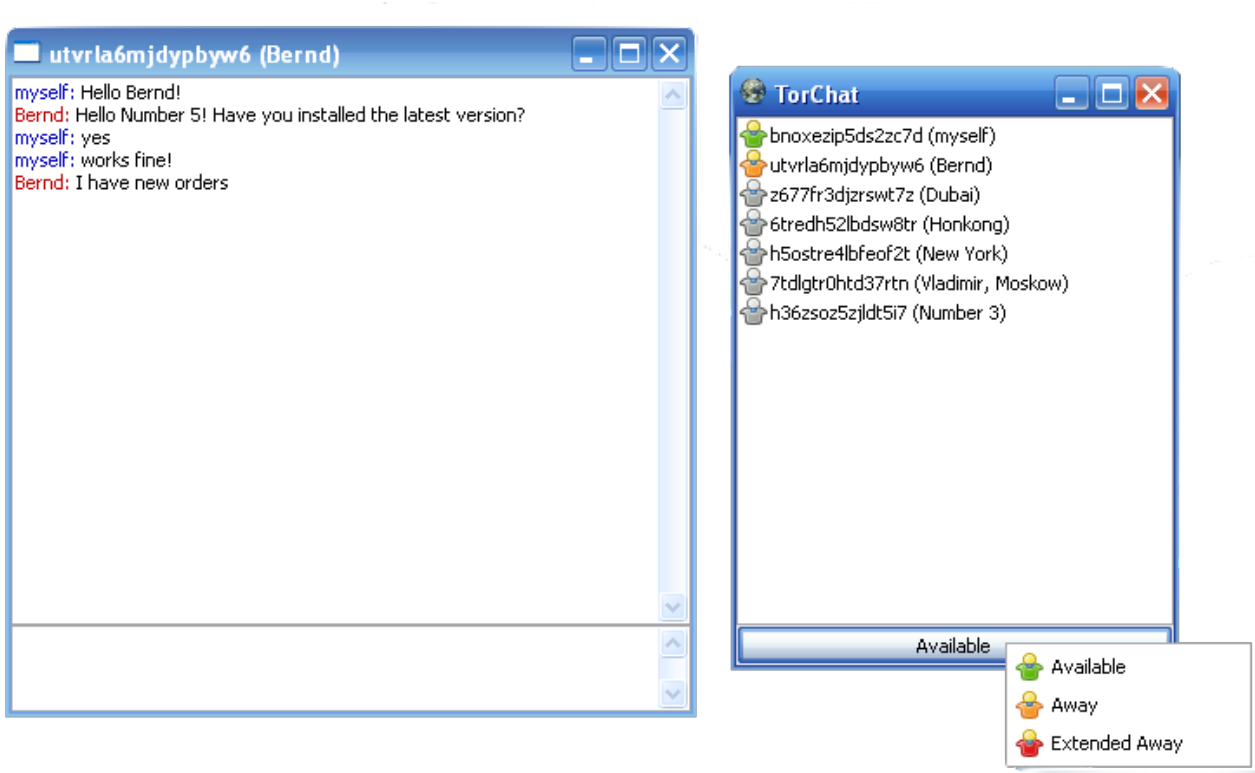


Abbildung 4.1: Grafische Oberfläche von TorChat[27]

TorChat ist für Windows und Linux verfügbar und die aktuellste Version *0.9.9* kann seit September 2012 von der Projektseite auf GitHub bezogen werden (`https://github.com/prof7bit/TorChat/downloads`). Der Client wurde seitdem nicht mehr weiterentwickelt, da der Entwickler den Fokus auf andere Projekte gerichtet hat. Dieser Client ist vom Funktionsumfang und von den unterstützten Plattformen betrachtet sehr eingeschränkt, weshalb andere plattformunabhängige IM Clients mit mehr Funktionen empfohlen werden. [27]

4.2 Ricochet

Ricochet (ehemals: *Torsion*) ist ein Peer-to-Peer Instant Messenger auf Basis von Tor Hidden Services. Hauptentwickler und Erfinder dieses Clients ist John Brooks, der beabsichtigte einen sicheren Messenger für anonyme Kommunikation zu entwickeln. Da ein zentrales Client-Server Modell vermieden wurde, ist jeder/jede Ricochet Anwender/in direkt mit seinen/ihren Kontakten verbunden. Wie bei TorChat kümmert sich auch Ricochet automatisch um die Konfiguration der Hidden Services, deren Onion Adressen schließlich zur Adressierung von Kontakten verwendet werden. Ricochet kann wie TorChat nach Installation ohne weitere Konfigurationen verwendet werden bzw. existiert auch eine portable Lösung, die keine Installation erfordert und nur ausgeführt werden muss.[28]
Der Transport von Sofortnachrichten erfolgt über ein binäres Transportprotokoll. Dieses Protokoll handhabt die Authentizität von eingehenden Verbindungen, im Gegensatz zum Protokoll von TorChat, etwas anders. Wenn eine Verbindung zu einem neuen Kontakt hergestellt werden soll, wird ein Request Paket geschickt, das folgende Daten beinhaltet:

- die Onion Addresse des/der Senders/Senderin,
- die Onion Addresse des/der Empfängers/Empfängerin,
- ein zufälliges Geheimnis, um normale Verbindungen (wenn der Kontakt bereits bekannt ist) zu authentifizieren,
- den öffentlichen Schlüssel des Hidden Services des/der Senders/Senderin,
- einen Nickname (optional) und
- eine digitale Signatur über das gesamte Request Paket

Der/Die Empfänger/in kann mit dem übermittelten öffentlichen Schlüssel die digitale Signatur und somit die Authentizität des/der Senders/Senderin überprüfen. Damit ist sichergestellt, dass der/die Sender/in wirklich den angegeben Hidden Service betreibt.[28]
Der Funktionsumfang von Ricochet ist sehr eingeschränkt, da ausschließlich Textnachrichten versendet werden können. Wie TorChat unterstützt Ricochet nur One-to-One Chats. Das bedeutet, dass keine Gruppen oder Chaträume erstellt werden können, um mit mehreren Personen gleichzeitig zu kommunizieren. Ricochet wird jedoch ständig weiterentwickelt und ist außerdem auf zahlreichen Plattformen (Windows, Linux, Mac OSX, ...) in unterschiedlichen Sprachen verfügbar. Die aktuelle Version *1.0.4* ist auf der Projektwebsite (`https://ricochet.im/releases/`) verfügbar.

Abbildung 4.2 zeigt die schlichte grafische Benutzeroberfläche von Ricochet unter Mac OSX:

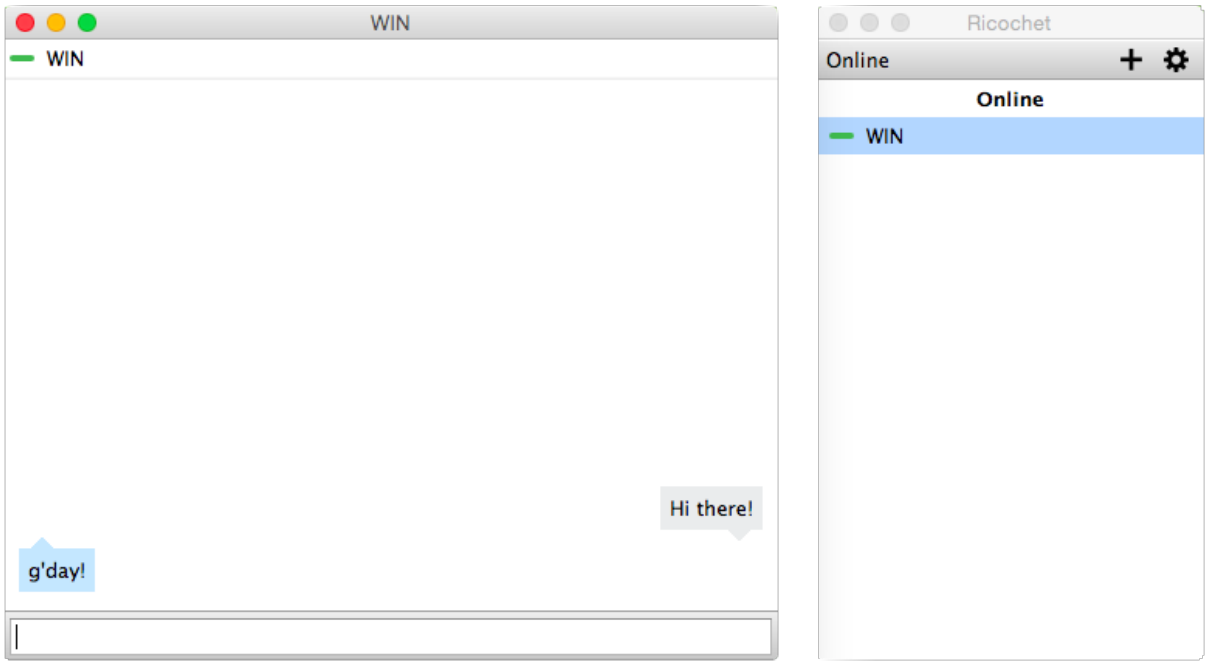


Abbildung 4.2: Grafische Oberfläche von Ricochet

Auch hier gilt: Wer ausschließlich sicher kommunizieren möchte, dem kann Ricochet durchaus empfohlen werden. Der Client wird aktiv weiterentwickelt und hat das Ziel sichere Kommunikation zu ermöglichen. Wer hingegen zusätzlich umfangreiche Instant Messaging Funktionen nutzen möchte, muss einen anderen IM Client verwenden.

4.3 Ergänzende nicht-dedizierte Tor IM Clients

IM Clients, die speziell für die Kommunikation über das Tor Netzwerk entwickelt werden, sind generell sehr rar. Zurzeit sind TorChat und Ricochet die einzigen größeren Projekte, die weitläufig bekannt sind. Aus diesem Grund sind in Tabelle 4.1 einige Clients zusammengefasst, die als sichere Alternativen eingesetzt werden können.

Im Unterschied zu TorChat und Ricochet können die nachfolgenden Clients auch ohne Tor verwendet werden. Da die Clients jedoch eine SOCKS Schnittstelle zur Verfügung stellen, können Onion Adressen aufgelöst und die Pakete der Transportprotokolle durch Tor getunnelt werden. Dies setzt jedoch die Installation und Konfiguration des Tor Proxy Clients voraus, was für nicht technisch-versierte Benutzer/innen eine Hürde darstellt.

Client	Protokoll	Lizenz	P2P	E2EE[1]	Plattform
Adium	XMPP	GPL	nein	OTR[2]	OS X
Cryptocat	XMPP	AGPL/GPL	nein	OTR	OS X, iOS, Chrome, Firefox, Safari, Opera
Jitsi	SIP/XMPP	LGPL	nein	OTR	Windows, Linux, OS X, Android
Pidgin	XMPP	GPL	nein	OTR[2]	Windows, Linux, OS X
qTox	Tox	GPL	ja	OTR	Windows, Linux, OS X, Android, iOS

Tabelle 4.1: Übersicht über Tor kompatible Instant Messaging Clients

[1] End-to-End Encryption
[2] optional

5 DChat - Eigenentwicklung eines IM Clients

„*The user's going to pick dancing pigs over security every time.*“ , Bruce Schneier.

Dieses Kapitel beschäftigt sich mit der Entwicklung eines Prototypen eines IM Clients für dezentrale, anonyme und sichere Kommunikation. Jeder Client ist dabei ausschließlich über das Tor Netzwerk zu erreichen. Ziel dieses Kapitels ist, die Voraussetzungen und die zu implementierenden technischen Schnittstellen zu evaluieren, um einen Tor-kompatiblen Prototypen zu entwickeln.

5.1 Was ist DChat?

DChat ist ein in C entwickelter Instant Messenger Prototyp mit dem Fokus auf Anonymität und kryptographische Sicherheit. In Kapitel 3 wurden verschiedene Szenarien erläutert, welche die mit Instant Messaging einhergehenden Gefahrenquellen am Beispiel XMPP verdeutlicht haben. Neben den Angriffsvektoren wurden jedoch auch verschiedene technologische Verfahren als Defensivmaßnahmen vorgestellt. Unter anderem wurden dabei die Eigenschaften und die Funktionsweise von Tor näher erläutert. Die Sicherstellung der Anonymität, als auch der sichere Transport von Nachrichten über Tor, bildeten die Entscheidungsgrundlage für die Verwendung dieser Technologie in DChat.
Jeder DChat Client läuft lokal auf dem System und setzt einen konfigurierten Hidden Service voraus, um mit anderen DChat-kompatiblen IM Clients kommunizieren zu können. Das in DChat eingesetzte Transportprotokoll verwendet zur Adressierung und Zustellung von Nachrichten ausschließlich Onion Addressen. Auf dieser Protokollebene werden somit niemals IP Adressen von Clients benötigt, die möglicherweise zur Aufhebung der Anonymität führen könnten. DChat Clients sind folglich ausschließlich über das Tor Netzwerk erreichbar.
Eine weitere Designentscheidung, die getroffen wurde, ist der Einsatz einer Peer-to-Peer Architektur (siehe Kapitel 2.3.2). DChat ist daher völlig dezentral ausgelegt und verzichtet somit auf dedizierte Server, die als Vermittler von Nachrichten zwischen mehreren Clients fungieren. Demzufolge kennt jeder Client

die Onion Adressen seiner Kontakte und führt eine direkte Verbindung zu jedem einzelnen Kontakt über das Tor Netzwerk. Abbildung 5.1 zeigt den Aufbau des DChat Peer-to-Peer Netzes:

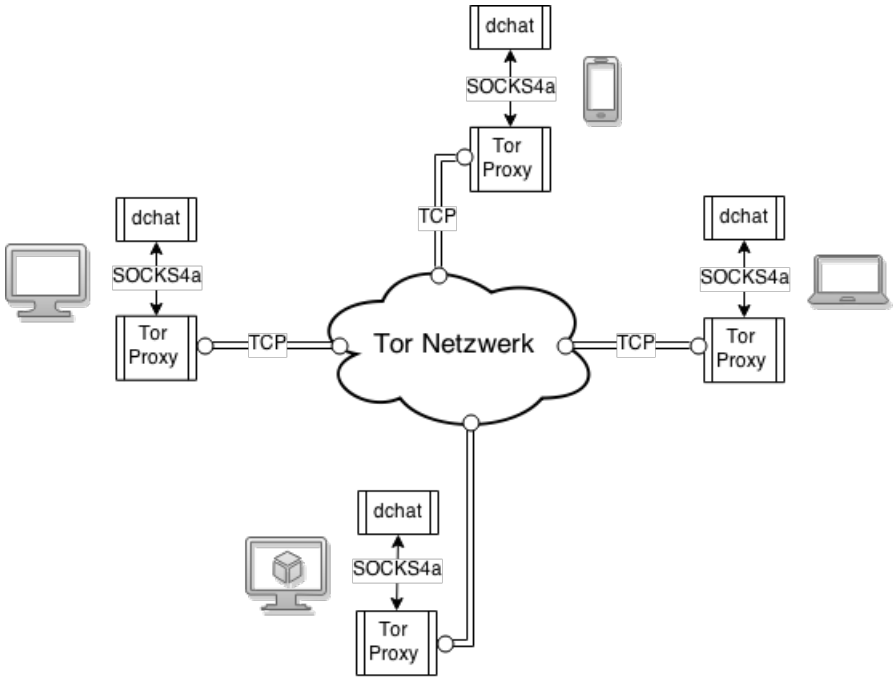


Abbildung 5.1: DChat Peer-to-Peer Netzwerk

Zusammengefasst ist DChat also ein Peer-to-Peer basierendes Instant Messaging System, das Tor als zugrundeliegendes Netzwerk verwendet.

5.1.1 Zielsetzung

Für den Entwurf und die Entwicklung von DChat wurden die folgenden Ziele definiert, die es während der Implementierungsphase immer wieder zu berücksichtigen galt:

- Sicherstellung der Anonymität
- Vertraulichkeit, Integrität und Authentizität von Nachrichten
- Full-Mesh IM Netzwerk als dezentrale P2P Architektur
- Lose Kopplung von Core Funktionalität und Graphical User Interface
- Portabilität (Stichwort: POSIX)
- Entwicklung eines einfachen, textbasierten Transportprotokolls
- Flexibilität bei der Erweiterung des Transportprotokolls

5.1.2 Voraussetzungen

Wie im einleitenden Teil dieses Kapitels bereits beschrieben wurde, transportiert DChat Nachrichten über das Tor Netzwerk. Aus diesem Grund benötigt jedes System, auf welchem DChat ausgeführt werden soll, Zugriff auf einen Tor Proxy. Für die korrekte Ausführung von DChat müssen grundsätzlich die folgenden beiden Voraussetzungen erfüllt werden:

1. **Vorhandensein eines Tor Proxy Clients**

   Auf `https://www.torproject.org/` kann der Tor Proxy Client für die verschiedensten Betriebssysteme heruntergeladen und anschließend installiert werden. Standardmäßig versucht DChat eine Verbindung zum System-lokalen Tor Client aufzubauen. Es ist allerdings auch möglich, einen von einem anderen Rechner zur Verfügung gestellten Tor Proxy Dienst zu verwenden. Aus Gründen der Sicherheit wird jedoch empfohlen, dass ausschließlich der System-lokale Tor Proxy verwendet werden sollte. Dies hat den Grund, dass DChat zwecks Einfachheit auf Ende-zu-Ende Verschlüsselung auf Applikationsebene (z.B. via OTR) verzichtet.

2. **Konfiguration eines Hidden Services**

   Eine weitere Voraussetzung, um mit anderen DChat Anwender/innen zu kommunizieren, ist die korrekte Konfiguration eines Hidden Services. Wie ein neuer Hidden Service eingerichtet wird, kann ebenfalls auf der Hompage des Tor Projekts nachgelesen werden. Da der DChat Client standardmäßig auf den Port *7777* auf *localhost* horcht, reicht jedoch prinzipiell die folgende Ergänzung in der Konfigurationsdatei *torrc* des Tor Clients auf einem Linux System aus:

```
HiddenServiceDir /var/lib/tor/hidden_service/
HiddenServicePort 7777 127.0.0.1:7777
```

Listing 5.1: DChat Hidden Service Konfiguration

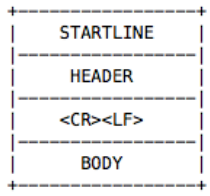
5.2 DChat Protokoll

Damit Clients untereinander Nachrichten und Daten austauschen können, müssen sie dieselbe „Sprache “ sprechen. Andernfalls würden sie sich nicht verstehen und die erhaltenen Nachrichten nicht interpretieren können. Aus diesem Grund werden Protokolle spezifiziert. IM Protokolle definieren beispielsweise die Struktur, das Format, die Adressierung, die Übertragungsart, usw. von Nachrichten. Die meisten IM

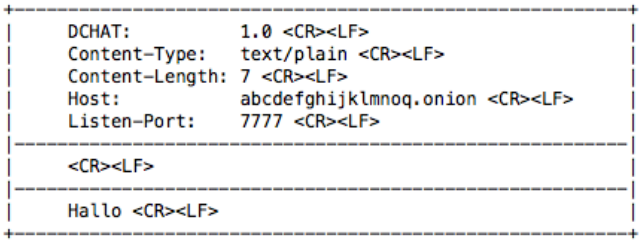
Protokolle sind relativ komplex zu implementieren, was folglich zu Portabilitätsproblemen aufgrund erhöhter Abhängigkeiten zu externen Bibliotheken führt. Deswegen wurde speziell für DChat ein sehr schlichtes, textbasiertes Protokoll entwickelt, welches nachfolgend näher beschrieben wird.

5.2.1 Format

Das DChat Protokoll ist ein textbasiertes IM Protokoll, dessen Nachrichtenformat sich an jenes von HTTP orientiert. Eine DChat Protocol Data Unit (PDU) beginnt mit einer Startzeile, welche die verwendete Protokollversion angibt (siehe Abschnitt 5.2.2). Anschließend folgt die Angabe aller Protokollheader. Diese werden zeilenweise angegeben, wobei jede Headerzeile mit den Steuerzeichen *Carriage Return (<CR>, '\r')* und *Line Feed (<LF>, '\n')* endet. Außerdem wird jede Headerzeile als Schlüssel-Wert Paar angegeben. Der Schlüssel definiert dabei immer den Namen des Headers und ist vom Wert mit einem Doppelpunkt getrennt. Das Ende der Headersektion einer PDU wird mit einer leeren Zeile signalisiert. Nach dieser Leerzeile folgt der letzte Teil einer PDU, der Payload bzw. dem Body. Die Struktur des Bodys ist abhängig vom Content-Type der PDU. Die implementierten Content-Typen werden später in diesem Abschnitt erläutert. Abbildung 5.2(a) veranschaulicht das definierte Nachrichtenformat und Abbildung 5.2(b) zeigt eine konstruierte beispielhafte DChat PDU, die in dieser Form von einem kompatiblen Client interpretiert werden kann.



(a) DChat PDU Format



(b) DChat PDU Beispiel

5.2.2 Header

Der Transport von Sofortnachrichten erfordert in DChat nur eine Handvoll spezifizierter Header. Diese werden nachfolgend genauer beschrieben. Jeder einzelne der folgenden Header müssen in einer PDU verpflichtend angegeben werden. Selbstverständlich werden auch optionale Header unterstützt, sodass DChat sehr leicht um neue Funktionen ergänzt werden kann. Auf die optionalen Header wird hier allerdings nicht näher eingegangen.

- **DCHAT**

  Dieser Header gibt die DChat Protokollversion einer PDU an. Da derzeit keine anderen Protokollversionen existieren, ist der Wert dieses Headers standardmäßig immer *1.0*. Wichtig ist, dass dieser Header **immer** den ersten Header einer PDU bildet, damit ein Client aufgrund der Versionsnummer bestimmen kann, ob er zu dieser Version kompatibel ist oder nicht.

- **Content-Type**

  Der Content-Type Header bestimmt den Typ der Daten, die im Body einer PDU übertragen werden. Zur der Zeit, in der die Arbeit verfasst wurde, werden zwei verschiedene Content-Typen unterstützt:

  - *text/plain*: Die im Body übertragenen Bytes sind als ASCII Zeichen zu interpretieren. Dieser Typ wird verwendet, um Sofortnachrichten zu transportieren.
  - *control/discover*: Im Body befinden sich alle Kontakte eines/einer Benutzers/Benutzerin, zu welchen eine aktive Verbindung besteht. Diese werden zeilenweise in der Form <*onion-addresse*> <*listen-port*> angegeben.

- **Content-Length**

  Die Content-Length definiert die Länge des Bodys einer PDU und wird in Bytes angegeben.

- **Host**

  Der Host Header definiert eine 16 Byte lange Onion-Adresse unter welcher der DChat Client einer erstellten PDU erreicht werden kann. Die Onion Adresse wird in diesem Header immer mit dem virtuellen TLD[1]-Suffix *.onion* angegeben.

- **Listen-Port**

  Der Listen-Port gibt an auf welchen lokalen Port ein DChat-Client horcht und Verbindungsanfragen entgegennimmt. Wenn nicht anders angegeben, verwendet DChat standardmäßig den Port 7777.

5.2.3 Kontaktaustausch

Der Aufbau des DChat Peer-to-Peer Netzes geschieht nach einem sehr einfachen Schema. Ein DChat Client, der eine Verbindung zu einem anderen Client aufbauen möchte, sendet diesem eine PDU mit dem

[1] Top Level Domain

Content-Type *control/discover*. Die PDU transportiert somit im Body alle bekannten Kontakte des Clients, wobei der Body auch leer sein kann, falls noch keine aktive Verbindung zu einem Kontakt besteht. Der andere Client sendet bei Annahme der Verbindung ebenfalls eine PDU mit den Kontaktinformationen aus, sodass beide Clients die Kontakte des jeweils anderen kennenlernen. Anschließend verwenden sie die neu gelernten Kontaktinformationen, um nach demselben Schema neue Verbindungen zu den neu erhaltenden Kontakten aufzubauen. Dieser Vorgang wiederholt sich solange, bis ein vollständiges Full-Mesh Netzwerk aufgebaut ist. Das bedeutet, dass jeder Client direkte Verbindungen zu allen anderen in diesem Netzwerk befindlichen Clients führt (siehe Abbildung 5.1).
In Grafik 5.2 wird der Vorgang des Verbindungsaufbaus und des Kontaktaustauschs abgebildet:

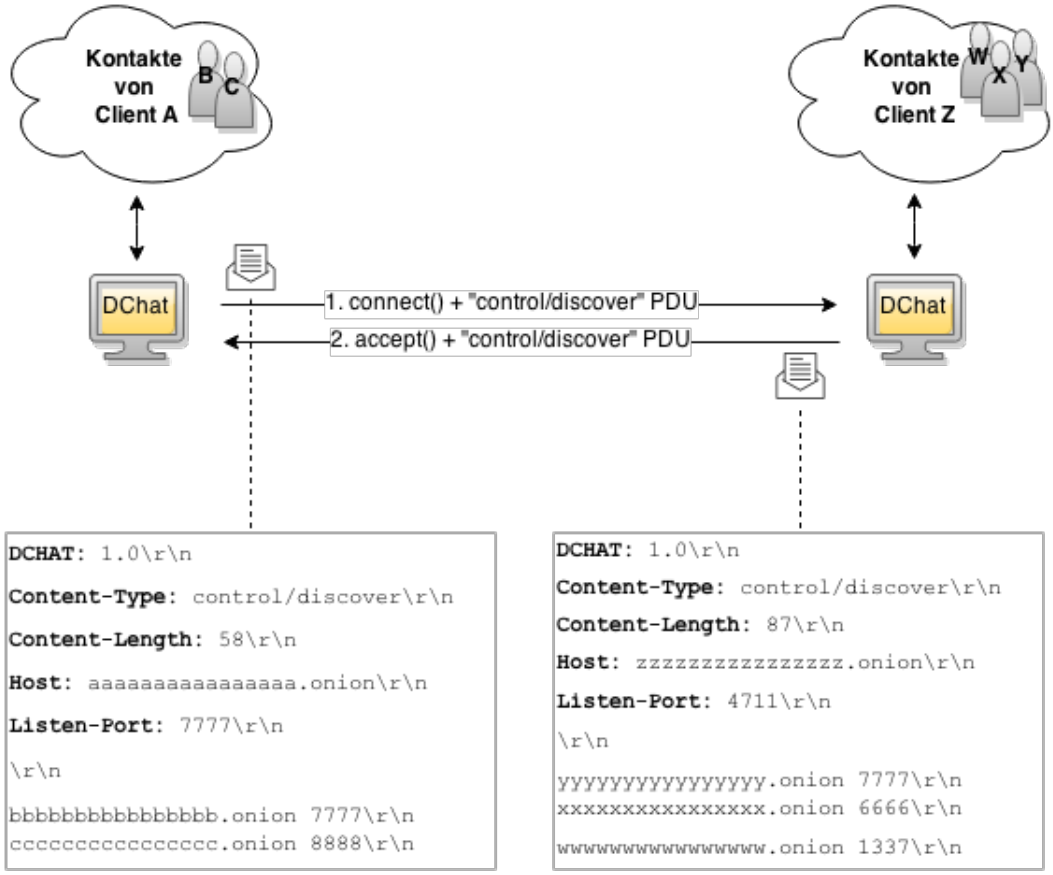


Abbildung 5.2: DChat Kontaktaustausch

Bevor Client A sich zu Client Z verbindet, existieren zwei voneinander getrennte Peer-to-Peer Netzwerke: Netzwerk von Client A (Client A, Client B, Client C) und Netzwerk von Client Z (Client W, Client X, Client Y, Client Z). Nachdem die Kontakte während des Verbindungsaufbaus untereinander ausgetauscht wurden, verschmelzen die beiden Netzwerke zu einem großen Peer-to-Peer Netzwerk.

Duplicate Detection

Leider führt der Mechanismus des Kontaktaustauschs technologisch bedingt zu einem Problem. Während des Kontaktaustauschs kann es passieren, dass die Kontaktlisten von Clients möglicherweise Duplikate enthalten. Dies tritt dann auf, wenn zwei Clients zur selben Zeit Kontaktinformationen des jeweiligen anderen Clients erhalten. Dieser Umstand führt wiederum unweigerlich dazu, dass die beiden betroffenen Clients gegenseitig eine Verbindung zueinander aufbauen, weshalb jeder Client den anderen Client zweimal als Kontakt in die Kontaktliste aufnimmt. Einmal aufgrund der Verbindungsanfrage und einmal aufgrund der Verbindungsannahme. Man benötigt deshalb einen Mechanismus, durch dessen Hilfe sich beide Clients darauf abstimmen können, welches Duplikat gelöscht werden soll, um das Peer-to-Peer Netzwerk aufrecht zu erhalten.

DChat implementiert zu diesem Zweck eine Duplicate Detection, die dafür sorgt, dass jeder Kontakt nur ein einziges Mal in die Kontaktliste aufgenommen wird. Dazu werden die Onion Addressen und die Listening Ports miteinander verglichen. Sollte sich also ein Kontakt zweimal in der Kontaktliste befinden, wird eine der beiden Kontaktstrukturen entfernt, indem der entsprechende TCP Socket geschlossen wird. Der Client mit der alphanumerisch höheren Onion Adresse und höheren Listening Port entfernt dabei jenen Kontakt, der aufgrund einer Verbindungsanfrage (vgl. *connect()* Systemcall) in die Kontaktliste aufgenommen wurde. Der andere Client mit der alphanumerisch niedrigeren Onion Adresse und niedrigeren Listening Port entfernt hingegen den Kontakt, der durch einer Verbindungsannahme (vgl. *accept()* Systemcall) hinzugefügt wurde.

Der folgende Pseudocode zeigt, wie der Duplicate Detection Mechanismus umgesetzt werden könnte. In dieser Form wurde er auch in DChat implementiert:

```
function check_duplicates(contact self, contact n){
    contact connect_contact
    contact accept_contact;

    //retrieve the two places of contact n from the contactlist (if n is a duplicate)
    places[2] = find_duplicates_in_contactlist(n);

    //check how the clients got into this contactlist - through a "connect" or an "accept"?
    if(
          get_contact_from_contactlist(places[0]).connect_type == CONTACT_CONNECT
      ){
       connect_contact = get_contact_from_contactlist(places[0]);
```

```
        accept_contact = get_contact_from_contactlist(places[1]);
    }
    else{
        accept_contact = get_contact_from_contactlist(places[0]);
        connect_contact = get_contact_from_contactlist(places[1]);
    }

    //if your onion address is greater than the onion address of the duplicated contact ->
    //delete those contact in your clientlist that has been added because of a "connect"
    if(self.onion_addr > n.onion_addr){
        close(connect_contact.fd);
        delete(connect_contact);
    }
    //otherwise delete the contact that has been added because of an "accept"
    else if(self.onion_addr < n.onion_addr){
        close(accept_contact.fd);
        delete(accept_contact);
     }
    //if the onion addresses are the same - check the listening port and apply the same
        principle
    else if(self.onion_addr == n.onion_addr){
        if(self.lport > n.lport){
            close(connect_contact.fd);
            delete(connect_contact);
        }
        else if(self.lport < n.lport){
            close(accept_contact.fd);
            delete(accept_contact);
        }
        // if onion addresses and listening ports are equal, any duplicate
        // ("connect" or "accept" contact) can be deleted
        else{
            close(connect_contact);
            delete(connect_contact);
        }
    }
}
```

Listing 5.2: Duplicate Detection Pseudocode

5.3 Implementierung

Für die Umsetzung von DChat wurden viele anfängliche Schwierigkeiten überwunden. Beispielsweise hat die Implementierung eines korrekten Protokollstacks (Kontaktaustausch, Duplicate Detection, etc.) sehr viel Zeit in Anspruch genommen. Auch die Definition der grafischen Benutzerschnittstelle war äußerst anspruchsvoll, sodass die grafische Oberfläche zur Kernfunktionalität lose gekoppelt ist. Eine weitere Hürde stellte anfangs auch die Implementierung der Schnittstelle zum Tor Proxy Client dar, da das Tor Projekt nur sehr wenig diesbezüglich dokumentiert hat. In den nachfolgenden beiden Abschnitten wird die Architektur des DChat Clients beschrieben und die Schnittstellen näher vorgestellt.

5.3.1 Architektur

Die DChat Kernfunktionalität besteht aus vier Hauptmodulen, die als Threads implementiert sind und über Interfaces mit externen Komponenten kommunizieren. Diese umfassen:

- **UI Handler**

  Dieses Modul ist das Hauptmodul und definiert mit Hilfe von Unix Sockets eine Schnittstelle für grafische Benutzeroberflächen. Die lokalen Unix Sockets werden dazu verwendet, um Ein- und Ausgaben zwischen grafischer Benutzeroberfläche und DChat zu transportieren. Dies ermöglicht eine lose Kopplung der Benutzeroberfläche zur eigentlichen DChat Kernfunktionalität. Eingaben eines/einer Benutzers/Benutzerin werden von diesem Modul zuerst interpretiert. Handelt es sich um einen gültigen In-Chat Befehl wird dieser ausgeführt (z.B. um Verbindung zu neuen Kontakt herzustellen), andernfalls wird die erhaltene Eingabe als Text interpretiert und als Sofortnachricht an alle verfügbaren Kontakte verschickt.

- **Socket Handler**

  Der Socket Handler prüft, ob neue Daten verfügbar sind, die von den Sockets der Kontakte gelesen werden können. Falls das zutrifft, wird von dem entsprechenden Kontaktsocket eine PDU Einheit eingelesen und diese je nach Content-Type entsprechend interpretiert. Handelt es sich beispielsweise um eine Sofortnachricht (Content-Type: *text/plain*) wird diese an den UI Handler weitergereicht, der diese wiederum zur Darstellung an die grafische Benutzeroberfläche weiterleitet. Eine PDU mit dem Content-Type *control/discover* initiiert hingegen einen Kontaktaustausch. Dazu werden die erhaltenen Kontaktinformationen an den Connect Handler weitergereicht, der eine

neue Verbindung zu jedem einzelnen Kontakt aufbaut und diesen gegebenenfalls der Kontaktliste hinzufügt (Siehe 5.2.3).

- **Connect Handler**

  Der Connect Handler ist dafür verantwortlich, neue Verbindungen herzustellen und die Kontaktliste zu erweitern. Jeder Verbindungsaufbau zu einem neuen Kontakt wird in einem eigenen Thread abgehandelt, sodass eine parallele Abarbeitung von Kontaktinformationen möglich ist. Würde dies nicht in dieser Form geschehen, hätte es lange Verzögerungen und Wartezeiten zur Folge bis alle Verbindungen aufgebaut sind und das Peer-to-Peer Netzwerk etabliert ist.

- **Accept Handler**

  Der Accept Handler ist das Gegenstück zum Connect Handler. Dieses Modul ist dafür verantwortlich, neue Verbindungen von etwaigen Kontakten anzunehmen und diese gegebenenfalls nach Überprüfung von Duplikaten der Kontaktliste hinzuzufügen.

Die soeben beschriebenen Module werden in Abbildung 5.3 übersichtlich dargestellt. Auf dieser Abbildung ist zu sehen, wie die Module zusammenhängen und welche Schnittstellen sie für die Kommunikation mit externen Komponenten verwenden.

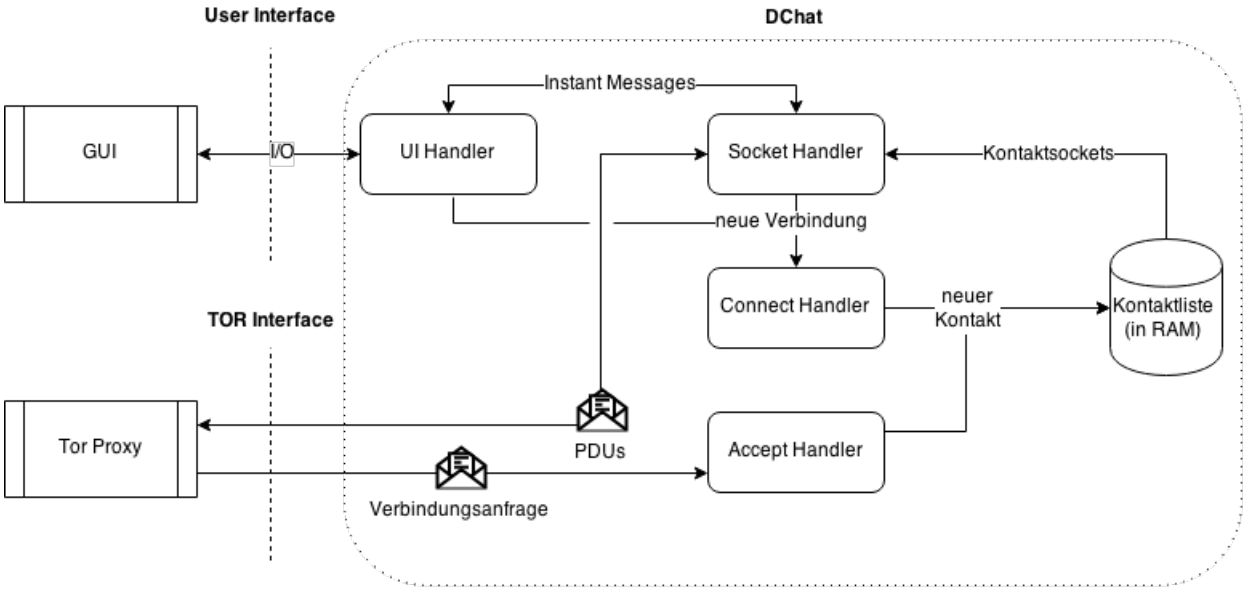


Abbildung 5.3: DChat Implementierungsarchitektur

5.3.2 Schnittstellen

Um Ein- und Ausgaben auf der grafischen Benutzeroberfläche darzustellen und um mit anderen Clients über das Tor Netzwerk kommunizieren zu können, sind Schnittstellendefinitionen erforderlich. DChat selbst bietet mit einem speziellen User Interface die Möglichkeit, verschiedenste Benutzeroberflächen an DChat zu koppeln. Diese Schnittstelle hat den Vorteil, dass Oberflächen mit beliebigen Frameworks in beliebigen Programmiersprachen entwickelt werden können. Um die Tor Technologie verwenden zu können, musste in DChat eine Schnittstelle implementiert werden, welche die Kommunikation mit dem Tor Proxy ermöglicht. Nachfolgend werden diese beiden Schnittstellen näher beschrieben.

Tor Interface

Der Tor-Proxy ist unter anderem dafür verantwortlich, virtuelle Onion URLs in IP Adressen aufzulösen. Dazu muss mit ihm über eine definierte Schnittstelle gesprochen werden. Tor unterstützt dabei verschiedene Protokollstandards (HTTP, SOCKS4, SOCKS4A und SOCKS5), die als generische Schnittstellen dienen.[29]

Im Fall von DChat wird über die SOCKS4A Schnittstelle mit dem Tor Proxy kommuniziert. Dazu wurde eine Java basierte Library (siehe `http://web.mit.edu/foley/www/TinFoil/src/tinfoil/TorLib.java`) zu einer C Library portiert. Eine SOCKS4 bzw. SOCKS4A PDU hat folgenden Aufbau[30]:

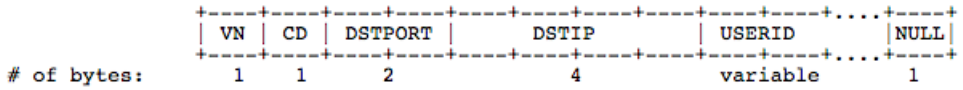


Abbildung 5.4: SOCKS4 Format[30]

Das erste Byte *VN* gibt die verwendete SOCKS Version an. Das zweite Byte gibt die Operation an, die der SOCKS Server durchführen soll. In DChat wird die *CONNECT* Operation verwendet, sodass der Tor Proxy die in dieser PDU enthaltene Onion URL auflöst und einen Circuit zum Hidden Service aufgebaut. Die nächsten sechs Bytes geben sowohl den Port, als auch die IP-Adresse des Ziels an. Wenn die Ziel-IP-Adresse unbekannt ist, definiert das SOCKS4A Protokoll, dass die ersten drei Bytes der Adresse den Wert Null und das vierte und letzte Byte einen Wert ungleich Null besitzen müssen. Nach der IP Adresse folgt eine variable Anzahl an Bytes für die *User-ID*, die mit einem dedizierten *NULL* Byte terminiert wird. Die User-ID ist hierbei als Domänenname zu verstehen. Ein SOCKS Server versucht diese in eine

IP Adresse aufzulösen, sobald er eine SOCKS4A PDU erhält. Da bei DChat die Ziel-IP-Adresse immer unbekannt ist, wird für den Verbindungsaufbau zu einem Kontakt eine SOCKS4A PDU erstellt, deren User-ID die Onion URL des entsprechenden Kontakts repräsentiert. Wenn der Tor Proxy diese PDU erhält, extrahiert dieser die Onion Adresse (=User-ID) des Hidden Services und baut zu diesem einen neuen Circuit auf.[30][31]

Nachfolgend werden nun die wichtigsten Codeausschnitte erklärt, die für die Implementierung der Tor Schnittstelle in DChat eine essentielle Rolle spielten. Aus Gründen der Übersichtlichkeit wurden die Überprüfungen der Return-Werte entfernt.

Die folgenden Präprozessorkonstanten wurden definiert, um die Erstellung von SOCKS4A PDUs zu vereinfachen. Diese Konstanten werden für den Aufbau eines Tor Sockets benötigt:

```
#define SOCKS_VERSION              0x04
#define SOCKS_CMD_CONNECT          0x01
#define SOCKS_DESTIP_NULL          0x00
#define SOCKS_DESTIP_NONE_NULL     0x01
```

Listing 5.3: SOCKS4A Präprozessorkonstanten

Der nächste Codeausschnitt implementiert zwei Funktionen, die zum Lesen und Schreiben von SOCKS4A PDUs verwendet werden. Beide Funktionen bekommen sowohl einen Socket Deskriptor, als auch eine SOCKS4A Struktur übergeben. Die erste Funktion liest vom Socket Deskriptor und speichert die gelesenen Bytes in die übergebene Struktur. Die andere Funktion schreibt die SOCKS4A Struktur Byte für Byte in den Socket Deskriptor. DChat benötigt diese beiden Funktionen um mit dem Tor Proxy über die SOCKS Schnittstelle kommunizieren zu können.

```
int
read_socks4a(int s, socks4a_pdu_t* pdu)
{
    int ret;

    // read SOCKS4a response (see SOCKS4a protocol)
    read(s, pdu, 8);
    pdu->port   = ntohs(pdu->port);      //convert to host byte order
    pdu->fakeip = ntohl(pdu->fakeip); // convert to host byte order

    return ret;
}
```

```
int
write_socks4a(int s, socks4a_pdu_t* pdu)
{
    // convert ip and port to network byte order
    uint16_t rport  = htons(pdu->port);
    uint32_t fakeip = htonl(pdu->fakeip);

    write(s, &pdu->version, 1);
    write(s, &pdu->command, 1);
    write(s, &rport, 2);
    write(s, &fakeip, 4);
    write(s, &pdu->delim, 1);
    write(s, pdu->hostname, strlen(pdu->hostname));
    write(s, &pdu->delim, 1);

    return 0;
}
```

Listing 5.4: Lesen / Schreiben einer SOCKS4A PDU

Die vermutlich wichtigste Funktion ist *create_tor_socket*. Diese Funktion verwendet die beiden zuvor beschriebenen Funktionen, um dem Tor Proxy darüber in Kenntnis zu setzen, einen Circuit zu einem Hidden Service aufzubauen. Die Onion URL als auch der Port des Hidden Services wird dabei als Parameter übergeben. Zuerst wird ein neuer TCP Socket für die Verbindung zum System-lokalen Tor Proxy erzeugt. Danach wird eine SOCKS4A PDU mit der gewünschten Onion URL und Port erstellt. Diese PDU wird schließlich dem Proxy übermittelt, welcher wiederum prüft, ob die angegebene Onion URL überhaupt existiert. Falls sie existiert, wird ein Circuit zu diesem Hidden Service aufgebaut. Letztendlich retourniert der Tor Proxy eine SOCKS4A PDU, die einen Statuscode über Erfolg oder Misserfolg des Aufbaus des Circuits beinhaltet. Wenn dies erfolgreich verlaufen ist, wird der TCP Socket Deskriptor an die aufrufende Funktion retourniert. Der Socket kann schließlich für die Kommunikation mit dem entfernten Rechner verwendet werden, wobei diese mit Hilfe der Tor Proxy Clients über das Tor Netzwerk geführt wird.

```
int
create_tor_socket(char* hostname, uint16_t rport)
{
    int s; // tor socket
    struct sockaddr_in da; // destination address to connect to
```

```
    socks4a_pdu_t pdu; // SOCKS request
    int ret;
    memset(&da, 0, sizeof(da));

    // ip address for connection to the local Tor client
    inet_pton(AF_INET, "127.0.0.1", &da.sin_addr);
    da.sin_family = AF_INET;
    da.sin_port = htons(9050); //Tor SOCKS Proxy Port (9050 is default)

    // connect to Tor client
    s = connect_to((struct sockaddr*) &da);

    // craft SOCKS request pdu
    memset(&pdu, 0, sizeof(pdu));
    pdu.version = SOCKS_VERSION;
    pdu.command = SOCKS_CMD_CONNECT;
    pdu.port = rport;
    pdu.fakeip = SOCKS_DESTIP_NONE_NULL;
    pdu.delim = SOCKS_DESTIP_NULL;
    pdu.hostname = hostname;

    // send connection request to Tor Proxy
   write_socks4a(s, &pdu);

    // read response code from Tor Proxy
    memset(&pdu, 0, sizeof(pdu));
    read_socks4a(s, &pdu));

    // SOCKS4A Response Code 90 means 'OK'
    if (pdu.command != 90)
    {
        return -1;
    }

    return s;
}
```

Listing 5.5: Tor Socket

User Interface

DChat wurde so entworfen, dass es unabhängig von grafischen Benutzeroberflächen ist. Der DChat Client selbst ist im Prinzip nur ein Dienst, der im Hintergrund ausgeführt wird. Es wird jedoch eine Schnitt-

stelle bereitgestellt, die die Kommunikation zu grafischen Benutzeroberflächen zulässt. Dies hat den Vorteil, dass Entwickler/innen beliebige Oberflächen für DChat entwickeln können. Im Zuge dieser Arbeit wurde mit Hilfe der *ncurses* Bibliothek eine Konsolen-basierte Oberfläche implementiert, sodass DChat in einem Terminal verwendet werden kann.

Das DChat User Interface ist sehr einfach aufgebaut, da zur Kommunikation mit dem Client UNIX Domain Sockets eingesetzt werden. UNIX Domain Sockets werden für die bi-direktionale Kommunikation zwischen Prozessen (Inter-Process-Communication) verwendet. Ein Socket dieser Art bindet sich an einen willkürlichen und eindeutigen Dateipfad im System, wo er auf Verbindungsanfragen wartet. Im Dateisystem repräsentiert ein UNIX Domain Socket eine spezielle Datei. Ein lokaler Prozess kann sich nun zu diesem Socket verbinden, wobei die Adresse dem Dateipfad entspricht. Diese Vorgehensweise ermöglicht es, dass zwei voneinander unabhängige Prozesse miteinander kommunizieren können.

DChat definiert drei UNIX Domain Sockets mit unterschiedlichen Funktionen:

- **I/O Socket**

  Dieser Socket bindet sich standardmäßig an den Pfad */usr/local/var/run/dchat_io.sock* und ist für Ein- und Ausgaben zuständig. Grafische Oberflächen können an diesen Socket Benutzereingaben schicken und DChat wird diese anschließend als Sofortnachrichten an alle verfügbaren Kontakte schicken. Sofortnachrichten, die DChat hingegen von Kontakten erhält, werden wiederum über den Socket an die Oberfläche weitergeleitet. Diese stellt die Nachrichten wiederum für den/die Benutzer/in dar. Die Ein- und Ausgaben, die über diesen Socket geleitet werden, werden dabei im Format *<Nickname>;<Nachrichtentext>;* übertragen.

- **Log Socket**

  Dieser Socket wird nicht bi-direktional sondern uni-direktional verwendet und bindet sich an den Pfad */usr/local/var/ran/dchat_log.sock*. DChat schreibt Lognachrichten mit zugehörigem Loglevel im Format *<Loglevel>;<Logtext>;* auf diesen Socket. Die Oberfläche kann diese Meldungen schließlich darstellen. DChat versendet beispielsweise eine Lognachricht, wenn sich ein neuer Kontakt verbunden hat. Als Loglevel werden die Syslog Loglevel (Emergency, Alert, Critical, ...) herangezogen.

- **Command Socket**

  Der Command Socket wird für die Ausführung von In-Chat Befehlen (z.B. Verbindung zu neuem Kontakt herstellen) verwendet und bindet sich an den Pfad */usr/local/var/run/dchat_cmd.sock*.

Dieser Socket wird wiederum bi-direktional verwendet. Die Oberfläche übermittelt DChat den auszuführenden Befehl in der Form *<Befehl>;<Parameter 1>;<Parameter 2>; ... ;<Parameter n>;*. Wenn DChat eine derartige Zeile erhalten hat, wird die Zeile Token für Token geparst und in einen Befehl konvertiert. Wenn der Befehl unterstützt wird und kein Syntaxfehler aufgetreten ist, wird dieser schließlich ausgeführt und es wird ein Erfolgsstatus auf dem Command Socket retourniert. Andernfalls wird ein Fehlerstatus retourniert und eine detaillierte Fehlermeldung (z.B. wo der Syntaxfehler aufgetreten ist) auf den Log Socket geschrieben.

5.4 Ergebnis

Das Endresultat von DChat wird auf den nächsten beiden Abbildern gezeigt. Abbildung 5.5 zeigt die Ausführung des DChat Kerns. Wie anhand der übergebenen Commandline Optionen zu erkennen ist, ist dieser Client unter der Onion Adresse *5qnnu3wrfx3dzyog.onion* und Port *7777* erreichbar. Darüber hinaus ist darauf die Initialisierung des User Interfaces zu sehen und man erkennt, dass sich eine grafische Oberfläche verbunden hat. Zusätzlich sind die über das User Interface empfangenen Benutzereingaben zu Testzwecken dargestellt.

Abbildung 5.5: DChat Core

In Abbildung 5.6 ist hingegen eine von DChat unabhängige grafische Benutzeroberfläche zu sehen. Die-

se wurde mit Hilfe der *ncurses* Bibliothek implementiert und ist für den Einsatz in der Konsole gedacht. Zum einen werden auf dieser Abbildung die Benutzereingaben gezeigt, die DChat auf *stdout* geloggt hat und zum anderen wird versucht, eine Verbindung zu der Onion Adresse *ytwzcgrrgadvwqtu.onion* aufzubauen. Der Tor Circuit wurde zum Client (*LIN*) erfolgreich aufgebaut und eine Sofortnachricht wurde von diesem Client ebenfalls empfangen und dargestellt. Weiters ist auf dieser Abbildung zu erkennen, dass der Duplicate Detection Mechanismus angewandt wurde. Client *LIN* hat offenbar versucht sich zu verbinden. Die Duplicate Detection hat jedoch erkannt, dass sich *LIN* bereits in der Kontaktliste befindet. Aus diesem Grund wurde die Verbindung getrennt und das Duplikat entfernt.

Abbildung 5.6: DChat GUI

Die in Kapitel 5.1.1 gesetzten Ziele wurden allesamt erreicht, denn es wurde ein einfacher IM Client Prototyp für sichere und anonyme Kommunikation entwickelt. Da unter anderem die Portabilität ein wichtiges Ziel darstellte, wurde DChat so portabel wie möglich implementiert. DChat ist daher auf unterschiedlichen Betriebssystemen lauffähig, wobei die Anwendung auf den folgenden Systemen getestet wurde:

- Mac OS X 10.9 (Mavericks) / Mac OS X 10.10 (Yosemite)
- Ubuntu 14.04 (Trusty Tahr)

- Debian 7.8 (Wheezy)
- Windows 7 / Windows 8[2]

Auch die Portierung auf ein mobiles Betriebssystem wie Android sollte keine zu große Hürde darstellen, da ein Tor Proxy Client für Android existiert und auf diesem System native C Programme ausgeführt werden können. Aufgrund zeitlicher Einschränkungen wurde jedoch auf eine Portierung verzichtet.
Im Unterschied zu TorChat und Ricochet sind mit DChat Multi-User und One-to-One Chats möglich. Auch die Verwendung eines textbasierten Protokolls bietet mehre Vorteile gegenüber den binären Protokollen, die in anderen Clients verwendet werden. Textbasierte Protokolle verbessern das Troubleshooting von Fehlern. Außerdem ermöglichen sie dynamische PDU Formate, die sehr leicht eingelesen und mit einem Parser in andere Darstellungsformen transformiert werden können. DChat bietet gegenüber TorChat und Riccochet zusätzlich den Vorteil, dass die definierte Benutzerschnittstelle unterschiedliche grafische Benutzeroberflächen ermöglicht. Unter DChat fehlt hingegen jegliche Verifizierung der Authentizität eines Hidden Services. Um Spoofing Attacken zu verhindern, müssen zukünftige DChat Versionen einen ähnlichen Mechanismus wie TorChat oder Ricochet für die Verifizierung von eingehenden Verbindungen implementieren. Der Autor dieser Arbeit präferiert dabei den Ansatz von Ricochet, da dieser leichter zu implementieren ist und zudem die elegantere Variante darstellt.
Abschließend kann gesagt werden, dass DChat nicht darauf abzielt, einen Ersatz für andere Protokolle bzw. IM Clients zu bieten. Wer sich jedoch mit simplen Basisfunktionen zufrieden gibt, um schnell und sicher Textnachrichten zu verschicken, kann durchaus auf DChat zurückgreifen. Die Installation als auch Konfiguration geht schnell und einfach vonstatten, es läuft auf sehr vielen Systemen und ermöglicht sichere und anonyme Kommunikation mit anderen Personen. Außerdem kann das DChat Transportprotokoll sehr leicht um neue Funktionen ergänzt werden, sodass zukünftige DChat Versionen Features wie Filesharing unterstützen könnten.

[2] siehe Cygwin (`https://www.cygwin.com/`)

6 Zusammenfassung und Ausblick

Instant Messaging birgt für Anwender/innen viele Risiken. Ob Malware, Schwachstellen im Client oder Information Disclosure, all dies sind potentielle Gefahren, die einem beim Instant Messaging bewusst sein müssen. Generell ergeben sich für einen/eine Angreifer/in beim Transport von Nachrichten mehrere Angriffsvektoren, wobei prinzipielle zwei Szenarien betrachtet werden können. In Szenario 1 befindet sich der/die Angreifer/in zwischen Client und Server und kann folglich den Datenverkehr dieser beiden Instanzen mitlesen und manipulieren. In Szenario 2 hat der/die Angreifer/in Zugriff auf den Datenverkehr, der zwischen den Servern transportiert wird. Um diesem Problem entgegenzuwirken, sodass Nachrichten zwischen Clients sicher transportiert werden, ist der Einsatz einer Technologie für Ende-zu-Ende Verschlüsselung erforderlich.

In dieser Arbeit wurden verschiedene Lösungen für Ende-zu-Ende Verschlüsselungen von Instant Messages vorgestellt und auf ihre Praxistauglichkeit überprüft, darunter PGP, OTR und Tor. All diese Technologien wurden für unterschiedliche Anwendungszwecke konzipiert, sodass nicht jede für Instant Messaging geeignet ist. PGP wurde beispielsweise im besonderen für den sicheren Transport von E-Mails entwickelt. OTR ist daher PGP vorzuziehen, da dieses Protokoll im Gegensatz zu PGP wichtige Eigenschaften wie Perfect Forward Secrecy, Malleable Encryption und Plausible Deniability aufweist.

Bei Ende-zu-Ende Verschlüsselung werden technisch-bedingt nur die Payload einer Nachricht verschlüsselt, jedoch niemals die für die Zustellung notwendigen Metadaten. Es ist daher möglich zu bestimmen, wann jemand online ist und mit welchen Personen kommuniziert wird. Diese Verbindungsdaten können möglicherweise vor einem Gericht als Entscheidungsgrundlage dienen. Der Einsatz von Tor ist somit empfehlenswert. Diese Technologie anonymisiert den Datenverkehr mittels Onion Routing und bei Kommunikation mit einem Hidden Service ist dieser sogar Ende-zu-Ende verschlüsselt. Die Anonymität ist jedoch nur dann gewährleistet, wenn sich alle beteiligten Clients und Server im Tor Netzwerk befinden.

Zurzeit gibt es kaum Instant Messaging Clients deren Transportprotokolle dediziert für Tor entwickelt wurden. TorChat und Ricochet sind zurzeit die größeren und weitläufig bekanntesten Projekte, die vom

Funktionsumfang her betrachtet jedoch sehr eingeschränkt sind. Selbstverständlich existieren auch andere Lösungen wie XMPP, die aber nicht im speziellen für Tor entwickelt wurden. Viele Clients bieten nämlich die Möglichkeit, einen SOCKS Proxy für die Namensauflösung zu konfigurieren, sodass mit einem lokalen Tor Proxy kommuniziert werden kann. Prinzipiell ist es dadurch zwar möglich Protokolle wie XMPP durch das Tor Netzwerk zu tunneln, allerdings ist dies keine zufriedenstellende Lösung für nicht-technisch versierte Personen.

Die Enthüllungen globaler Spionageprogramme staatlicher Organisationen bewegten viele Menschen zu einem verstärkten Sicherheitsbewusstsein[32]. Viele Entwickler/innen arbeiten zurzeit an neuen Client Lösungen, die es auch nicht-technisch versierten Personen ermöglicht, sicher zu kommunizieren. In der näheren Zukunft werden daher Clients erwartet, die den Fokus auf starke Ende-zu-Ende Verschlüsselung und Anonymität richten.

IM Clients, aber auch andere Anwendungen, können sehr leicht um schützende Verschlüsselungstechnologien ergänzt werden. Dies wurde in dieser Arbeit mit DChat belegt. DChat ist ein rudimentärer, dezentraler Peer-to-Peer Client, welcher über eine SOCKS Schnittstelle mit dem Tor Proxy kommunizieren kann und dadurch anonyme Kommunikation ermöglicht. Das textbasierte Transportprotokoll ist sehr einfach aufgebaut und ermöglicht im Gegensatz zu den Protokollen von TorChat und Ricochet sogar Multi-User Chats. Unter anderem war es das Ziel dieser Arbeit, einen einfachen Secure-Messenger zu entwickeln. Dieses Ziel konnte mit DChat erfolgreich erreicht werden. DChat ist als Prototyp zu betrachten und dient für andere Projekte als Referenz, da die wesentlichen Aspekte wie die Schnittstelle zu Tor sehr gut dokumentiert wurden. Zukünftige Versionen von DChat werden zusätzliche nützliche Funktion wie Filesharing oder das Verwalten von Kontakten beinhalten und weitere unterstützte Plattformen wie Android als auch mehrere neue grafische Benutzeroberflächen sind zu erwarten.

Abschließend kann gesagt werden, wer seine Privatsphäre bei Instant Messaging schützen möchte, muss auf etablierte Technologien wie OTR und Tor zurückgreifen und einen Secure-Messenger einsetzen. Alternativ kann auch ein eigenes sicheres IM System konfiguriert werden, indem ein eigenständiger XMPP Server als Hidden Service eingerichtet wird. Dies wird jedoch ausschließlich technisch-versierten Personen empfohlen.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[1] Electronic Frontier Foundation, “Timeline of NSA Domestic Spying,” abgerufen am: 09.02.2015. [Online]. Available: https://www.eff.org/nsa-spying/timeline

[2] G. Greenwald und E. MacAskill, “NSA Prism program taps in to user data of Apple, Google and others,” Juni 2013, abgerufen am: 09.02.2015. [Online]. Available: http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data

[3] Juniper Research, “Anzahl der gesendeten Instant Messaging-Nachrichten weltweit im Jahr 2012 und Prognose für 2017 (in Billionen),” Statista, Februar 2014, abgerufen am: 09.02.2015. [Online]. Available: goldmanresearch.com

[4] Thomson Reuters, “Geschätzte Anzahl der täglich über WhatsApp geteilten Foto-, Video- und Sprachnachrichten weltweit im Jahr 2014 (in Millionen),” Statista, Februar 2014, abgerufen am: 09.02.2015. [Online]. Available: reuters.com

[5] IETF, “A Model for Presence and Instant Messaging,” Februar 2000, abgerufen am: 10.02.2015. [Online]. Available: https://tools.ietf.org/html/rfc2778

[6] Claudia Ruch, “Instant Messaging and Applications,” o.J.

[7] M. Mohammad und P. C. van Oorschot, “Secure public instant messaging: A survey,” *Proceedings of Privacy, Security and Trust*, pp. 69–77, 2004.

[8] S. Herring, D. Stein und T. Virtanen, *Pragmatics of computer-mediated communication*. Walter de Gruyter, 2013, vol. 9.

[9] KRDS, “Top 10 Instant Messenger nach Anzahl der Nutzer weltweit im Jahr 2013 (in Millionen),” Statista, Jänner 2014, abgerufen am: 10.02.2015. [Online]. Available: deutsche-startups.de

[10] Wilhelm Eisenschmid, "Peer-to-Peer-Architekturen," *Universität Ulm, Proseminar Virtuelle Präsenz*, pp. 1–12, 2005.

[11] A. Lehmann, T. Eichelmann und U. Trick, "Neue Möglichkeiten der Dienstebereitstellung durch Peer-to-Peer-Kommunikation," *ITG-Fachbericht-Mobilfunk*, 2008.

[12] P. Saint-Andre, K. Smith und R. Tronçon, *XMPP: the definitive guide*. O'Reilly Media, Inc., 2009.

[13] Telegram Messenger LLP, "MTProto Mobile Protocol," abgerufen am: 12.02.2015. [Online]. Available: https://core.telegram.org/mtproto

[14] IETF, "SIMPLE Made Simple: An Overview of the IETF Specifications for Instant Messaging and Presence Using the Session Initiation Protocol (SIP)," April 2013, abgerufen am: 15.02.2015. [Online]. Available: https://tools.ietf.org/html/rfc6914

[15] RetroShare Team, "What is Retroshare," abegrufen am: 15.02.2015. [Online]. Available: http://retroshare.sourceforge.net/wiki/index.php/Documentation:WhatisRetroShare

[16] Cyril, "Cryptography and Security in Retroshare," Dezember 2012, abgerufen am: 15.02.2015. [Online]. Available: https://retroshareteam.wordpress.com/2012/12/28/cryptography-and-security-in-retroshare/

[17] ——, "RetroShare's anonymous routing model," November 2012, abgerufen am: 15.02.2015. [Online]. Available: https://retroshareteam.wordpress.com/2012/11/03/retroshares-anonymous-routing-model/

[18] Hannes Mehnert, "Secure Instant Messaging-am Beispiel XMPP," 2008.

[19] Electronic Frontier Foundation, "Secure Messaging Scorecard," abgerufen am: 18.02.2015. [Online]. Available: https://www.eff.org/de/secure-messaging-scorecard

[20] N. Hindocha und E. Chien, "Malicious threats and vulnerabilities in instant messaging," in *Virus Bulletin Conference*, 2003.

[21] IETF, "Simple Authentication and Security Layer (SASL)," abgerufen am: 17.02.2015. [Online]. Available: http://tools.ietf.org/html/rfc2222

[22] N. Borisov, I. Goldberg und E. Brewer, "Off-the-record communication, or, why not to use PGP," in *Proceedings of the ACM workshop on Privacy in the electronic society*. ACM, 2004, pp. 77–84.

[23] Network Associates, Inc. und Tochtergesellschaften, *Einführung in die Kryptographie*. [Online]. Available: ftp://ftp.pgpi.org/pub/pgp/6.5/docs/german/IntroToCrypto.pdf

[24] Ogmios, "Web of Trust," abgerufen am: 19.02.2015; lizenziert unter CC BY-SA 3.0 über Wikimedia Commons.

[25] R. Dingledine, N. Mathewson und P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

[26] The Tor Project, Inc, "Tor: Hidden Service Protocol," abgerufen am: 26.02.2015. [Online]. Available: https://www.torproject.org/docs/hidden-services.html.en

[27] Bernd Kreuss, "TorChat - Messenger application on top of the Tor network and it's location hidden services," abgerufen am: 16.03.2015. [Online]. Available: https://code.google.com/p/torchat/

[28] John Brooks, "Ricochet - Anonymous peer-to-peer instant messaging," abgerufen am: 17.03.2015. [Online]. Available: https://ricochet.im/

[29] The Tor Project, Inc, "Tor's extensions to the SOCKS protocol," abgerufen am: 03.02.2015. [Online]. Available: https://gitweb.torproject.org/torspec.git/plain/socks-extensions.txt

[30] Ying-Da Lee, "SOCKS: A protocol for TCP proxy across firewalls," abgerufen am: 04.03.2015. [Online]. Available: http://www.openssh.com/txt/socks4.protocol

[31] ——, "SOCKS 4A: A Simple Extension to SOCKS 4 Protocol," abgerufen am: 04.03.2015.

[32] ZDF Politbarometer, "Umfrage zur Verbesserung des eigenen Datenschutzes nach Abhörskandal," Statista, Juli 2013, abgerufen am: 16.04.2015. [Online]. Available: http://www.zdf.de