Secure Instant Messaging - am Beispiel XMPP

Hannes Mehnert
hannes@berlin.ccc.de
XMPP: hannes@jabber.berlin.ccc.de

8. April 2008

Zusammenfassung

Instant Messaging wird schon länger viel benutzt. Dabei wird auch immer wieder behauptet, dieses sei "sicher". Dieser Artikel versucht, diese weit verbreitete These mit Fakten zu unterlegen, in welchen Fällen welche Sicherheitseigenschaften gewährleistet sind. Betrachtungsgegenstand ist das Instant Messaging Protokoll XMPP (früher: Jabber), da dieses frei und offen ist. Es werden verschiedene Verschlüsselungsmöglichkeiten diskutiert. Des weiteren wird anonymes Instant Messaging mit Hilfe von TOR und XMPP vorgestellt.

1 Kurze Übersicht über XMPP

XMPP [16] ist ein **offener** Standard, der XML zum Austausch sämtlicher Daten benutzt.

Die Infrastruktur für XMPP ist ein **dezentrales** Netzwerk. Eine XMPP-ID ist wie eine eMail-Adresse aufgebaut: $user@$host. Optional wird noch eine Komponente, /$ressource verwendet, um mehrere gleichzeitige Sessions zu unterstützen. Als Beispiele werden in diesem Artikel `alice@jabber.foo.com/somewhere` und `bob@jabber.bar.com/somewhere-else` benutzt.

XMPP ist **flexibel**, da es durch so genannte XEPs (XMPP Extension Protocols) erweitert werden kann. Auch Integration von anderen Instant Messaging Protokollen ist vorhanden mit Hilfe von Transports.

2 Kommunikation

Es gibt Client-Server- und Server-Server-Kommunikation. Die Client-Server-Kommunikation findet über TCP Port 5222 [1] bzw. 5223 (SSL) statt. Die Server-Server-Kommunikation findet allgemein über TCP Port 5269 statt.

[1] Die hier genannten Ports sind die normal konfigurierten und von der IANA vergebenen, es können aber via DNS SRV records andere Ports benutzt werden

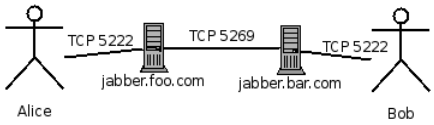

Abbildung 1: Verbindung

Wenn `alice@jabber.foo.com` eine Nachricht an `bob@jabber.bar.com` mit dem Inhalt "Hello, world" verschicken will, verschickt der Client von Alice diese erstmal an den Server `jabber.foo.com`, der die Nachricht weiter an den Server `jabber.bar.com` schickt. Dieser schickt die Nachricht an Bobs Client, falls dieser online [2] ist. Wenn er nicht online ist, speichert der Server die Nachricht, bis Bobs Client online kommt, siehe Abbildung 1.

3 XMPP ist "sicher"

Es kann nicht per se gesagt werden, ein Protokoll sei sicher, da zumindest die Bedrohung (bzw. das Angriffsszenario), vor der das Protokoll sicher sein soll, genannt werden muss.

Wenn ein Angreifer oder eine Angreiferin im lokalen Netz als größte Gefährdung ausgemacht wird, reicht eine verschlüsselte Verbindung vom Client zum Server. Wenn allerdings der Angreifer oder die Angreiferin Zugriff auf den Server hat, reicht die verschlüsselte Verbindung nicht mehr aus, sondern dann ist End-to-end-Encryption notwendig.

Bei XMPP sind prinzipiell folgende Daten schützenswert:

- **Nachrichten** - Inhalt der Nachricht - "Hello, world"
- **Verbindungsdaten** - Metadaten der Nachricht - von `alice@jabber.foo.com` an `bob@jabber.bar.com`
- **Presence** - Statusänderung eines Accounts - Alice ist nun online
- **Roster (Buddy List)** - Liste der Accounts, mit denen kommuniziert wird (Speicherung auf dem Server) - Bob bekommt die Presence-Nachricht, dass Alice online ist, wenn er in Alice Roster ist
- **vCard** - angegebene Daten der Accountinhaberin oder des Accountinhabers - Alice Nachname, Adresse, etc.

In der Kryptographie werden verschiedene Eigenschaften der Verschlüsselung unterschieden:

- **Vertraulichkeit**: niemand außer Alice und Bob können die Nachricht lesen

[2] Genauer online mit einer Priorität >= 0. Eine Priorität kann von einem Client gesetzt werden. Prioritäten kleiner 0 sind primär für Bots gedacht

- **Integrität**: niemand kann die Nachricht auf dem Weg von Alice zu Bob verfälschen, ohne dass Bob dies entdeckt
- **Authentizität**: Bob kann überprüfen, dass die Nachricht tatsächlich von Alice kam
- **Verbindlichkeit**: Alice kann nicht abstreiten, die Nachricht verschickt zu haben

In den folgenden Abschnitten werden zunächst die beiden erwähnten Szenarien analysiert.

4 Angreifer im lokalen Netzwerk

Zuerst wird das Szenario, dass Alice mit Bob kommunizieren will, ohne dass eine mögliche Angreiferin Eve dieses mithören kann, betrachtet. In diesem Szenario hat Eve nur Zugriff auf das lokale Netzwerk von Alice oder Bob und keinen Zugriff auf den Server. Den Servern (sowohl den Personen mit Zugriff auf diese als auch installierten Betriebssystemen sowie laufenden Services) wird somit vollständig vertraut.

Die zu schützenden Daten sind die Nachrichten sowie die Verbindungsdaten und die Presence. Da diese alle über die gleiche Verbindung verschickt werden und den Servern vertraut wird, ist eine detaillierte Betrachtung der verschiedenen Daten nicht notwendig.

Um gegen dieses Angriffsszenario sicher zu sein, muss die Client-Server-Kommunikation die beschriebenen kryptografischen Eigenschaften erfüllen. Auf diese wird im Folgenden eingegangen.

Anschließend wird das Szenario erweitert, so dass Eve Zugriff auf das Netzwerk zwischen den beiden Servern hat. Daher wird die Server-Server-Kommunikation kurz untersucht.

4.1 Vertraulichkeit, Integrität und Verbindlichkeit der Client-Server Kommunikation

Im RFC 3920 [1] wird zur Vertraulichkeit TLS [3] empfohlen. Dieses muss vom Client und Server implementiert werden. Dieses bietet verschiedene Ciphersuites, mindestens soll `TLS_RSA_WITH_3DES_EDE_CBC_SHA` implementiert sein. Das bedeutet, RSA Schlüssel werden zum Austausch der symmetrischen Schlüssel für 3DES benutzt. Zur Datenintegrität wird der Hashalgorithmus SHA1 benutzt.

Hier wäre sicherlich die Erweiterung um RFC 3268 [4], AES in TLS, als Mindestanforderung sinnvoll, da AES schneller als 3DES arbeitet und eine größere Key- und Blocklänge bietet.

Der Client muss laut 5.1 (Use of TLS) das Zertifikat validieren. Die Validierung (14.2) muss in mehreren Schritten ausgeführt werden. Zunächst sollte das

Zertifikat gegen die erwartete Identität geprüft werden. Falls dies nicht erfolgreich ist, muss der Benutzer oder die Benutzerin notifiziert werden. Im zweiten Schritt sollte das Zertifikat und die gesamte Kette von Zertifikaten der Benutzerin oder dem Benutzer zur Bestätigung gezeigt werden, falls es nicht anhand der bereits vertrauten Certificate Authorities verifiziert werden kann. Der Client muss das Zertifikat speichern, um bei der nächsten Verbindung zum gleichen Server überprüfen zu können, ob es sich geändert hat.

Die Validierung muss durchgeführt werden, falls das Zertifikat in einem Vertrauensanker (trust anchor) terminiert, also, falls es irgendeinen Weg von Root-Zertifikaten, denen vertraut wird, zu dem vorliegenden Zertifikat gibt. Falls das Zertifikat von einer nicht bekannten Certificate Authority signiert wurde oder ein self-signed Zertifikat ist, sollte die Validierung ausgeführt werden.

Certificate Authorities ist ein zentralistisches Konzept, bei dem definiert wird, welchen Trust Center vertraut wird. Dieses passt gut zu hierarchischen Strukturen, wo die oberste Hierarchieebene bestimmt, was vertrauenswürdig ist; nämlich alles, was von ihr kommt. Auf chaotische Strukturen, die meist hierarchiefrei sind, lässt sich sowas nicht abbilden. Wieso sollte mensch delegieren, wem er alles vertraut? Auch der CACert Ansatz ist hier nicht sinnvoll, da bei diesem nicht detailliert spezifiziert werden kann, wem vertraut wird. Möglicherweise will ich nur meinen drei besten Nerdfreunden vertrauen, und denen, den sie vertrauen, aber nicht dem gesamten CACert, da unklar ist, wie diese Leute den entsprechenden Trustlevel erreicht haben.

4.1.1 Angriff: Person in the middle

Bei einem Person in the middle (PITM) Angriff täuscht Eve Alice vor, der Server zu sein. Falls der Server nicht hinreichend authentifiziert wird, also hier das Zertifikat nicht hinreichend überprüft wird, kann Eve Benutzernamen und Passwort von Alice abfangen, falls der Client eine Klartext Authentifizierung zulässt.

Da die gesamte Kommunikation über Eve läuft, kann diese von Eve gelesen und nach Belieben geändert und gefiltert werden.

Um ein PITM zu verhindern, muss das Zertifikat also validiert werden. Zumindest das Speichern des Zertifikates ist eine gute Idee, da damit nur die erste Verbindung ohne PITM durchgeführt werden muss, um eine spätere PITM zu erkennen, ähnlich wie bei ssh mit den `known_hosts`. Falls ein PITM erfolgreich durchgeführt wird, sind Eve Benutzername und Passwort bekannt und Eve kann sich somit als Alice einloggen und ausgeben.

4.1.2 SSL/TLS in der Praxis

Betrachten wir nun die eigene Client-Server-Verbindung, die vom lokalen Client zu einem offenen XMPP-Server wie `jabber.ccc.de` oder `jabber.berlin.ccc.de` besteht. Da fällt auf, dass diese keine Zertifikate haben, die von einer allgemein als vertrauenswürdig angesehenen CA signiert wurden.

Server Das Zertifikat von `jabber.ccc.de` ist ein CAcert, somit muss der CAcert CA vertraut werden, um die Validierung vorschriftsgemäß durchzuführen.

Das Zertifikat von `jabber.berlin.ccc.de` ist von der CCCV CA signiert, deren Zertifikat via `http://jabber.berlin.ccc.de/cccv_ca.pem` erhältlich ist.

Client Nun zur Integration der Zertifikate in verschiedene Clients. Getestet wurden Psi 0.10 [5], AdiumX 1.2.1 [6], Pidgin 2.2.1 [8] und Gajim 0.11.2 [7].

Psi Die geteste Version 0.10 speichert die vertrauenswürdigen Zertifikate in der Datei `~/.psi/certs/rootcert.xml`. Falls das Serverzertifikat nicht validiert werden kann, wird eine Warnung angezeigt, die in den Optionen allerdings abgeschaltet werden kann. Für die Verbindung mit SSL ist das Plugin Qt Cryptographic Architecture (QCA) erforderlich.

Eine aktuelle Version von Psi (0.11 oder eine Revision 1065 aus dem Subversion) bietet die Möglichkeit, einfach das Zertifikat im PEM Format in `~/.psi/certs` zu hinterlegen. Somit ist keine manuelle Editierung der XML Datei notwendig.

Es muss allerdings darauf geachtet werden, dass "Ignore SSL warnings" in den Connection-Einstellungen ausgeschaltet ist. Andernfalls werden SSL Warnungen ignoriert und ein PITM ist möglich.

Adium Seit Version 1.2 (Januar 2008) bietet auch Adium die Möglichkeiten, Zertifikate zu überprüfen. In der aktuellen Version (1.2.3) gibt es bei den Account-Einstellungen "Require SSL/TLS" und "Do strict certificate checks". Beides sollte eingeschaltet sein, damit das Zertifikat des Servers validiert wird [3]. Falls dieses unbekannt ist, erscheint ein Popup-Window, ob das Zertifikat akzeptiert werden soll (und in die Keychain hinzugefügt werden soll).

Pidgin Bei Pidgin (früher Gaim) findet keine Verifizierung des Zertifikates statt. Nur wenn "Force old (port 5223) SSL" eingeschaltet ist, wird SSL gesprochen. "Require SSL/TLS" in Pidgin macht einfach eine unverschlüsselte Verbindung zum Server auf, wenn "Force old SSL" ausgeschaltet ist und der Server kein TLS unterstützt.

In Pidgin kann für die SSL/TLS Unterstützung sowohl Network Security Services (NSS) als auch GnuTLS verwendet werden, standardmäßig wird NSS benutzt. Im Quellcode ist für GnuTLS die Validierung des Zertifikates weiter implementiert als für NSS, wo einfach OK zurückgegeben wird: `pidgin-2.3.1/libpurple/plugins/ssl/ssl-nss.c`:

```
static SECStatus
ssl_auth_cert(void *arg, PRFileDesc *socket, PRBool checksig,
                         PRBool is_server)
{
```

[3] Allerdings hat Adium Probleme bei der Hostname Zuordnung, falls SRV records existieren

```
        return SECSuccess;
#if 0
...
#endif
}
```

Somit werden alle Zertifikate als vertrauenswürdig angesehen. pidgin-2.3.1/libpurple/plugins/ssl/ssl-gnutls.c:

```
static void
ssl_gnutls_verified_cb(PurpleCertificateVerificationStatus st,
                       gpointer userdata)
{
        PurpleSslConnection *gsc = (PurpleSslConnection *) userdata;

        if (st == PURPLE_CERTIFICATE_VALID) {
                /* Certificate valid? Good! Do the connection! */
                gsc->connect_cb(gsc->connect_cb_data, gsc, PURPLE_INPUT_READ);
        } else {
                /* Otherwise, signal an error */
                if(gsc->error_cb != NULL)
                        gsc->error_cb(gsc, PURPLE_SSL_CERTIFICATE_INVALID,
                                      gsc->connect_cb_data);
                purple_ssl_close(gsc);
        }
}
```

Hier wird die Verbindung abgelehnt, falls das Zertifikat nicht validiert werden kann.

Gajim Bis einschließlich zur Version 0.11.2 gab es keine Unterstützung zur Verifikation von Zertifikaten in Gajim. Dort konnte mittels "Use SSL (Legacy)" SSL eingeschaltet werden.

Der aktuelle Gajim (Revision 9342 aus dem Subversion) bietet die Möglichkeit, Zertifikate zu verifizieren. Der Client versucht automatisch, zuerst eine TLS-Verbindung aufzubauen, falls dieses nicht erfolgreich ist, eine SSL-Verbindung aufzubauen; falls beides fehlschlägt, wird eine unverschlüsselte Verbindung aufgebaut. Einstellbar ist dies über "Advanced Configuration Editor", dann beim entsprechenden Account durch Setzen von "connection_types".

Auch das Überprüfen und Speichern des Fingerprints (für die nächste Verbindung zum gleichen Server) ist vorgesehen [21], funktioniert allerdings gerade nicht.

Fazit Bei keinem der Clients wird das Zertifikat für eine nächste Verbindung gespeichert.

Somit sind aus Vertraulichkeitsgründen Psi, Adium und Gajim aus dem Subversion (oder >= 0.12) zu empfehlen, jeweils mit sorgfältig überprüften und eingepflegten CA Zertifikaten.

4.2 Authentifizierung

XMPP spezifiziert in [1] die Benutzung von SASL, dem Simple Authentication and Security Layer, [2]. Es wird mindestens DIGEST-MD5 gefordert, was eine Challenge-Response Authentifizierung, in der als Hashalgorithmus MD5 verwendet wird, entspricht. MD5 ist in den letzten Jahren durch einige Angriffe ins Gespräch gekommen, diese lassen sich aber nicht auf die Verwendung im Challenge-Response Verfahren anwenden.

Im XEP78 [9] wird die Authentifizierung, die früher verwendet wurde, erläutert. Hier handelt es sich um eine Plaintext- oder Digest-Authentifizierung . Bei Plaintext wird das Passwort in Klartext übermittelt. Bei Digest Authentifizierung wird SHA1(concatenate(Stream-ID, Passwort)) zurückgegeben.

Alle getesteten Releases der Clients haben noch immer mit der alten Authentifizierung gearbeitet, wobei zumindest Digest und nicht Plaintext benutzt wurde. Angeblich sollen die Clients aber auch SASL unterstützen.

Die Sicherheit beider Mechanismen ist in etwa gleich, jeweils wird dem Client etwas zufälliges vom Server mit dem Benutzerpasswort zusammengesetzt, und danach der Hash davon zum Server zurückgesendet.

Der Zufall vom Server sollte schon kryptografisch zufällig sein, ansonsten kann mit Hilfe von mitgeschnittenen Authentifizierungsvorgängen offline ein Wörterbuch-Angriff gemacht werden (falls der Zufall beispielsweise von der Serverzeit abhängt). Eine Untersuchung der Stream-ID in verschiedener Serverimplementationen, die in dem obsoleten, aber noch benutzten XEP78 als Nonce verwendet wird, brachte folgendes:

- `jabberd-2`: (trunk/sx/server.c:143)

```
random wurde mit srand(time(NULL)) geseeded
for(i = 0; i < 40; i++) {
  r = (int) (36.0 * rand() / RAND_MAX);
  id[i] = (r >= 0 && r <= 9) ? (r + 48) : (r + 87);
}
```

- `jabberd1`: Version 1.4.4 (jabberd/lib/xstream.c:236):

```
sprintf(id,"%X",(int)time(NULL));
```

- `jabberd1`: Version 1.6.1.1 (jabberd/lib/xstream.cc:356):

```
der PRNG wird mit srand(time(NULL)) geseeded

snprintf(id, sizeof(id), "%08X%08X%08X%08X%08X",
         rand(), rand(), rand(), rand(), rand());
shahash_r(id, id); /* don't let them see what our rand() returns */
```

- `ejabberd`: (src/randoms.erl:53):

  ```
  seed vom random mit now()

  get_string() ->
       random_generator ! {self(), get_random, 65536*65536},
       receive
           {random, R} ->
               integer_to_list(R)
       end.
  ```

  Dieses benutzt das Erlang random Modul [17].

Somit haben zumindest aktuelle Serverimplementationen (außer Jabberd1-1.4.4, der aber schon 2 Jahre alt ist) keine kryptografisch hochwertigen zufälligen Stream-IDs, da die benutzten Pseudo Random Number Generators (PRNG) mit der Zeit des Serverstartups initialisiert werden. Mit der "Last Activity" Erweiterung [25] kann die Serverstartup-Zeit sekundengenau angefragt werden. Diese eröffnet zumindest die theoretische Möglichkeit, die Stream-ID vorherzusagen. Beim ejabberd wird als Seed auch die Mikrosekunden benutzt.

4.3 Server-Server-Kommunikation

Auch bei der Server-Server-Kommunikation sollte sowohl SASL als auch TLS benutzt werden. Zusätzlich gibt es noch "server dialback", das gegen domain spoofing schützen soll. Eine Server-Server-Verbindung sieht also wie folgt aus: `jabber.foo.com` stellt eine Verbindung mit `jabber.bar.com` auf Port 5269 her. `jabber.bar.com` stellt eine Verbindung mit `jabber.foo.com` her, um zu überprüfen, ob `jabber.foo.com` auch tatsächlich der Rechner ist, von dem aus die Verbindung aufgebaut wurde. Anschließend passiert das gleiche für den Rückkanal, also fängt `jabber.bar.com` an, die Verbindung zu initiieren.

Die verschiedenen Serverimplementationen können wohl untereinander via TLS kommunizieren. Eine genauere Untersuchung fand nicht statt, da eine Benutzerin oder ein Benutzer den Status der Verbindung von einem Server zu einem anderen Server nicht nachprüfen kann. Aus Sicherheitsgründen muss davon ausgegangen werden, dass die Verbindung nicht verschlüsselt wird.

4.4 Fazit

Falls dem Server vertraut wird, ist für eine Benutzerin oder einen Benutzer unklar und nicht überprüfbar, ob die Server-Server-Kommunikation verschlüsselt wird oder nicht. Somit kann möglicherweise jeder Rechner im Netzwerk von Server `jabber.foo.com` zu Server `jabber.bar.com` mitlesen.

Das einzige, wo sich mensch sicher sein kann, ist die eigene Client-Server-Kommunikation, falls das Zertifikat erfolgreich verifiziert wurde. Die Client-Server Kommunikation des Gegenüber muss nicht zwingend verschlüsselt sein.

Hier muss je nach Expertise und geistigem Zustand des Gegenübers abgewägt werden.

Bei TLS/SSL wird die Kommunikation zwischen den einzelnen Komponenten verschlüsselt, also vom Client zum Server, dann vom Server zu Server, dann vom Server zum Client. Die einzelnen Server haben die kompletten Kommunikationsdaten unverschlüsselt vorliegen. Falls der Empfänger oder die Empfängerin momentan nicht online ist, wird die Nachricht auch unverschlüsselt auf dem Server des Empfängers gespeichert.

5 Eigener Server vertrauenswürdig

Falls nur dem Server von Alice vertraut wird, nicht aber dem von Bob, bleiben weiterhin sowohl die Nachrichten als auch die Verbindungsdaten und die Presence für den Server von Bob unverschlüsselt. Auch der Eintrag von Alice in Bobs Roster ist für den Server von Bob nachvollziehbar.

Der besondere Schutz der Nachricht wird im nächsten Szenario detailliert behandelt.

6 Server nicht vertrauenswürdig

Als nächstes Szenario wird wieder betrachtet, dass Alice eine Nachricht an Bob schicken will. Diesmal wird allerdings dem Server nicht vertraut. Dieses kann verschiedene Gründe haben: der Anwenderin oder dem Anwender ist unbekannt, wer Zugriff auf die Server hat; ob die Zugriffsberechtigten möglicherweise mit Angreifern oder Angreiferinnen oder Institutionen (Strafverfolgungsbehörden) zusammenarbeiten; oder ob das Betriebssystem und die laufenden Dienste auf dem Server unsicher sind, so dass sich andere Personen dadurch Zugriff zum Server verschaffen können.

Es soll also verhindert werden, dass Eve, die Zugriff auf einen Server hat, die Kommunikation mitlesen kann.

Hierzu ist also Verschlüsselung von Alice zu Bob nötig, ohne dass die Nachricht auf den Servern entschlüsselt und wieder verschlüsselt wird. Dies wird End-to-end-Encryption genannt. In den nächsten Sektionen wird auf zwei unterschiedliche Mechanismen der End-to-End-Encryption eingegangen (PGP und OTR).

Bei der End-to-end Encryption werden nur Nachrichten verschlüsselt, also keine Verbindungsdaten oder Presences.

Somit hat Eve schon die Informationen, dass Alice mit Bob kommuniziert (Verbindungsdaten). Diese können nicht geschützt werden, da sie auf den Servern dafür benötigt werten, um die Nachricht von Alice an Bob entsprechend weiterzuleiten.

Auch weiß Eve, dass Alice online ist (Presence). Die Presence kann nicht geschützt werden, da sie an alle Kontakte im Roster gesendet wird. Falls sie also

verschlüsselt werden sollte, müsste sie an jeden Kontakt einzeln verschlüsselt werden; dazu müsste jeder Kontakt Verschlüsselung unterstützen.

Zusätzlich kann Eve durch bestimmte Erweiterungen die Jabber-Client-Version, das Betriebssystem, die Zeit des letzten Logins, die Zeit der letzten Nachricht, die Geolokation, die aktuell gehörte Musik.

Auch die "Freunde" von Alice kennt Eve schon anhand des Rosters; möglicherweise gruppiert nach Themengebieten oder was Alice für eine Sortierung vorgenommen hat.

Falls Alice persönliche Daten in die vCard eingetragen hat, weiß Eve auch diese, da diese auch auf dem Server gespeichert werden.

6.1 End-to-end-Encryption

Die Eigenschaften der Verschlüsselungsmethode sollten hier auch wieder Vertraulichkeit, Integrität, Authentizität sowie Verbindlichkeit sein.

PGP [11] wurde zur Verschlüsselung von eMail "erfunden". Es basiert auf öffentlichen und privaten Schlüsseln. Öffentliche Schlüssel sind für alle TeilnehmerInnen via Keyserver erhältlich. Deren Echtheit muss überprüft werden (durch die Überprüfung des Fingerprints über einen sicheren Kommunikationskanal wie persönliches Treffen, Telefon o.ä.), damit die Authentizität gewährleistet ist. Es bietet alle geforderten Eigenschaften: Vertraulichkeit durch Verschlüsselung, Integrität, Authentizität und Verbindlichkeit durch digitale Signaturen.

Alle Nachrichten werden mit dem öffentlichen Schlüssel des Empfängers verschlüsselt, so dass nur dieser die Nachricht entschlüsseln kann. Meist werden die Nachrichten auch zusätzlich mit dem öffentlichen Schlüssel des Senders verschlüsselt, damit dieser die Nachrichten später nochmal lesen kann. Der Sender kann mit seinen privaten Schlüssel die Nachricht zu signieren.

Somit können alle Nachrichten auch zu einem späteren Zeitpunkt von einer Inhaberin oder einem Inhaber des privaten Schlüssels entschlüsselt werden. Falls also zu einem späteren Zeitpunkt der private Schlüssel in die Hände von Eve gelangt, und die Nachrichten aufgenommen wurden, können diese entschlüsselt werden. Wenn die Nachrichten auch eine Signatur enthalten, was für Authentizität unerlässlich ist, kann sogar "kryptografisch" nachgewiesen werden, dass Alice an Zeitpunkt X eine bestimmte Nachricht an Bob geschickt hat (unter der Voraussetzung, dass an diesem Zeitpunkt nur Alice ihren privaten Schlüssel hatte).

Daher haben sich einige Wissenschaftler Gedanken über private Konversationen und Anforderungen gemacht [13]. Sie haben das Off-the-record Protokoll (OTR) entwickelt [12]. Auch dieses basiert auf öffentlichen und privaten Schlüsseln. Es gibt aber pro Sitzung spezielle Sitzungsschlüssel, die unabhängig von den öffentlichen Schlüsseln sind. OTR bietet zusätzliche Eigenschaften:

- **Perfect forward secrecy**: ein Sitzungsschlüssel ist nicht abhängig vom vorherigen Sitzungsschlüssel

- **Malleable encryption**: die Änderung eines Bits im verschlüsselten Text provoziert diegleiche Änderung des Bits im entschlüsselten Text

Perfect forward secrecy bietet somit die Eigenschaft, dass, wenn ein Sitzungsschlüssel von Eve herausgefunden wird, nicht sämtliche mitgeschriebene Kommunikation entschlüsselt werden kann, da der nächste Sitzungsschlüssel unabhängig vom vorhergehenden ist. Dieses wird dadurch erreicht, dass immer kurzlebige Schlüssel (beispielsweise für jede Nachricht ein neuer Schlüssel) erzeugt werden, die mit Hilfe eines Authenticated Diffie-Hellmann Key Exchanges zwischen den Kommunikationspartnern ausgetauscht werden.

Malleable encryption wird durch ein XOR des Schlüsselstroms mit dem Klartext erreicht. Somit wird, falls ein Bit im verschlüsselten Text geändert wird auch der Klartext bei gleichen Schlüsselstrom. Dieses hat folgende nützliche Eigenschaft: Es kann nicht nachgewiesen werden, dass Alice einen bestimmten Nachrichtentext an Bob geschrieben hat. Dieses führt zu plausible deniability, also, später kann Alice plausibel machen, nicht die entsprechende Nachricht mit Bob gewechselt zu haben.

Prinzipiell wird auch auf mögliche Angriffe gegen diese Methoden eingegangen. Dazu werden sowohl die Speicherung der privaten Schlüssel und das Verhalten, falls ein Benutzer gerade offline ist, im Detail betrachtet.

Im "Freiheitblog" gibt es eine Serie von Artikeln zur Konfiguration von End-to-end-Encryption verschiedener Clients [27] [28] [29] [30] .

6.2 PGP Encryption

Wie PGP in XMPP benutzt wird, ist in XEP 27 [10] spezifiziert. Jede Nutzerin und jeder Nutzer hat einen privaten und einen öffentlichen Schlüssel. Der Schlüssel der Kommunikationspartnerin bzw. des Kommunikationspartners muss vorher verifiziert worden sein.

PGP Schlüsselaustausch Der Austausch der öffentlichen Schlüssel wird durch Keyserver vollzogen, die Vertrauenswürdigkeit kann anhand des PGP Web of trust geprüft werden. Das Web of trust ist das Vertrauen, das mensch den Personen gibt, deren öffentlichen Schlüssel signiert hat. Diese Signaturen der öffentlichen Schlüssel werden zusammen mit den öffentlichen Schlüsseln auf Keyservern gespeichert. Um einen Schlüssel zu signieren, muss der Fingerprint des öffentlichen Schlüssels auf einem authentifizierten Kommunikationskanal [4] überprüft werden. [5]

[4] Indem die Identität des Schlüsselinhabers oder der Schlüsselinhaberin sinnvoll überprüft wurde. Wenn der Schlüssel für eMail benutzt wird, also die Validität der eMail-Adresse und ob die Inhaberin oder der Inhaber eMails and diese Adresse empfangen kann. Bei Nutzung von XMPP dementsprechend analog die XMPP-ID.

[5] Eigentlich "sollte" statt "muss", da natürlich mensch auch ohne Überprüfung des Fingerprints den Schlüssel signieren kann. Der Signatur ist nicht anzusehen, wie sehr die Identität der angeblichen Besitzerin oder des angeblichen Besitzers und der Fingerprint überprüft worden sind.

Wenn Bob und Carol sich schon länger kennen und einander vertrauen, haben sie gegenseitig ihre Schlüssel signiert. Alice und Bob kennen sich auch schon länger und vertrauen sich. Wenn Alice nun Carol kennenlernt, kann Alice anhand der Signatur von Bob auf Carols Schlüssel und dem Vertrauen zu Bob verifizieren, dass Carols Schlüssel Carol gehört, ohne den Schlüssel auf einem gesicherten Kommunikationskanal zu überprüfen.

Das Web of trust ist eines der ersten Social Networks. Es beinhaltet die Informationen, wer wen wann signiert hat. Dieses kann beispielsweise als Kennenlernen der Personen gewertet werden. Aus Datensparsamkeitsgründen muss mensch sich genau überlegen, ob beziehungsweise welche Informationen von ihm oder ihr publik sind. Natürlich funktioniert das Web of trust am Besten, wenn jede und jeder die Schlüssel der Leute unterschreibt, denen sie und er vertraut und die sinnvoll überprüft worden.

6.2.1 Discovery

Jede Presence Nachricht wird kryptografisch signiert, damit wird zum Ausdruck gebracht, dass die Benutzerin oder der Benutzer PGP verschlüsselte Nachrichten empfangen und entschlüsseln kann, und welche KeyID sie oder er benutzt.

Hier sind Replay-Attacken möglich. Das heißt, eine empfangene Presence von Alice kann von Eve gespeichert und zu einem späteren Zeitpunkt nochmal versendet werden, so dass Bob denkt, Alice wäre online und würde gerade diese Presence verschicken. Hierzu muss es Eve allerdings möglich sein, diese Presence als Alice zu verschicken, dazu benötigt sie entweder das Passwort von Alice oder Zugriff auf den XMPP-Server von Alice oder Bob. Auch Zugriff auf das Netzwerk zwischen Alices und Bobs Server ist ausreichend, um eine Replay-Attacke durchzuführen.

6.2.2 Key Storage

Der private PGP-Schlüssel ist meist durch eine Passphrase geschützt auf der Festplatte gespeichert. Der Benutzer oder die Benutzerin muss sie beispielsweise beim Start des Chatclients angeben, diese wird im RAM gespeichert. Aktuell sei hier auf [26] verwiesen, um Schlüssel aus dem RAM auszulesen.

6.2.3 Verschlüsselung von Nachrichten

Die Nachrichten, die Alice und Bob austauschen, werden verschlüsselt, aber nicht signiert. Auch hier kann Eve, bei Zugriff auf einen der XMPP-Server, Nachrichten, die mit Alice bzw. Bobs öffentlichen Schlüssel verschlüsselt sind, einschleusen, falls der öffentliche Schlüssel bekannt ist. Also können weder Alice noch Bob sicher sein kann, dass Bob bzw. Alice die entsprechende Nachricht versendet hat.

Eve kann nicht lesen, worüber Alice und Bob kommunizieren, wenn sie nicht im Besitz eines privaten Schlüssels von Alice oder Bob ist. Somit kann sie auch nur Nachrichten, die möglicherweise kontextfrei sind, einschleusen.

6.2.4 Offline Storage

Die Nachrichten, die Alice an Bob schickt, während Bob offline ist, sind auch verschlüsselt, und werden somit verschlüsselt auf dem Server gespeichert.

6.2.5 Geteste Clients

Von mir getestete Clients, die die PGP Verschlüsselung unterstützen, sind Gajim und Psi; außerdem noch Jarl [15] (nicht mehr aktiv in Entwicklung). Einige Vergleichstabellen von vielen Clients ist unter [31] oder [32] verfügbar.

Sowohl bei Gajim als auch bei Psi muss die Zuordnung zwischen Benutzer und Schlüssel von Hand gemacht werden, hier wird nicht anhand der signierten Presence der richtige Schlüssel ausgesucht.

6.2.6 Fazit

Alle abgefangenen Nachrichten können, falls einer der privaten Schlüssel zu irgendeinem Zeitpunkt abhanden kommt, vollständig entschlüsselt werden.

Wenn die einzelnen Nachrichten zusätzlich noch signiert wären, könnte auch kryptografisch bewiesen werden, dass Alice die Nachrichten an Bob geschickt hat, indem die Signatur der Nachrichten überprüft wird. Natürlich nur, wenn bewiesen werden kann, dass Alice zum Zeitpunkt des Sendens die einzige war, die ihren privaten Schlüssel kannte.

Auch nicht außer Acht zu lassen sind die Replay-Attacken auf Presence Nachrichten und die Einspeisung neuer verschlüsselter Nachrichten.

Bob kann aufgrund fehlender Signatur nicht nachprüfen, ob tatsächlich Alice gerade die Nachricht geschrieben hat.

6.3 Off-the-Record Messaging

Bei Off-the-Record Messaging hat jeder Benutzer einen öffentlichen und privaten Schlüssel. Auch der initiale Austausch des Fingerprints muss durch einen authentifizierten Kommunikationskanal geschehen, beispielsweise durch eine PGP Signatur des Fingerprints des Schlüssels, falls Alice und Bob schon PGP Schlüssel von dem jeweils anderen validiert haben.

6.3.1 Discovery

Bei Off-the-Record werden die Nachrichten nicht signiert. Dafür wird die erste herausgehende Nachricht um verschiedene Whitespaces erweitert, die angibt, dass OTR unterstützt wird.

Außerdem kann in den Clients explizit angefordert werden, dass eine Session verschlüsselt werden soll, dann wird der Schlüsselaustausch gestartet.

6.3.2 Session initiation

Falls beide Clients OTR supporten, kann mit Hilfe eines Authenticated Diffie-Hellman Key Exchange eine Nonce ausgehandelt werden, die nicht abhängig

von den privaten und öffentlichen Schlüssel der Chatteilnehmer ist, aber durch einen Dritten nicht abgehört werden kann, da nicht alle Parameter kommuniziert werden.

Somit wird ein Session Key ausgehandelt, mit dem die Nachrichten verschlüsselt werden können.

6.3.3 Verschlüsselung von Nachrichten

Nachrichten werden verschlüsselt und mit einem signierten Authentication Code (HMAC) versehen. Somit kann Bob überprüfen, dass Alice tatsächlich die entsprechende Nachricht verschickt hat.

6.3.4 Client State

Ein OTR Client muss somit Informationen über den Status der Session haben, ob diese aktuell verschlüsselt ist mit einem Session Key; oder ob gerade der Session Key ausgehandelt wird, etc.

Dieses kann zu Problemen führen, wenn die States der beiden Clients nicht kompatibel sind, also der eine Client denkt, sie wären gerade am Neuverhandeln eines Session Keys und der andere noch einen alten Session Key hat.

Auch wenn Pakete nicht in der richtigen Reihenfolge ankommen, kann dieses den Client verwirren, so dass Nachrichten mit einem neueren Session Key zuerst eintreffen, diese nicht entschlüsselt werden können, und später, nachdem der Session Key upgedatet wurde, ältere Nachrichten, die auch nicht mehr entschlüsselt werden können.

Dieses ist nur rudimentär gelöst, indem die zuletzt versendete Nachricht zwischengespeichert wird, und, falls der andere Client einen Fehler zurücksendet, der Session Key neu ausgehandelt wird und die Nachricht mit dem neuen Schlüssel verschlüsselt ein weiteres mal versendet wird.

6.3.5 Key Storage

OTR speichert den privaten Schlüssel unverschlüsselt auf der Festplatte. Falls Eve Zugriff zu diesem bekommt, können ab dann verschlüsselte Sessions von Eve initiiert werden, ohne dass dieses entdeckt werden kann.

6.3.6 Offline Storage

Wenn Bob gerade nicht online ist, können, falls der Client von Alice noch eine Session mit Bobs Client hat, weiterhin verschlüsselte Nachrichten verschickt werden, die dann auf dem Server gespeichert werden. Falls Bob allerdings seinen Client beendet hat, und somit die Session geschlossen hat, kann dieser die Nachrichten von Alice nach Verbindung nicht entschlüsseln, da der Session Key nicht mehr vorhanden ist.

6.3.7 Clients

Clients, die OTR unterstützen, sind Pidgin und AdiumX. Auch psi-0.11 mit einem Patch [18] kann OTR unterstützen. Leider kann mit diesem nur entweder mit allen Kontakten via OTR kommuniziert werden oder mit keinem und er hat kein Benutzerinterface im Chatfenster. Diese Benutzungsprobleme wurden mit [19] und [20] gelöst.

6.4 Fazit

Sowohl PGP als auch OTR haben noch Probleme: die Netzwerkinstabilität (packet loss, out of order), die zur Verwirrung der States der OTR Clients führen können; das nicht wirklich gelöste Problem von OTR sind die offline Nachrichten; die nicht signierten Nachrichten, die ein PGP Client verschickt. Dies kann auch irrelevant sein, beispielsweise, wenn aus dem Kontext klar hervorgeht, worüber Alice und Bob reden, und Eve nur unsinnige Nachrichten einstreut (da Eve ja die Nachrichten von Alice an Bob nicht lesen kann).

7 Anonymes Instant Messaging

Im folgenden wird vorgestellt, wie Instant Messaging mit Hilfe von TOR [14] anonym betrieben werden kann. TOR ist ein Anonymisierungsnetzwerk, dass Onion Routing benutzt. Somit wird jedes Paket, das verschickt werden soll, mit mehreren Zwiebelschalen verpackt und an den ersten Router geschickt. Dieser kann die äußerste Zwiebelschale entfernen (entschlüsseln) und schickt den Rest des Paketes an den nächsten Router. Somit ist nicht nachvollziehbar, von wo nach wo das Paket verschickt wurde (es sei denn eine Entität hat Zugriff auf viele Router)

Bei TOR gibt es Hidden Services, diese sind nur über TOR zu erreichen und haben einen Namen, der aus 16 Zeichen (ein Teil des Fingerprints des public Keys) und `.onion` besteht, beispielsweise `ww7pd547vjnlhdmg.onion`. Diese haben also keine öffentliche IP, sondern sind nur über das TOR-Netzwerk erreichbar, da der Hostname indirekt auf TOR Router zeigt, die bestehende Verbindungen vom Hidden Service haben. Durch einige Verbindungen vom TOR Client zu mehreren TOR Routern ist somit eine anonyme Verbindung möglich, wo weder der Server noch der Client wissen, mit welcher IP sie kommunizieren.

Da diese Verbindung schon verschlüsselt ist, ist es nicht mehr notwendig zwischen XMPP Client und XMPP Server mit SSL/TLS zu verschlüsseln.

7.1 Anfallende Daten und notwendiges Vertrauen

Wenn nun ein XMPP-Server als TOR Hidden Service betrieben wird, kann nur noch der Betreiber oder die Betreiberin sehen, wer mit wem chattet. Für beispielsweise den Internet Provider eines Nutzers oder einer Nutzerin ist nicht einsehbar, dass gerade ein XMPP Dienst, geschweige denn welcher, genutzt wird.

Somit fallen auch bei der Vorratsdatenspeicherung keine interessanten Daten beim Internet-Provider an.

Entweder muss dem Betreiber oder der Betreiberin des Servers vertraut werden oder ein eigener Server muss installiert werden. Der Betreiber bzw. die Betreiberin sieht, welche Daten auf dem Server hinterlegt wurden sowie welche Kontakte im Roster vorhanden sind. Aber immerhin kann der Betreiber oder die Betreiberin nicht mehr adressiert werden, diese oder dieser ist anonym, da der TOR Hidden Service nicht lokalisiert oder zu einer realen Person zugeordnet werden kann.

Im allgemeinen nicht davon ausgegangen werden, dass alle Server vollständig anonym sind, einige können möglicherweise auch aus dem öffentlichen Internet erreicht werden. Sowohl Server-Server Kommunikation zwischen anonymen und nicht anonymen Server ist denkbar und möglich, als auch, dass ein Transport existiert, der zwischen dem anonymen und dem nicht anonymen Netzwerk kommuniziert.

Die Server-Server Kommunikation gestaltet sich etwas komplizierter, da der XMPP Hidden Service keine Netzwerkverbindung in das Internet haben sollte, sondern Verbindungen nach außen nur über den SOCKS-Proxy von TOR herstellen soll.

In bestimmten Szenarien, falls beispielsweise ein Nutzer oder eine Nutzerin nur Accounts eines anderen, öffentlichen XMPP-Servers, in dem Roster hat, ist die Anonymität nicht mehr gewährleistet, da aufgrund der Einträge in den anderen Rostern unmittelbar auf den Benutzer oder die Benutzerin geschlossen werden kann. Somit kann es passieren, dass die Anonymität eines Nutzers oder einer Nutzerin aufgehoben wird.

7.2 Patch für ejabberd

Nicht anonyme XMPP-Server, die eine öffentliche IP Adresse besitzen, müssen, falls sie mit anonymen XMPP-Servern kommunizieren wollen, verändert werden, so dass sie für `.onion` XMPP-Server den TOR SOCKS-Proxy benutzen.

Auch wenn zwei anonyme Server kommunizieren wollen, muss die Serversoftware den SOCKS-Proxy von TOR benutzen.

Hierzu wurde ein Patch [22] entwickelt, der dem ejabberd die Nutzung von SOCKS-Proxies ermöglicht, und durch eine Konfigurationsoption kann gesteuert werden, welcher SOCKS-Proxy benutzt wird, ebenso welche Verbindungen über den SOCKS-Proxy aufgebaut werden sollen. Hier kann mensch zwischen `all`, `none` (default) und `dot_onion_only` gewählt werden.

Ein TOR Hidden Service, der die Option auf `dot_onion_only` gesetzt hat, kann somit nur mit anderen TOR Hidden Services reden. Dieses ist sicherheitstechnisch auf den ersten Blick das "sicherste", da somit verhindert wird, dass von dem Server an einen öffentlichen Server Daten geschickt werden. Falls allen anderen Hidden Services, mit welchen Nachrichten ausgetauscht werden, vertraut wird, dass sie vollständig anonym sind, ist dies auch so. Solange also jede/r Einzelne einen eigenen XMPP-Server als reinen TOR Hidden Service betreibt, sind die jeweiligen Daten am Besten geschützt.

7.3 Praktische Tests

Der Verbindungsaufbau zu einem Hidden Service ist deutlich langsamer als zu einem öffentlichen Server, da das TOR zuerst einen Circuit zum Lookup-Server aufbauen muss; Dort muss ein Introduction Point des Hidden Service gesucht werden; zu diesem muss wieder ein Circuit aufgebaut werden. Zusätzlich braucht es noch einen Rendezvous Point, und die Daten des Rendezvous Point müssen dann via Introduction Point an den Hidden Service übermittelt werden. Der Hidden Service muss daraufhin eine Verbindung zum Rendezvous Point aufbauen. Insgesamt müssen also vier verschiedene TOR Circuits aufgebaut werden, was eine deutliche Latenz beinhaltet. Nachdem die Verbindung zum Hidden Service erfolgreich aufgebaut wurde, hat die Kommunikation auf demgleichen Server etwa vier Sekunden Latenz.

Die Server-Server Kommunikation muss insgesamt vier dieser Verbindungen aufbauen, da eine Verbindung von Server `ww7pd547vjnlhdmg.onion` zu `3khgsei3bkgqvmqw.onion` erst nach einem erfolgreichen Dialback von `3khgsei3bkgqvmqw.onion` zu `ww7pd547vjnlhdmg.onion` erfolgreich aufgebaut ist. Diese ist dann nur für Nachrichten von `ww7pd547vjnlhdmg.onion` zu `3khgsei3bkgqvmqw.onion`. Für die Rückrichtung werden somit nochmal zwei TCP-Verbindungen aufgebaut. Für die initiale Verbindung zwischen zwei Server können somit durchaus fünf Minuten vergehen. Das ist auch der Grund, wieso im Patch zu ejabberd der `FSMTIMEOUT` deutlich erhöht werden musste.

Ab dann ist eine Latenz von meist weniger als acht Sekunden von Nachrichten von einem Benutzer oder einer Benutzerin des Servers `3khgsei3bkgqvmqw.onion` zu einem Benutzer oder einer Benutzerin des Servers `ww7pd547vjnlhdmg.onion`.

7.4 Client Support

Bei allen benutzten Clients war es möglich, einen SOCKS-Proxy einzutragen. Leider machen sowohl Adium als auch Psi DNS Anfragen des Hostnames über den normalen Resolver, statt den Hostname an den SOCKS-Proxy weiterzugeben. Somit wird der Hostname des XMPP-Servers, zu dem verbunden werden soll, an den eingetragenen Nameserver unverschlüsselt via Netzwerk übermittelt.

Für Psi gibt es einen Patch, so dass Psi keine DNS-Anfragen mehr versendet, falls ein SOCKS-Proxy eingetragen ist [23].

Bei Pidgin und Adium funktionierte die Einstellung eines SOCKS-Proxy out of the box.

Bei Gajim wird Version aus dem Subversion benötigt. Zusätzlich muss in dem “Advanced Configuration Editor” bei den “connection types” der entsprechenden Verbindung “tls” und “ssl” gelöscht werden, so dass nur noch plain da steht. Ansonsten verschluckt sich der Gajim an dem SSL Handshake.

7.5 Ausblick

Die Entwicklung eines Transports, der zwischen anonymen und öffentlichen XMPP-Accounts Kommunikation ermöglicht, steht noch aus. Dieses ist wie beschrieben nicht ganz ungefährlich, da dieser Transport eine zentrale Stelle darstellt, an der Kommunikationsdaten gesammelt werden. Somit gehen auch sämtliche Presence sowie Nachrichten durch diesen Transport, was Begehrlichkeiten wecken könnte, um anonyme ChatterInnen zu deanonymisieren.

Multi-User Chat [24] benutzt eine Subdomain des Hostname, also `conferences.jabber.ccc.de` oder auch `conferences.3khgsei3bkgqvmqw.onion`. TOR kann diese allerdings nicht auflösen, somit ist MUC nur innerhalb eines Hidden Servers möglich. Eine Lösung wäre, in TOR alle `*.foo.onion` an `foo.onion` zu schicken.

Server-Server Kommunikation zwischen anonymen und öffentlichen Servern sollte mit SSL/TLS verschlüsselt werden, da ansonsten die TOR Exit Nodes die unverschlüsselten Nachrichten sehen.

8 Fazit

Es wurden sowohl Client-Server als auch Server-Server Kommunikation untersucht. Es gab zwei Angriffsszenarien, dem Server wird vertraut bzw. nicht vertraut. Beim letzteren Szenario wurden zwei verschiedene End-to-end encryption Systeme erklärt. Beide sind nicht frei von Fehlern; sie bieten verschiedene Eigenschaften. Je nachdem, welche kryptografischen Eigenschaften von einer Person angestrebt sind, sollte entschieden werden, ob PGP oder OTR verwendet wird.

Jedes der kryptografischen Verfahren ist besser, als unverschlüsselt zu kommunizieren

Anonymes Chatten ist durch die Installation eines XMPP-Servers als TOR Hidden Service möglich. Auch hier muss beachtet werden, wie viel Sicherheit notwendig ist, und dann Abstriche bei den Kommunikationsmöglichkeiten gemacht werden (beispielsweise ein reiner TOR Hidden Service Server). In einer geschlossenen Benutzergruppe, die viel Kommunikationsdisziplin an den Tag legt, und einen oder mehrere eigene Server betreibt, kann somit anonym und sicher kommuniziert werden.

Literatur

[1] RFC 3920 Extensible Messaging and Presence Protocol (XMPP): Core `http://www.ietf.org/rfc/rfc3920.txt`

[2] RFC 2222 Simple Authentication and Security Layer (SASL) `http://www.ietf.org/rfc/rfc2222.txt`

[3] RFC 2246 The TLS Protocol Version 1.0 `http://www.ietf.org/rfc/rfc2246.txt`

[4] RFC 3268 Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS) http://www.ietf.org/rfc/rfc3268.txt

[5] Psi http://psi-im.org/

[6] AdiumX http://www.adiumx.com/

[7] Gajim http://www.gajim.org/

[8] Pidgin http://www.pidgin.im/

[9] XEP 0078: Non-SASL Authentication http://www.xmpp.org/extensions/xep-0078.html

[10] XEP 0027: Current Jabber OpenPGP Usage http://www.xmpp.org/extensions/xep-0027.html

[11] OpenPGP http://www.openpgp.org/

[12] Off-the-Record Messaging http://www.cypherpunks.ca/otr/

[13] Off-the-Record Communication, or, Why Not To Use PGP http://www.cypherpunks.ca/otr/otr-wpes.pdf

[14] TOR, The onion router https://www.torproject.org/

[15] Jarl - A Perl/Tk Jabber Client http://jarl.sourceforge.net/

[16] Wikipedia: Extensible Messaging and Presence Protocol http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol

[17] Erlang Random Number Generator http://www.erlang.org/doc/man/random.html

[18] Off-the-Record Messaging Plugin for Psi http://www.tfh-berlin.de/~s717689/

[19] Patch für Psi mit Benutzerinterface http://berlin.ccc.de/~hannes/psi-otr-socks.diff

[20] OTR Plugin für Psi mit Benutzerinterface http://berlin.ccc.de/~hannes/psi-otr-0.3-hacked.tar.gz

[21] Gajim: SSL certificate verification feature http://trac.gajim.org/ticket/720

[22] Patch für ejabberd-2, um SOCKS Kommunikation zu ermöglichen http://berlin.ccc.de/~saite/patches/ejabberd-2.0.x_socks5.diff bzw. http://berlin.ccc.de/~saite/patches/ejabberd-2.0.x_socks5.txt

[23] Patch für Psi, der keine DNS Anfragen mehr macht, wenn ein SOCKS-Proxy gewählt ist http://berlin.ccc.de/~hannes/psi-socks.diff

[24] XEP 0045: Multi-User Chat http://www.xmpp.org/extensions/xep-0045.html

[25] XEP-0012: Last Activity http://www.xmpp.org/extensions/xep-0012.html

[26] Cold boot attacks on disk encryption http://citp.princeton.edu.nyud.net/pub/coldboot.pdf

[27] Freiheitblog: Verschlüsseltes Instant Messaging- Teil 1: Einleitung http://freiheitblog.wordpress.com/2008/02/25/verschlusseltes-instant-messaging-teil-1-einleitung/

[28] Freiheitblog: Verschlüsseltes Instant Messaging- Teil 2: Pidgin http://freiheitblog.wordpress.com/2008/02/25/verschlusseltes-instant-messaging-teil-2-pidgin/

[29] Freiheitblog: Verschlüsseltes Instant Messaging- Teil 3: Kopete http://freiheitblog.wordpress.com/2008/03/03/verschlusseltes-instant-messaging-teil-3-kopete/

[30] Freiheitblog: Verschlüsseltes Instant Messaging- Teil 4: Trillian http://freiheitblog.wordpress.com/2008/03/06/348/

[31] Wikipedia: Comparison of instant messaging clients http://en.wikipedia.org/wiki/Comparison_of_instant_messaging_clients

[32] End-User Jabber Clients http://www.jabber.org/clients/