Universität Konstanz
Fachbereich Informatik und Informationswissenschaft
Lehrstuhl Verteilte Systeme

Bachelorarbeit

Implementierung & Evaluation eines JavaScript-basierten, OTR-fähigen XMPP Clients

Klaus Herberth
01/754547

23. September 2013

1. Gutachter: Prof. Dr. Marcel Waldvogel
2. Gutachter: Prof. Dr. Marc H. Scholl

Betreuer: Daniel Kaiser & Daniel Scharon

Zusammenfassung

Diese Arbeit führt in die Techniken eines XMPP-Clients inklusive Off-the-Record Messaging ein. Dabei wird auf die Funktionsweise beider Protokolle eingegangen und im weiteren Verlauf die Entwicklung und Implementierung eines JavaScript-basierten Clients für die soziale Kommunikationsplattform Diaspora erläutert. Zum Schluss werden sowohl Unterschiede zwischen Desktop- und Browser-Client behandelt, als auch Sicherheitsaspekte einer JavaScript-Anwendung und die Funktionsweise in verschiedenen Browsern.

Selbstständigkeitserklärung

Der Verfasser erklärt, dass er die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Klaus Herberth
Konstanz, 23. September 2013

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit befasst sich mit der Erstellung und Evaluation eines unabhängigen Chats für die Plattform Diaspora[1]. Dieser stellt dabei die Möglichkeit einer sicheren Ende-zu-Ende Verschlüsselung bereit, bei welcher der Benutzer die Chat-Anwendung nicht als eigenständiges Programm wahrnimmt, sondern als Teil der sozialen Plattform. Hierbei ist es wichtig die Anwendung grafisch an die bestehende Oberfläche anzupassen und Benutzungsprinzipien beizubehalten.

Als Name wurde „JavaScript Xmpp Chat“ gewählt, im weiteren Verlauf nur noch als JSXC abgekürzt. JSXC ist trotz seiner nahtlosen Integration ein eigenständiger Chatclient. Das heißt man muss selbst seine Kontakte hinzufügen und Anfragen beantworten. Dabei wird die Kontaktliste des Benutzers vom XMPP Server übernommen und auf Änderungen reagiert. Dies bedeutet, sollte sich ein Benutzer unter dem gleichen Namen in einem zweiten Chatclient einloggen und dort einen Kontakt hinzufügen, wird dieser automatisch auch bei JSXC angezeigt.

1.1 Anforderungen

Die folgende Liste sortiert die Anforderungen an den Chatclient nach ihrer Priorität. Das heißt Punkte die sich weiter unten befinden sind nicht zwingend erforderlich sondern als nützliche Ergänzung zu verstehen (kursiv markiert).

1. Nahtlose Integration in die bestehende Benutzeroberfläche
2. Getrennte gleichzeitige Kommunikation mit verschiedenen Partnern
3. 1-zu-1 Kommunikation
4. Verschlüsselte Kommunikation
5. *Gleichzeitige Benutzung in mehreren Tabs*
6. *Unterstützung in allen geläufigen Browsern (Internet Explorer, Mozilla Firefox, Opera, Safari, Chrome)*
7. *Unterstützung von Smilies*
8. *Flexibilität*

1.2 Abgrenzung

Folgend einige Punkte die nicht von JSXC abgedeckt werden sollen.

1. Datenaustausch
2. Mehrbenutzerchat
3. Benachrichtigungen (Anwesenheit, Tippen, ...)

[1] Diaspora stellt eine quelloffene Alternative zu herkömmlichen sozialen Plattformen dar. Weitere Informationen unter `http://diasporaproject.org/`

1.3 Bestehende Software

Noch folgend eine Auflistung bestehender Applikationen welche einen ähnlichen Funktionsumfang bieten, aber aus verschiedenen Gründen nicht geeignet sind und damit eine Neuentwicklung nötig machten.

- Jappix Mini [9]
- Candy-Chat [6]
- Cryptocat [7]
- Prodromus [16]
- ZXMPP [22]

Gegen *Jappix Mini* spricht, dass es in einem Komplettpaket mit der sozialen Plattform Jappix kommt. Eine Entkopplung wäre bestimmt möglich gewesen, aber ein fehlender Roster, die Fokussierung auf Gruppenchats und die wenig verbreitete XMPP-Bibliothek JSJaC[2] sprachen dagegen.

Candy-Chat ist nicht für eine Integration in eine bestehende Webseite entwickelt worden, sondern als eigenständige Anwendung, die den ganzen Browser in Anspruch nimmt. Damit eignet sie sich nicht für eine Eingliederung in Diaspora.

Einen Chat innerhalb eines Browsers ermöglicht auch *Cryptocat*, der dabei aber einen ganz anderen Weg geht. Die Anwendung integriert sich direkt als Add-On in den Browser und macht damit eine Installation nötig. Dies ist bei Diaspora nicht gewollt und daraus folgt, dass Cryptocat ausscheidet.

Prodromus dient als kleine Support-Anwendung die keine Übereinstimmung mit den Anforderungen hat.

Der Entwickler von *ZXMPP*, Ivan Vučica, hat sich dazu entschieden keine existierenden Bibliotheken zu verwenden[3] und alles von Grund auf selbst zu programmieren. Da die Wartung und Erweiterbarkeit bei Ein-Mann-Projekten ein Problem darstellt, schied ZXMPP aus.

[2] `https://github.com/sstrigler/JSJaC/`

[3] Im Gegensatz dazu setzt JSXC auf weit verbreitete Bibliotheken, siehe 4.3.

2 Grundlagen

Dieses Kapitel führt in die Welt der Webapplikationen ein und bildet eine Grundlage für den weiteren Verlauf dieser Abschlussarbeit.

2.1 Client-seitig

Der Client ist in unserem Fall der PC des Benutzers welcher später die Diaspora Webseite öffnet. Alle Techniken, die zur Darstellung oder Arbeitsweise von JavaScript Xmpp Chat (JSXC) von Nöten sind, werden folgend aufgelistet:

2.1.1 Hypertext-Markup-Language (HTML)

HTML ist eine Auszeichnungssprache, die hauptsächlich zur Strukturierung in Webseiten verwendet wird. Dabei beschreibt die Sprache den Inhalt, ohne ihm eine Darstellung zu verleihen. Wie die beschriebenen Elemente aussehen ist abhängig vom Browser und den definierten Styles (siehe 2.1.2).

2.1.2 Cascading Stylesheets (CSS)

Bei CSS handelt es sich um eine Sprache für Formatvorlagen. Mit dieser kann man Farbe, Form, Position und etliche weitere Attribute von HTML-Elementen bestimmen. Besonders für Webseiten ist diese Sprache nützlich, da sie zentral ausgelagert und anschließend in jedes Dokument eingebunden werden kann.

2.1.3 Javascript (JS)

JS ist eine clientseitige Skriptsprache, dass heißt sie wird direkt im Browser ausgeführt, womit der Server keine Informationen über den Ablauf der Skripte hat.

2.1.4 Asynchronous JavaScript and XML (AJAX)

AJAX ist keine neue Skriptsprache, sondern eine Beschreibung für den asynchronen Datenaustausch mittels JS. Bei klassischen Webseiten sendet der Benutzer über seinen Browser eine Anfrage an den Server und bekommt eine komplette Webseite zurück. Bei AJAX kann man nun per JS neue Daten anfordern und diese in der Webseite anzeigen (Abbildung 2.1), damit ist kein erneutes Laden der Seite nötig.

2.1.5 Web Storage[1]

Bis vor kurzem war die einzige Möglichkeit Daten bei einem Benutzer zu speichern die Verwendung von Cookies, welche zwei gravierende Nachteile haben: Erstens werden Cookies bei jeder Anfrage mitgesendet. Somit hat auch der Server Zugriff auf diese Daten. Zweitens ist der Speicherplatz sehr begrenzt.

Web Storage, eine der Neuerungen in HTML5, besitzt diese beiden Hindernisse nicht und ermöglicht es, je nach Browser, 5 MB oder mehr zu speichern. Dabei unterscheidet man zwischen Local- und

[1] `http://www.w3schools.com/html/html5\_webstorage.asp`

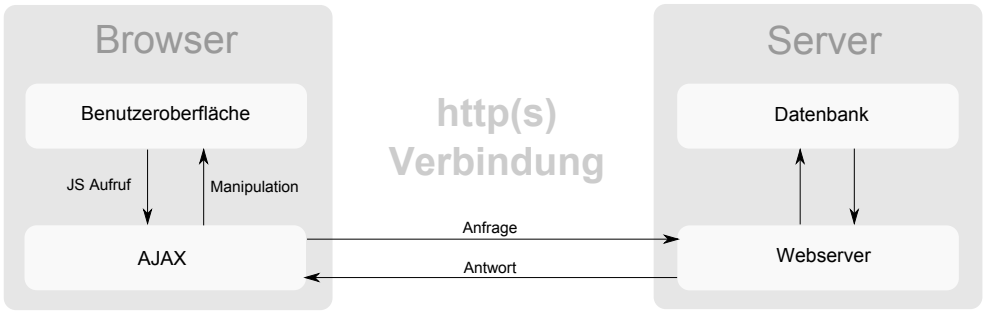


Abbildung 2.1: Funktionsweise einer asynchronen AJAX Verbindung.

Session-Storage. Local-Storage wird bis zur expliziten Löschung durch den Benutzer oder das Programm aufbewahrt. Wohingegen Session-Storage nur über die Lebensdauer eines Tabs vorhanden ist. Fragen in Bezug zur Sicherheit des Web-Storage werden in Abschnitt 5.3.4 behandelt.

Technisch gesehen ist der Webstorage dabei ein Key-Value-Store. Das bedeutet man speichert und lädt Werte nur anhand eines Schlüssels. Es existieren keine Tabellen oder Ähnliches.

2.2 Server-seitig

Der Server ist der Computer welcher die Anfragen an Diaspora verarbeitet und Antworten zurück schickt.

2.2.1 Ruby-on-Rails (RoR)

RoR ist ein Web-Framework in der Sprache Ruby zum Erstellen von dynamischen Seiten. Das bedeutet, beim Besuch ein und derselben Webseite können unterschiedliche Inhalte präsentiert werden. Dank dynamischer Seiten sind Gästebücher und Ähnliches komfortabel[2] möglich.

2.3 Protokolle

2.3.1 Extensible Messaging and Presence Protocol (XMPP)

XMPP[3] ist das Protokoll über welches der eigentliche Chatverkehr abgehandelt wird. Jeder Benutzer hat dabei eine eigene Jabber-Id (Jid), welche vom Aufbau her einer E-Mail-Adresse entspricht. Um mit mehreren Clients gleichzeitig unter einer Jid erreichbar zu sein, wird eine sogenannte Ressource an die Jid angehängt. Diese kann eine beliebige Zeichenkette sein; beispielsweise:

klaus.herberth@uni-konstanz.de/Arbeit

Die Möglichkeiten von XMPP gehen dabei weit über einfache Chat-Anwendungen hinaus. „The purpose of XMPP is to enable the exchange of relatively small pieces of structured data (called "XML stanzas") over a network between any two (or more) entities.“ [17]. Dazu wird zwischen Client und Server oder Server und Server jeweils in beide Richtungen ein Stream geöffnet. Dies geschieht durch Senden eines öffnenden Stream-Tags („<stream>“). Innerhalb diesem können nun XML Stanzas ausgetauscht werden. Es gibt den „message“ Stanza für allgemeine Nachrichten, den „presence“ für die Verfügbarkeit von Teilnehmern und einen Frage-Antwort Stanza, genannt „iq“, für allgemeine Anfragen [vgl. 17]. Welche Struktur die einzelnen Elemente genau haben,

[2]Natürlich könnte auch ein Gästebuch oder eine soziale Plattform mit Hilfe von E-Mails und jeder Menge manueller Arbeit erstellt werden.

[3]ehemals Jabber

wird in den XMPP Extension Protocols (XEP) spezifiziert, die die Grundbausteine von XMPP bilden. So definiert RFC6121 „the basic functionality expected of an instant messaging and presence application [...]“ [18]. Darüber hinaus existieren noch eine Vielzahl von weiteren Erweiterungen, wie zum Beispiel für den Dateiversand oder den Multi-Benutzer Chat. Möchte man einen geöffneten Stream schließen, wird ein „</stream>“ gesendet.

Ein beispielhafter Ablauf ist in Tabelle 2.1 gezeigt.

Client	**Server**
`<stream>`	
	`<stream>`
... SASL Authentifizierung (2.3.2) ...	

`<message to='ex@amp/le'>`
`  <body>Hallo</body>`
`</message>`	

	`<presence from='ex@amp/le1' to='me@foo'>`
	`  <show>chat</show>`
	`</presence>`

`<iq type='get' id='jas8dfkjl3'>`
`  <query xmlns='jabber:iq:roster'/>`
`</iq>`	

	`<iq to='me@foo/res' id='jas8dfkjl3' type='result'>`
	`  <query xmlns='jabber:iq:roster'>`
	`  <item subscription='both' name='Freund 1' jid='f1@foo'/>`
	`  <item subscription='both' name='Freund 2' jid='f2@foo'/>`
	`  <item subscription='both' name='Freund 3' jid='f3@foo'>`
	`  <group>EveryBody</group>`
	`  </item>`
	`  </query>`
	`</iq>`

`<... />`	
	`<... />`
`</stream>`	
	`</stream>`

Tabelle 2.1: Ablauf einer Client zu Server Kommunikation. Zuerst wird der Stream geöffnet, um darauf hin eine Nachricht zu versenden. Anschließend wird der Client über die Verfügbarkeit eines Freundes informiert, worauf hin dieser seinen Roster anfragt und erhält. Am Ende wird der Stream geschlossen.

2.3.2 Simple Authentication and Security Layer (SASL)

„The Simple Authentication and Security Layer (SASL) is a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms.“ [1]. Dieses Protokoll dient als Abstraktionsschicht zwischen Protokollen und Methoden zur Authentifizierung. Damit ist es möglich verschiedene Verfahren anzubieten und das sicherste, welches bei beiden Parteien verfügbar ist, auszuwählen. Hierzu sendet der Client an den Server eine Anfrage mit der Bitte um alle verfügbaren Methoden. Danach wählt er eine aus und sendet diese zurück (Abbildung 2.2). Erst danach fängt der eigentliche Authentifizierungsprozess an.

SASL ist weit verbreitet und findet unter anderem Verwendung bei LDAP, SMTP, POP und IMAP.

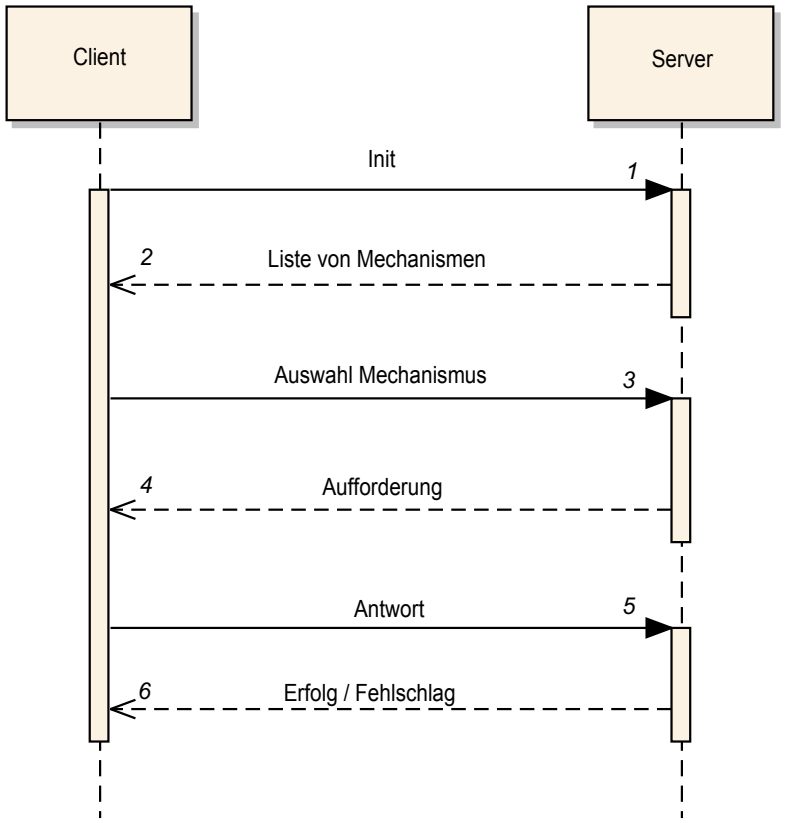


Abbildung 2.2: Ablauf der SASL Kommunikation

Als verpflichtenden Mechanismus wird bei XMPP Salted Challenge Response Authentication Mechanism (SCRAM), siehe 2.3.2, vorgeschrieben: „[...] servers and clients MUST support the SASL Salted Challenge Response Authentication Mechanism [...]“ [17, S. 161]. Aus Kompatibilitätsgründen werden auch noch DIGEST-MD5 und PLAIN unterstützt, welche alle in den nächsten Abschnitten erläutert werden.

PLAIN

PLAIN ist der einfachste aber auch unsicherste von allen drei Mechanismen. Das Passwort wird als Klartext an den Server gesendet, und anschließend mit dem gespeicherten Wert[4] verglichen. Sollte keine Übereinstimmung festgestellt werden, wird der Benutzer abgewiesen. Da das Mitlesen bei

[4]Dem Server steht es frei ob er das Password ebenfalls im Klartext speichert oder es durch eine Hash-Funktion schützt. In letzterem Fall muss er das empfangene Passwort ebenfalls hashen um es vergleichen zu können.

unverschlüsselten Verbindungen keine Probleme bereitet und somit eine Replay-Attacke[5] fördert, wird die Benutzung von Transport Layer Security (TLS) empfohlen. „The mechanism is intended to be used with data security protections provided by application-layer protocol, generally through its use of Transport Layer Security (TLS) services.“ [21, S. 2].

Unter der Prämisse, dass der Benutzer das Zertifikat kontrolliert, ist der Austausch des Passwortes über diese Methode sicher. Da diese Überprüfung eher selten praktiziert und allzu oft einem unsicheren Zertifikat vertraut wird, ist PLAIN nur die letzte Möglichkeit. Um Downgrade-Attacken[6] zu unterbinden ist es vorteilhafter diesen Mechanismus gar nicht anzubieten.

DIGEST-MD5

Um Replay-Angriffe zu umgehen wurde der DIGEST-MD5 Mechanismus entwickelt. Dabei sendet der Server dem Client eine beliebige Zeichenfolge, genannt „nonce“, mit welcher das Passwort gehasht wird. Der Unterschied zwischen nonce und dem bekannten Salt[7] ist, dass der Salt langlebig ist und eine nonce nur für einen einzigen Hash benutzt wird. Um chosen-plaintext attacks (CPAs)[8] zu vermeiden fügt auch der Client eine nonce hinzu, die „cnonce“. Damit auf dem Server keine Klartext-Passwörter abgelegt werden, wird das Passwort zusammen mit einem realm[9] und dem Benutzername gehasht (HP). Dieses Vorgehen wird nicht zwingend gefordert, sondern nur empfohlen. (vgl. [11])

$$HP = MD5(benutzername : realm : passwort)$$

Der Client berechnet nach Erhalt der nonce wie folgt die Antwort (vereinfacht):

$$response = MD5(HP : nonce : cnonce)$$

Wie der Ablauf einer solchen Kommunikation aussieht, sehen Sie in Tabelle 2.2.

S:	realm=”jabber.uni-konstanz.de”,nonce=”djO8dksKHDF8shdk”,charset=utf-8
C:	username=”klaus”, realm=”jabber.uni-konstanz.de”, nonce=”djO8dksKHDF8shdk”, cnonce=”lkKdl83KJjd832lK”, response=c0b7fc18669b55e609d7c385697c2e34, charset=utf-8
S:	rspauth=0c27a10770bae9555751ff01dc38383c

Tabelle 2.2: Vereinfachtes Beispiel der Aufforderung, der Antwort und dem Resultat einer DIGEST-MD5 Authentifizierung (C steht für Client und S für Server). Dabei sind die letzten drei Schritte (4-6) von Grafik 2.2 abgebildet. Das Resultat (rspauth) aus Schritt 3 besteht dabei aus einem Hash, welcher das Passwort enthält. Damit kann der Client sicher gehen, dass der Server in Besitz dessen ist („mutual authentication“).

Im Jahr 2011 wurde DIGEST-MD5 als SASL Mechanismus heruntergestuft und fortan auch von XMPP nicht mehr als Haupt-Methode empfohlen. Dieser Schritt wurde mit unterschiedlichen Punkten begründet. So ist die Dokumentation nicht eindeutig genug und bietet zu viele Schwachstellen. Eine davon ist, dass Passwörter und Benutzernamen von UTF-8 in ISO-8859-1 konvertiert werden müssen, oder dass nur MD5 als Hash-Algorithmus zugelassen ist. „The MD5 hash is sufficiently weak to make a brute force attack on DIGEST-MD5 easy with common hardware“ [12, S. 4]. Eine

[5] Bei einer Replay-Attacke wird durch das erneute Senden von zuvor mitgelesenen Informationen, versucht eine Authentifizierung zu erlangen.

[6] Ein Man-in-the-middle (MitM) könnte die Liste der verfügbaren Mechanismen so manipulieren, dass nur PLAIN zu Verfügung steht. Dies wird als Downgrade-Attacke bezeichnet. [vgl. 19, S. 112]

[7] Mit einem Salt soll verhindert werden, dass die Hashes auf unterschiedlichen Systemen bei dem gleichen Schlüssel identisch sind. Dazu wird vor der Anwendung des Hashes der Salt an den Schlüssel angehängt. [vgl. 19, S. 258]

[8] Bei einer CPA wird in unserem Fall die nonce von einem Angreifer verändert um den Hash leichter zu knacken.[vgl. 19, S. 61]

[9] Meist die Domain.

Behebung dieser Probleme ist zwar möglich, würde aber die Komplexität noch weiter steigern und neue Schwachstellen eröffnen, wie zum Beispiel eine Downgrade-Attacke. [vgl. 12] Dadurch ist die Verwendung eines anderen Mechanismuses zwingend erforderlich.

SCRAM

Wie im Abschnitt zuvor erläutert benötigt es einen neuen Mechanismus, welcher klar definiert ist und verschiedene Hash-Funktionen unterstützt. Aus diesen Gründen wurde SCRAM [13] entwickelt welcher HMAC (Hash-Message Authentication Code) [3] benutzt, „a mechanism for message authentication using cryptographic hash functions [...] with a secret shared key„ [10, S. 1], um dies zu erreichen. Der Vorteil von HMAC gegenüber normalen Hash-Funktionen ist, dass ein Angreifer nur einen gültigen Hash-Wert erzeugen kann, wenn er im Besitz des Geheimnisses ist. SCRAM benutzt als Geheimnis das Passwort des Benutzers (n) und generiert daraus in mehreren Iterationen (i) unter Zuhilfenahme eines Salt (s) das SaltedPassword. Schlussendlich wird über den ClientProof (p), welcher in mehreren Schritten zusammen mit einer nonce (r) erzeugt wird, die Korrektheit des Passwortes überprüft. Damit der Client sichergehen kann, dass er nicht mit einem Betrüger redet und der Server wirklich in Besitz des Passwortes ist sendet dieser nach erfolgreicher Verifikation eine Server-Signatur (rspauth) auf Basis des SaltedPassword und der nonce. Einen typischen Ablauf der Kommunikation sehen Sie in Tabelle 2.3.

C:	n=Klaus,r=beliebigeNonce
S:	r=beliebigeNoncePlusServerNonce,s=irgendEinSalt,i=4096
C:	r=beliebigeNoncePlusServerNonce,p=kld887jkdkhjkjKklB8dKJd8
S:	rspauth=kdi87GDke32kds

Tabelle 2.3: Ablauf einer SCRAM Kommunikation. Wie man sieht, fügt der Server zu der nonce des Clients noch eine Eigene hinzu.

Die XMPP Dokumentation schreibt zwingend 2 Hash-Algorithmen vor: „servers and clients MUST support the SASL SCRAM – inparticular, the SCRAM-SHA-1 and SCRAM-SHA-1-PLUS variants.“ [17, S. 161]. Darüber hinaus ist theoretisch jeder beliebige iterative kryptographische Hash-Algorithmus möglich.

2.3.3 Off-the-Record Messaging (OTR)

Das OTR Protokoll ermöglicht einen sicheren Datenaustausch („Encryption“) ohne eine spätere Beweismöglichkeit von wem welche Nachricht versendet wurde („Deniability“). Dabei ist während der Kommunikation trotzdem sichergestellt, dass man sich mit der richtigen Person unterhält („Authentication“). Dies wird über das Generieren von neuen Schlüsseln beim Nachrichtenaustausch bewerkstelligt. Durch die ständige Neugenerierung wird zudem sichergestellt, dass die einzelnen Nachrichten nicht mit dem Hauptschlüssel entschlüsselt werden können („Perfect forward secrecy“). Diese Eigenschaften sind der Grund warum OTR bei Chat-Anwendungen vor zum Beispiel PGP oder X.509-Zertifikaten bevorzugt wird. [vgl. 4]

Um den genaueren Ablauf einer OTR-Session zu erläutern nehmen wir an eine fiktive Person „Alice“ möchte mit „Bob“ eine private Unterhaltung führen. Dazu hat Alice zwei Möglichkeiten: Sie sendet mit einer Nachricht eine Folge von nicht druckbaren Kontrollzeichen[10] (Beispiel[11]) oder sendet eine explizite Anfrage (Beispiel[12]). Bei beiden Varianten wird gleichzeitig die gewünschte

[10] Diese Version hat den Vorteil, dass Bob keine unverständlichen Zeichen sieht, wenn sein Client OTR nicht unterstützt. Des weiteren wird diese Methode so angewendet, dass automatisch an jede gesendete Nachricht Kontrollzeichen angehängt werden. Damit wird die Verschlüsselung, wenn verfügbar, benutzt und kann nicht vergessen werden. Leider kann es aber sein, dass manche Clients implizite Anfragen nicht annehmen.

[11] „Eine ganz normale Nachricht. $x20x09x20x20x09x09x09x09x20x09x20x09x20x09x20x20x20x20x20x09x09x20x20x09x09$“

[12] „*?OTRv2?* Hier folgt noch eine beliebige und optionale Erklärung, falls Bob's Client kein OTR versteht.“.

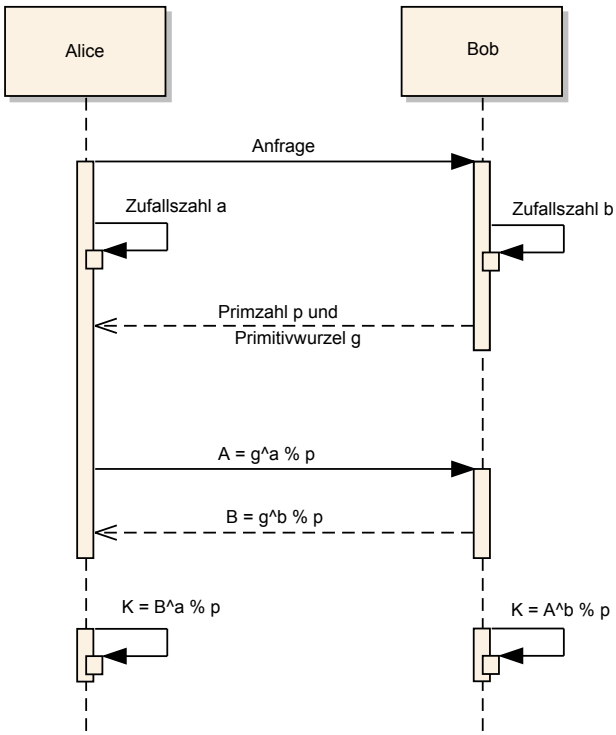


Abbildung 2.3: Ablauf eines Diffie-Hellman Schlüsselaustausches ohne Signierung. Wie man sieht kann der gemeinsame Schlüssel K nur mit den nicht übertragenen Zufallszahlen a und b errechnet werden.

Protokoll-Version angehängt.

Sollte Bob die Anfrage akzeptieren folgt der Authenticated Key Exchange (AKE). „The general idea is that Alice and Bob do an unauthenticated Diffie-Hellman (D-H) key exchange to set up an encrypted channel, and then do mutual authentication inside that channel." [14]. Nach einem erfolgreichen Schlüsselaustausch kennt Alice den öffentlichen D-H Schlüssel von Bob und umgekehrt (Einen vereinfachten Ablauf sehen Sie in Abbildung 2.3). Um MitM Attacken abwehren zu können müssen sich die Gesprächspartner authentifizieren können. Dazu werden während des AKE die Daten mit einem langlebigen DSA-Schlüssel signiert. Dieser wird nicht für die Verschlüsselung verwendet sondern nur für die Authentifizierung [vgl. 14]. Wie diese bewerkstelligt wird erklärt Abschnitt 2.3.4.

Möchte nun Alice eine geheime Botschaft an Bob senden, generiert sie aus ihrem und seinem Schlüssel ein gemeinsames Geheimnis (K) und verschlüsselt die Nachricht unter Zuhilfenahme von Advanced Encryption Standard (AES), bevor sie zusammen mit einem neuen D-H-Schlüssel versendet wird. Bob kann nun die Nachricht entschlüsseln und seinerseits mit dem neuen Schlüssel eine Mitteilung senden. In Tabelle 2.4 ist die komplette Struktur einer OTR Nachricht abgebildet.

2.3.4 Socialist millionaire protocol (SMP)

Um beim Chatten sicherzugehen, dass der Gesprächspartner auch derjenige ist, welcher er vorgibt zu sein, muss man seinen Partner authentifizieren. Somit schließt man einen MitM-Angriff oder auch die Unterhaltung mit einem Dritten aus. Um den Gesprächspartner zu verifizieren, kann man manuell die Richtigkeit des Schlüssels seines Partners kontrollieren, oder das SMP [5] benutzen. Dabei wird über ein gemeinsames Geheimnis, welches nicht übertragen wird, die Identität des Partners geprüft.

Als Grundlage für dieses Protokoll dient das „millionaires's problem" [20], bei welchem es dar-

PR	Protokol Version
MT	Nachrichten Type
ST	Sender Tag
ET	Empfänger Tag
F	Flags
SK	Sender Schlüssel-Id
EK	Empfänger Schlüssel-Id
DH	Nächster DH-Schlüssel
C	Zähler, welcher bei jeder Nachricht inkrementiert wird
EM	Verschlüsselte Nachricht
A	Authentifizierung (HMAC)
R	Alte MAC Schlüssel

Tabelle 2.4: Struktur einer OTR Nachricht. Sender und Empfänger Tag sind die jeweilige Ressourcen der Jid. Dadurch, dass die alten schon verwendeten MAC Schlüssel offengelegt werden, kann im Nachhinein jeder eine OTR Nachricht fälschen, wodurch später niemand die Richtigkeit einer Aussage beweisen kann („Deniability“).

um geht, dass zwei Personen wissen wollen wer denn reicher ist ohne ihr Vermögen zu offenbaren. Eine Variante davon ist das „socialist millionaires' problem“ in dem beide Millionäre sich damit begnügen, zu wissen, ob sie gleich viel Geld besitzen. In unserem Szenario heißen diese Personen Alice und Bob, welche einen Hash aus ihren Fingerprints und einem gemeinsamen Geheimnis vergleichen wollen. Dabei wird nicht direkt der Hash-Wert verglichen, ansonsten bräuchte man das SMP nicht, sondern daraus resultierende Berechnungen. Das Verfahren ist angelehnt an den oben erläuterten Diffie-Hellmann Schlüsselaustausch und in Abbildung 2.4 dargestellt.

Nachdem Alice und Bob in Besitz von R_{ab}, P_a und P_b sind und wissen dass folgendes gilt:

$$\begin{aligned} R_{ab} &= \left(\frac{Q_a}{Q_b}\right)^{a_3 b_3} \\ &= (g_1^{a-b} g_2^{x-y})^{a_3 b_3} \\ &= g_3^{a-b} g_2^{(x-y)a_3 b_3} \\ &= \left(\frac{P_a}{P_b}\right) (g_2^{a_3 b_3})^{x-y} \end{aligned}$$

Können Sie sichergehen, dass $R = \left(R_{ab} \cdot \frac{P_b}{P_a}\right) = 1$ nur gilt wenn $x = y$, ansonsten wäre R eine zufällige Zahl. [vgl. 2]

2.3.5 Bidirectional-streams Over Synchronous HTTP (BOSH)

Da das Internet nach dem Anfrage/Antwort-Prinzip erstellt worden ist, bei XMPP aber eine bidirektionale Verbindung benötigt wird, muss es eine Brücke zwischen diesen Techniken geben. So eine Brücke ist zum Beispiel BOSH [15]. Dabei wird eine Verbindung so lange gehalten, bis der Server eine Nachricht zu senden hat. Hiermit ist eine Simulation einer bidirektionaler Verbindung möglich. Diese Technik nennt man polling bzw. long-polling.

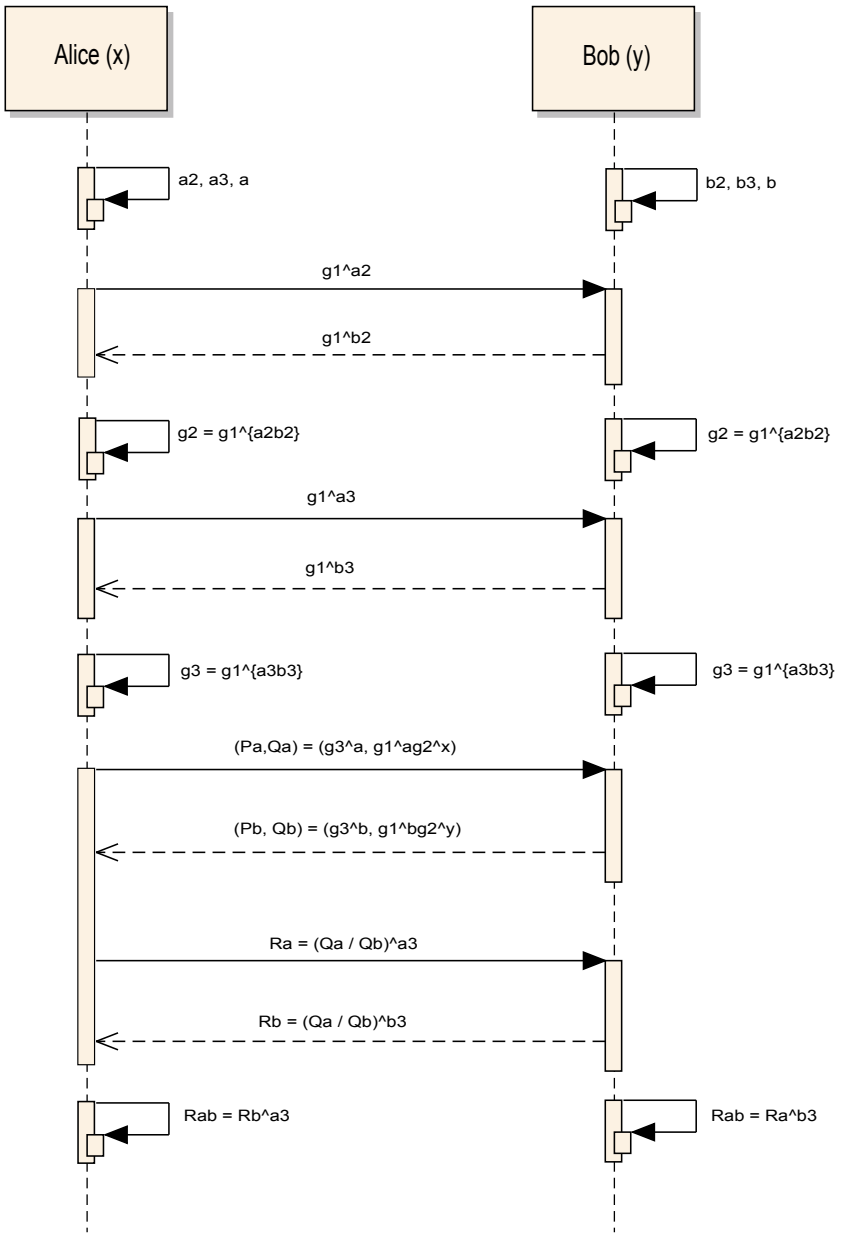


Abbildung 2.4: Ablauf einer SMP Aushandlung mit dem Ziel zu erfahren ob $x = y$ ohne x oder y zu übertragen. a_2, a_3, a, b_2, b_3 und b sind geheimgehaltene Zufallszahlen. Zum Austausch der Generatoren g_2 und g_3 findet jeweils ein D-H-Schlüsselaustausch statt.

3 Entwurf

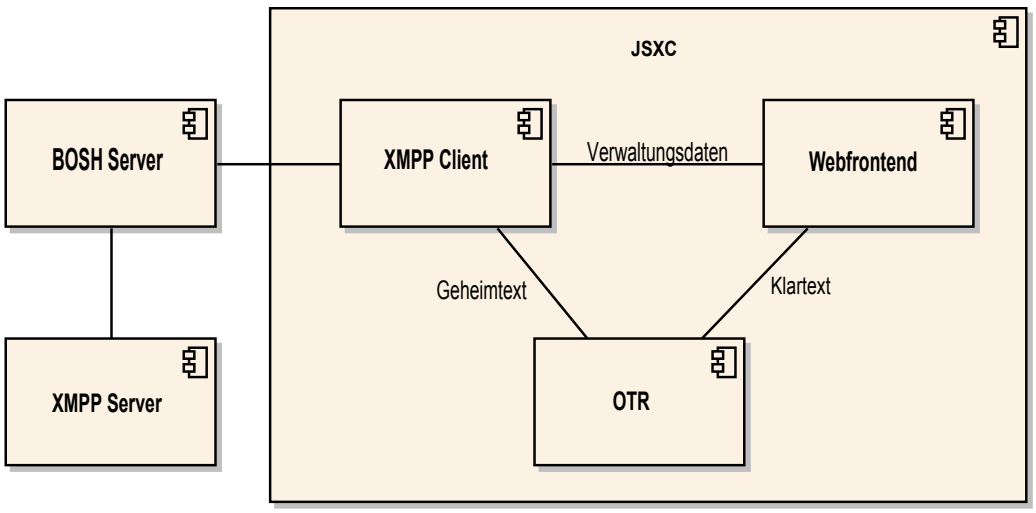


Abbildung 3.1: Komponentenübersicht

3.1 Frontend

Um dem Benutzer nicht das Gefühl zu geben es handele sich bei der Chat-Anwendung um einen Fremdkörper, ist eine nahtlose Integration gepaart mit bekannten Bedienkonzepten entscheidend.

Aus diesen Gründen besteht das Frontend aus zwei Elementen. Zum Einen die Kontaktliste auf der rechten Seite des Bildschirms, und zum Anderen die Liste aller geöffneten Konversationen am unteren Rand. Beide Elemente sind in den Farben des bestehenden Systems gehalten. In jedem Bereich wird neben dem Kontaktnamen durch verschiedene Farben der Zustand des Kontaktes visualisiert (online, abwesend, offline).

Die Liste aller Kontakte ist scrollbar und besitzt am unteren Rand eine Schaltfläche zum Hinzufügen von neuen Kontakten. Beim Überfahren der einzelnen Namen werden die Optionen zu dem Kontakt sichtbar (siehe 3.3(a)). Hier kann der Name geändert oder komplett aus der Liste gelöscht werden. Zudem sind Kontakte mit einer ausstehenden Freundschaftsanfrage blasser dargestellt und mit einem Tooltip versehen. Um eine Ordnung innerhalb der Liste zu erreichen sind die Kontakte sortiert. Ganz oben die Gruppe derer die online sind, danach die abwesenden und darauf folgend die Offline-Kontakte. Innerhalb dieser Gruppen wurde eine alphabetische Ordnung beibehalten.

Jede geöffnete Konversation ist als eigene Box sichtbar. Bei eingehenden Nachrichten blinken diese kurz auf um den Empfang zu signalisieren. Im minimierten Zustand zeigt eine farbliche Markierung ungelesene Nachrichten an (siehe 3.3(b)). Eine extra Schaltfläche visualisiert durch Farben den Zustand der Verbindung (unverschlüsselt, verschlüsselt, verifiziert), siehe 3.4. Durch ein Drop-Down Menü kommt man zu den Optionen für die Authentifizierung, dem Verbindungsstatus und der Anzeige der Fingerprints.

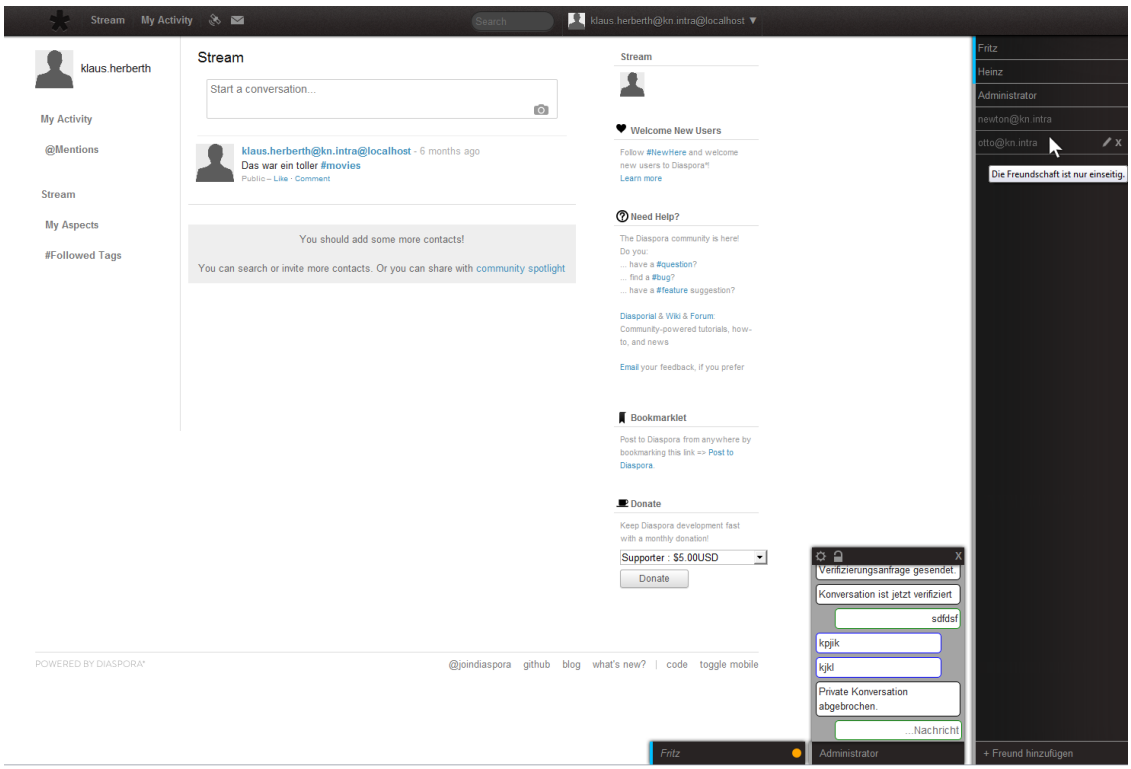
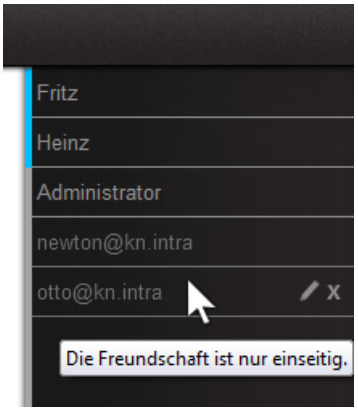
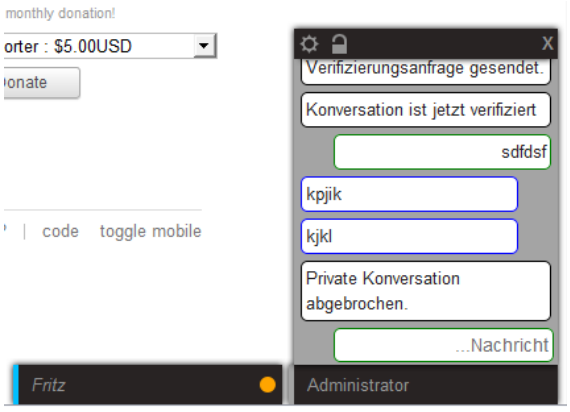


Abbildung 3.2: JSXC integriert sich in das bestehende Diaspora Theme.



(a) Beim Überfahren werden die entsprechenden Optionen sichtbar.



(b) Ungelesene Nachrichten werden mit einem orangenen Punkt gekenntzeichnet.

Abbildung 3.3: Detailansicht des Rosters und der Chat-Fenster

(a) Unverschlüsselt.

(b) Verschlüsselt.

(c) Verschlüsselt und Authentifiziert.

Abbildung 3.4: Zustände des Nachrichtenstatus.

3.2 Backend

Die generelle Funktionsweise von JSXC ist wie folgt: Zuerst kontrolliert die Anwendung ob mehrere offene Tabs vorhanden sind. Danach wird überprüft, ob eine Verbindung wiederhergestellt werden kann, oder ob eine neue aufgebaut werden muss. Wenn eine Verbindung steht, werden eingehende Nachrichten dargestellt und ausgehende ggf. verschlüsselt und versendet.

Als Verbindungsprotokoll zum XMPP-Server wird BOSH (Abschnitt 2.3.5) benutzt mit einer zu Diaspora unabhängigen Implementierung. Dadurch kann JSXC in jede beliebe Webanwendung eingefügt werden.

Alle Informationen die persistent über die Lebensdauer einer einzelnen Seite hinaus verfügbar sein müssen, werden im Web Storage gespeichert. Wie zum Beispiel die letzten 10 Nachrichten eines jeden Benutzers, der private Schlüssel[1] oder Informationen[2] über den Status des OTR Protokolls.

3.2.1 Interne Kommunikation

Mit dem Begriff „interne Kommunikation“ ist die Kommunikation der einzelnen Tabs untereinander gemeint. Diese geschieht über den Web Storage, denn dieser feuert in jeder Instanz ein Event wenn ein Wert geändert wird. Auf dieses Event kann man lauschen und ggf. darauf reagieren. Somit muss Tab A nur einen Wert in den Storage schreiben, um mit den Anderen zu kommunizieren. (Abbildung 3.5)

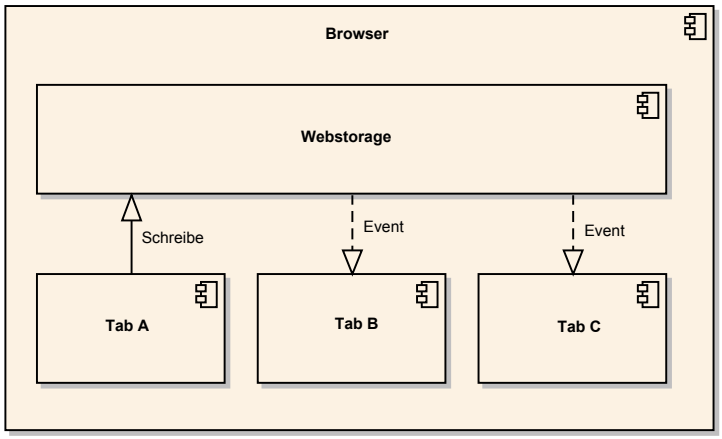


Abbildung 3.5: Funktionsweise des Webstorage und dessen Events.

3.2.2 Rollenbestimmung

Von mehreren geöffneten Tabs ist immer einer für die Kommunikation mit dem XMPP-Server zuständig, diesen nennen wir „Master“. Alle anderen „Slave“, diese kommunizieren mit dem Haupt-Tab. Wenn eine Seite neu geladen wird, muss zuerst überprüft werden, ob ein Master existiert oder ob man selbst diese Rolle übernehmen muss. (Abbildung 3.6)

Danach sendet der Master in einem festen Intervall ein Keep-Alive Signal, damit alle Slaves wissen, dass er noch nicht geschlossen wurde. Sollte das Signal ausbleiben[3] muss ein neuer Master bestimmt werden. Dazu warten die Tabs eine zufällige Zeit, welche kleiner drei Sekunden ist und starten danach mit dem Senden eines Pings. Da Ping, Pong und das Keep-Alive Signal auf dem

[1] In Kapitel 5.3.4 werden die dazugehörigen Sicherheitsaspekte beleuchtet.
[2] Einzelheiten werden in Abschnitt 3.2.5 behandelt.
[3] Der Master-Tab wurde geschlossen.

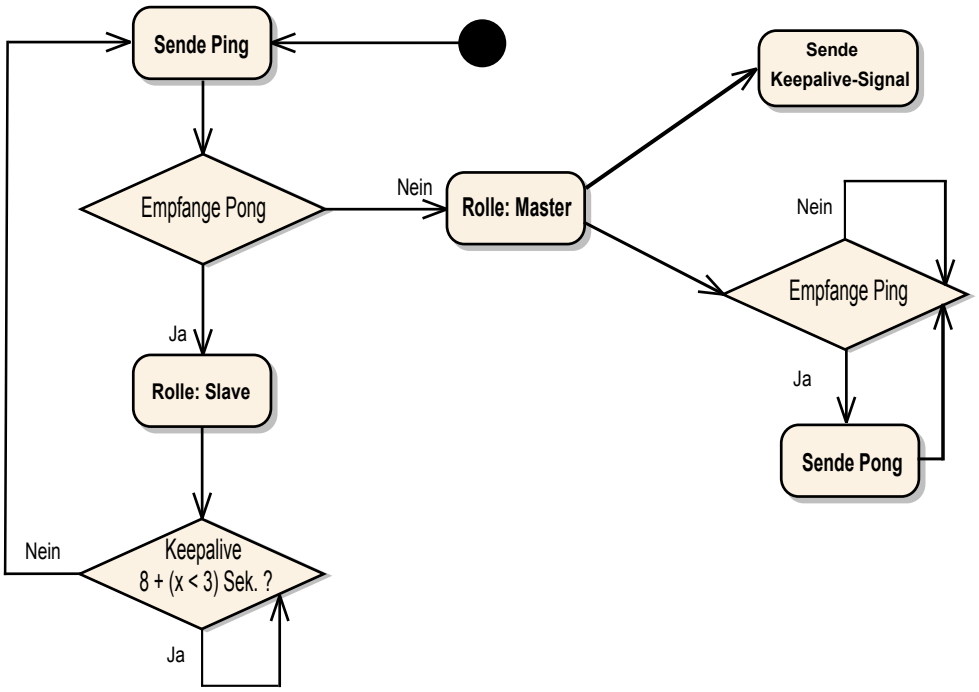


Abbildung 3.6: Rollenbestimmung nach dem Laden der Seite.

gleichen Schlüssel im Webstorage realisiert sind, ist sichergestellt, dass nur ein Tab der neue Master wird. Denn nach dem ersten Ping des schnellsten Tabs denken alle anderen, dies sei ein Keep-Alive Signal und gehen in den normalen Slave-Betrieb.

3.2.3 Verbindungsaufbau

Um eine neue Verbindung aufzubauen wird die Jid und das entsprechende Passwort benötigt. Damit JSXC sich in Diaspora integriert, schaltet es sich in den Anmeldeprozess ein. Das bedeutet, wenn sich der Benutzer anmelden möchte, werden diese Daten zuerst für die Anmeldung am XMPP-Server benutzt und erst danach wird die eigentliche Anmeldung durchgeführt. Alle Daten für den Wiederaufbau werden im Web Storage gespeichert. Dies sind bei BOSH eine Session-Id (Sid) und eine Request-Id (Rid). Bei jedem Austausch zwischen Server und Client wird die Rid inkrementiert. Damit wird die Wahrscheinlichkeit einer fremden Übernahme der Verbindungsparameter minimiert. Durch das Abgreifen der Anmeldedaten bleibt JSXC unabhängig von der eigentlichen Anwendung und kann dadurch auch in anderen Webseiten eingesetzt werden. (Abbildung 3.7)

3.2.4 Verbindungswiederherstellung

Sollte der Master Daten zur Wiederherstellung einer Verbindung im Speicher vorfinden wird die Kommunikation mit dem Server fortgesetzt und die Benutzeroberfläche wiederhergestellt. Das erlaubt, dass der Benutzer dort weitermachen kann wo er vor der Seitennavigation aufgehört hat. (Abbildung 3.8)

3.2.5 Verschlüsselung

Erhält der Benutzer eine Anfrage für eine private Unterhaltung oder möchte selbst eine initiieren, wird diese automatisch gestartet und alle Parameter zur Wiederherstellung werden gespeichert. Dazu gehören unter anderem der aktuelle und frühere D-H-Schlüssel, Zustände des OTR und SMP, die beim AKE ausgetauschte „secure session id“ und einige weitere Eigenschaften. All dies

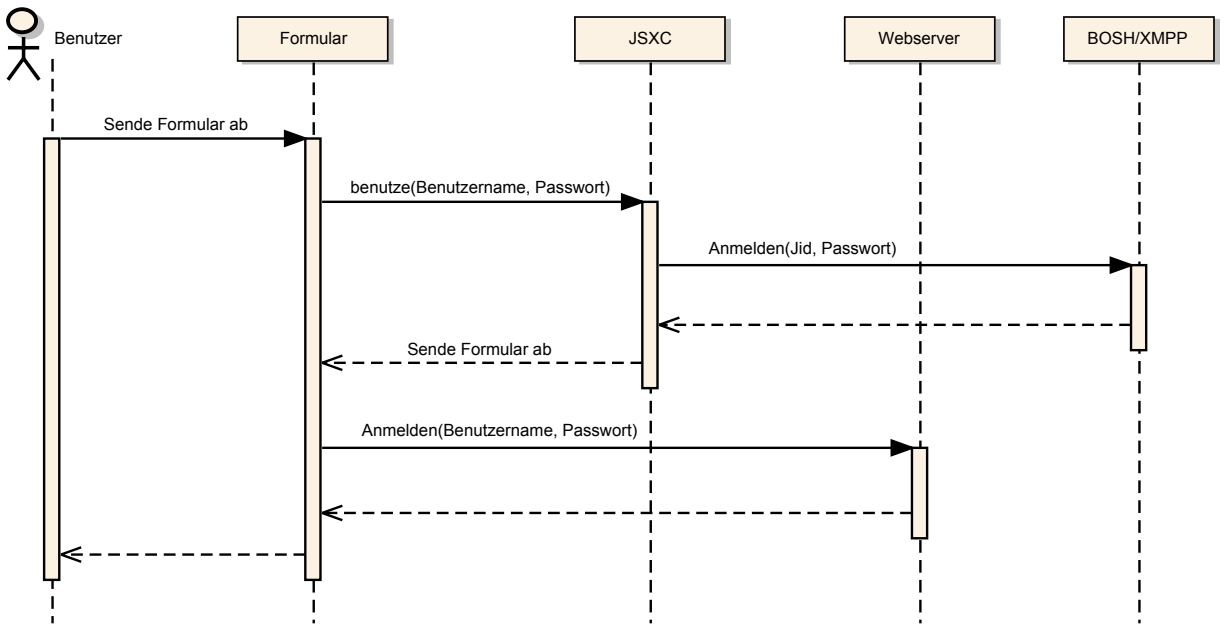


Abbildung 3.7: Ablauf eines Verbindungsaufbaus. Zuerst werden die Informationen des Anmelde-Formulars benutzt um den Benutzer am XMPP-Server anzumelden. Erst danach geschieht die Anmeldung am eigentlichen Webserver.

ermöglicht bei einem Neuladen der Webseite eine private Unterhaltung fortzusetzen. (Abbildung 3.9)

3.2.6 Verbindungsabbau

Um der Integration von JSXC auch beim Verbindungsabbau treu zu bleiben, wird dieser beim Logout oder nach einem Timeout von 10 Min. vorgenommen. Dazu klinkt sich der Chat zwischen die Diaspora Abmeldung und meldet sich zuerst beim Chatserver ab. Da nicht jeder Benutzer die Logout Funktion verwendet ist ein Timeout nötig.

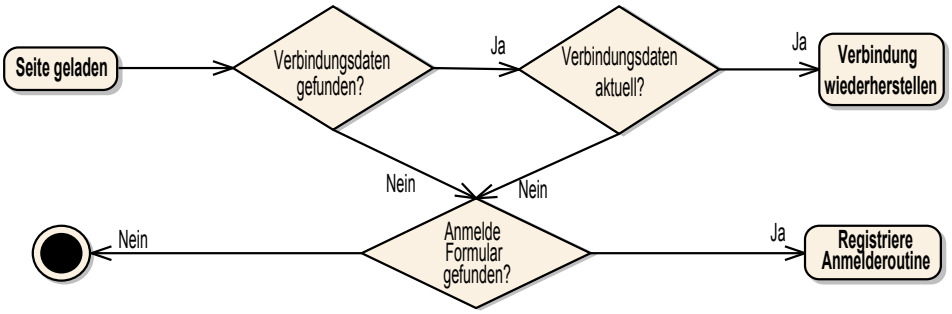


Abbildung 3.8: Verbindungswiederherstellungs-Routine des Masters.

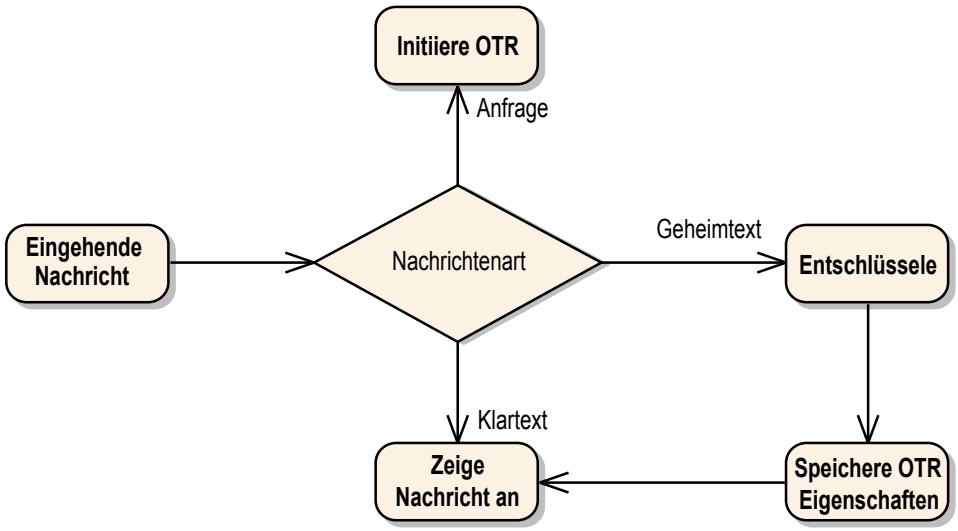


Abbildung 3.9: Routine beim Empfangen von Nachrichten.

4 Implementierung

4.1 Werkzeuge & Umgebung

Für die Entwicklung wurde ein kompletter Linux-Apache-MySQL-PHP (LAMP) Server in einer virtuellen Maschine aufgesetzt. Als Host-System diente Windows 7 und als Gast Debian „Wheezy“. Dort wurde auch Netbeans in Version 7.3 und Openfire in Version 3.8 installiert. Dass die Wahl des XMPP Servers auf Openfire fiel hat mit dessen einfacher Konfigurationsoberfläche zu tun. Des Weiteren wurde der aktuelle Stand des Master Branches aus dem DiSy Diaspora Repository[1] in einer RoR 3.2.x Umgebung installiert. Damit man sich in Diaspora auch authentifizieren kann, wurde noch ein OpenLDAP Server aus den offiziellen Paketquellen von Debian hinzugefügt.

Zusammenfassend kann man sagen, dass die Installation des LAMP Servers ohne Probleme von statten ging, dafür diejenige von Diaspora durch viele Paketabhängigkeiten[2] und eine mangelhafte Installationsanweisung um so mehr Probleme verursachte.

Für die Fehlersuche wurde hauptsächlich der Browser Opera wegen seiner Entwicklungswerkzeuge gewählt.

4.2 Frontend

Um die Wartbarkeit der Anwendung zu wahren wurde auf eine strikte Trennung von Code, Design und Struktur geachtet. Dazu sind alle HTML-Strukturen, wie Roster oder Fenster, an einem zentralen Ort gespeichert (siehe 4.4.7) und Formatierungen in einer CSS-Datei ausgelagert. Dies hat nicht nur den Vorteil der Übersichtlichkeit, sondern ermöglicht auch das einfache und schnelle Anpassen des Templates.

Für die Konsistenz der Oberfläche wurde die von Diaspora verwendete Dialogbox-Bibliothek (für ein Beispiel siehe Abbildung 4.1) mitverwendet. Dabei wurde allerdings darauf geachtet, dass diese Bibliothek jederzeit durch eine andere ersetzt werden kann. Erreicht wurde dies durch einen eigenen Wrapper für Dialogboxen.

[1] `https://github.com/disy/diaspora`
[2] Bei Ruby werden Pakete „Gems“ genannt.

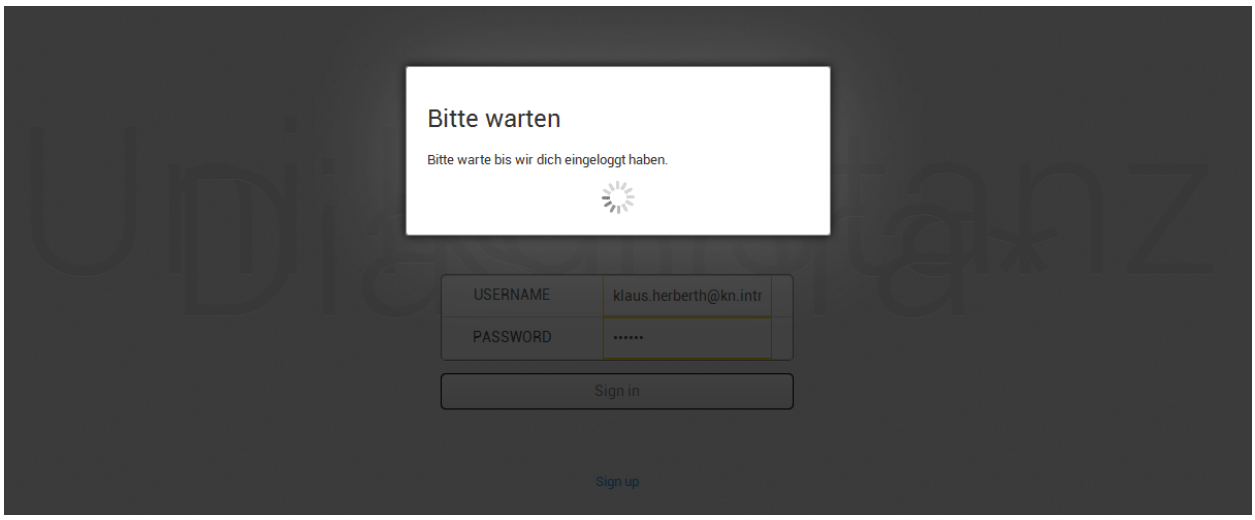


Abbildung 4.1: Beispiel einer Dialogbox.

4.3 Bibliotheken

4.3.1 jQuery

Abschnitt 5.1 geht auf die Unterschiede der einzelnen Browser ein, so wie die Tatsache, dass sie JavaScript unterschiedlich interpretieren. Dieses Problem wird durch die Benutzung einer Bibliothek oder in diesem Kontext auch Framework genannt, reduziert. Zudem werden viele alltägliche Aufgaben und die umständliche JS Syntax vereinfacht. Die mit 91,8%[3] Marktanteil weit verbreitetste ist jQuery[4]. Dies hat den Vorteil vieler Plugins, die den Funktionsumfang der Bibliothek erweitern und einer großen, hilfreichen Community die bei Problemen hilft.

4.3.2 StropheJS

Diese Bibliothek implementiert die Kommunikation mit dem BOSH und damit auch dem XMPP Server. Durch sie werden XML (Extensible Markup Language) Stanzas erstellt, gesendet und empfangen. Ebenso wie jQuery ist StropheJS[5] weit verbreitet und gibt dadurch Sicherheit in Bezug auf Wartung und Unterstützung.

4.3.3 OTR in JavaScript

Anders als bei den vorherigen Bibliotheken gibt es nur eine, welche OTR in JS implementiert. Anfangs beinhaltete der Code noch einige Fehler, welche aber rasch beseitigt und Upstream eingebracht werden konnten.

4.4 Namespaces (NSs)

Da in einem Projekt wie Diaspora verschiedenste JS Skripte parallel laufen, ist eine Organisation in NS's unerlässlich. Als Haupt-Namespace wurde der Namen der Anwendung JSXC verwendet, welcher zur besseren Organisation, wiederum in mehrere Unter-Namespaces eingeteilt wurde.

[3] `http://w3techs.com/technologies/overview/javascript_library/all`, Stand: 18.08.2013
[4] `http://jquery.com/`
[5] `http://strophe.im/strophejs/`

4.4.1 jsxc

Der Haupt-Namespace beinhaltet zentrale Funktionen der Anwendung. Unter anderem die Initialisierungsfunktion, die Routine zur Bestimmung des Masters und einige Hilfsfunktionen.

4.4.2 jsxc.options

Dieser Abschnitt beinhaltet alle Einstellungen der Anwendung. Für manche sind Grundeinstellungen vorgegeben, wie zum Beispiel für die Sprache. Andere müssen beim Programmstart explizit mit angegeben werden, denn es wäre nicht sinnvoll, unter anderem eine Jid fest im Quellcode zu verankern.

4.4.3 jsxc.gui

Funktionen, welche die gesamte Benutzeroberfläche betreffen oder nicht in einen der Unter-Namespaces passen, wurden in jsxc.gui untergebracht. Beispielsweise Funktionen die Dialoge oder Warnhinweise einblenden.

4.4.4 jsxc.gui.roster

Aufgaben welche den Roster manipulieren, wie zum Beispiel das Einfügen, Ändern und Löschen von Freunden, beginnen mit jsxc.gui.roster.

4.4.5 jsxc.gui.dialog

Der Adapter für die Dialogbox-Bibliothek ist gruppiert unter jsxc.gui.dialog. Er besteht nur aus einer Funktion zum Öffnen und einer zum Schließen.

4.4.6 jsxc.gui.window

Neben dem Roster am rechten Bildschirmrand hat jede Unterhaltung noch ein eigenes Fenster. Aktionen wie Schließen oder Minimieren sind diesem Bereich zugeordnet.

4.4.7 jsxc.gui.template

Um alle HTML (Hypertext-Markup-Language) Fragmente an einem zentralen Ort zu verwalten, wurde der Namespace jsxc.gui.template geschaffen. Er beinhaltet des Weiteren auch eine Funktion welche Variablen in den Fragmenten ersetzt. Wie zum Beispiel die Übersetzung in die gewählte Zielsprache.

4.4.8 jsxc.xmpp

Ereignis-Handler die in Bezug zur Kommunikation mit dem XMPP Server stehen oder Funktionen die komplexere Nachrichten an diesen schicken, wurden unter jsxc.xmpp gebündelt. Dabei findet die Verbindung nicht direkt über diesen NS statt, sondern über die entsprechende Bibliothek (siehe 4.3.2).

4.4.9 jsxc.storage

Da die Browser Application programming interface (API) für den Local-Storage minimalistisch ist, wurde für diese ein Wrapper erstellt. Dadurch wurden Aufgaben, wie das Bearbeiten von gespeicherten JS Objekten oder das Inkrementieren von Zahlenwerten an einer zentralen Stelle ausgelagert. Auch wird durch den Wrapper sichergestellt, dass auch im Local-Storage der NS „jsxc“ verwendet wird.

4.4.10 jsxc.otr

Schlüsselgenerierung, Verschlüsselungsanfragen und Verarbeitung u.s.w. werden hier behandelt.

4.4.11 jsxc.l10n

Da die Anwendung multilingual sein sollte befinden sich hier alle Übersetzungen.

5 Evaluation

In der Evaluation dieser Anwendung wird auf drei Hauptpunkte eingegangen. Als erstes auf den Vergleich verschiedener Browser, danach wie sich der Chat in Kontrast zu einer gewöhnlichen Desktop Anwendung verhält und zu guter Letzt betrachten wir die Sicherheit von JSXC.

5.1 Browser im Vergleich

Nicht jeder Browser verhält sich gleich, denn fast Jeder hat seine eigene HTML-Rendering-Engine. Mit Hilfe dieser wird HTML Code dargestellt und JS ausgeführt. Da die verschiedenen Hersteller verschiedene Schwerpunkte setzten sind teilweise auch unterschiedliche Funktionen verfügbar. Erschwerend kommt hinzu, dass Spezifikationen nur sehr langsam fertiggestellt werden und einige Browserhersteller diese im Entwicklungsstadium schon implementieren. Das hat natürlich den Vorteil, dass neue Funktionen schneller verfügbar sind. Dafür muss man gravierende Veränderungen und teilweise inkonsistente APIs in Kauf nehmen. Aus diesen Gründen ist ein Vergleich und Test der unterschiedlichen Engines sinnvoll.

Auf dem Prüfstand stehen Google's Engine „Blink", „Gecko" von Mozilla, „Webkit" von Apple und Microsoft's „Trident". „Presto" von Opera wurde außen vorgelassen, da die Entwicklung eingestellt wurde. Auch wurde von Tests der Engine „KHTML" von der KDE Software Compilation, wegen geringer Verbreitung, abgesehen.

Da heutzutage mobile Browser fast genau so weit verbreitet sind wie ihre Desktop-Verwandten, wurde nicht nur Chrome 28, Firefox 23, Safari 5[1] und der Internet Explorer 10 untersucht sondern auch zwei weit verbreitete Android Versionen: Dolphin Browser und Opera Mobile.

Alle Tests liefen unter Windows 7 mit einem „Intel® Core™ i5-2500K" und Android 2.3.6 mit einer 800 Megahertz CPU.

5.1.1 Funktionsumfang

Um den Funktionsumfang der einzelnen Browser zu überprüfen wurde Tabelle 5.1 abgearbeitet und jeweils notiert, ob alles wie gewünscht funktioniert. Im Einzelnen wurde kontrolliert ob die Anmeldung am XMPP-Server funktioniert, der Roster lädt und man Kontakte umbenennen kann. Weitere Punkte waren das Öffnen eines Chat-Fensters, die Generierung eines neuen Schlüssels und das Versenden und Empfangen von verschlüsselten sowie unverschlüsselten Nachrichten. Die Tauglichkeit bei mehreren geöffneten Tabs wurde unter dem Punkt Multi-Tab geprüft und die Authentifizierung unter SMP. Der letzte Punkt bewertet in wie weit die Anwendung in der Realität zu verwenden ist, dazu zählt die Funktionsweise, Größe und Positionierung aller Elemente. Dies ist besonders bei mobilen Endgeräten wichtig, welche in Tabelle 5.1 mit einem Stern (*) gekennzeichnet sind.

Wenn man die Ergebnisse betrachtet kann man zusammenfassend sagen, dass alle getesteten Desktop-Varianten wie gewünscht funktionieren. Ausgenommen der Internet Explorer (IE) hat Probleme bei mehreren geöffneten Tabs, denn er verhält sich bei Storage Events anders als alle anderen Browser. Er löst das Event in allen Tabs aus und nicht nur in denen die über eine Änderung informiert werden müssten. Da dieser Trigger elementar für die interne Kommunikation ist (siehe 3.2.1) bricht die selbige zusammen.

[1] Safari 6 ist leider momentan noch nicht für Windows verfügbar.

	Chrome	Firefox	Safari	I-Explorer	Dolphin*	Opera*
Anmelden	✓	✓	✓	✓	✓	✓
Roster laden	✓	✓	✓	✓	✓	✓
Umbenennen	✓	✓	✓	✓		
Chat öffnen	✓	✓	✓	✓	✓	✓
Schlüssel	✓	✓	✓	✓	✓	✓
Klartext	✓	✓	✓	✓	✓	(✓)
Geheimtext	✓	✓	✓	✓	✓	(✓)
Multi-Tab	✓	✓	✓			
SMP	✓	✓	✓	✓		
Benutzbar	✓	✓	✓	✓		

Tabelle 5.1: Funktionsübersicht aller getesteten Browser.

Wie zu erwarten war, ist die Unterstützung für mobile Browser unzureichend. Bei Opera Mobile war nicht einmal das Senden von Nachrichten möglich. Was aber auch nicht verwundert, da JSXC für Geräte mit großem Bildschirm optimiert wurde, denn auf kleinen würde ein zusätzlicher Bereich für einen Chat nur stören. Erschwerend kommt hinzu, dass kryptografische Funktionen viel Rechenleistung benötigen, welche bei mobilen Geräten tendenziell geringer ist.

Zusammenfassend kann man sagen, dass die Unterstützung der Zielgruppe wie erwünscht ausfällt.

5.1.2 Geschwindigkeit

Um die Geschwindigkeit aller Browser zu testen eignet sich die Laufzeit der komplexen Berechnung des DSA Schlüssels am Besten. Aus diesem Grund wurde in allen Umgebungen fünfmal ein Schlüssel berechnet und die benötigte Zeit in ms gestoppt. Um einen Vergleichswert zu erhalten wurde der gestutzte Mittelwert benutzt, welchen man erhält, indem man das Min- und Maximum ignoriert und aus der restlichen Menge wie gewohnt das arithmetische Mittel zieht. Diese Variante schien sinnvoll, da die Zeiten stark variierten wie in Tabelle 5.2 zu sehen ist.

Auch hier fallen die Ergebnisse nicht weiter überraschend aus. In erster Linie hängt die Dauer von der Rechenleistung des entsprechenden Prozessors ab und da der des Huaweis um einiges schwächer ist, als der des PC's sind die Zeiten auch um ein vielfaches höher. An den Laufzeiten erkennt man auch, dass die Entwicklung an den JS-Engines schnell voran geschritten ist. So ist Rechenleistung des älteren Safaris 5 weniger als halb so hoch wie die des 3. platzierten Chrome.

5.2 Browser im Vergleich zur Desktop-Anwendung

Der Hauptunterschied liegt wohl in der Verfügbarkeit der Anwendungen. So muss die klassische Variante zuerst heruntergeladen und danach installiert werden, bei der Browser-basierten muss

	Chrome	Firefox	Safari	I-Explorer	Dolphin*	Opera*
d_1	3589	827	7698	3953	170432	86755
d_2	4749	1216	11186	1935	123753	219909
d_3	2477	7789	89555	3755	146768	154230
d_4	6538	924	7724	3217	189322	124819
d_5	3899	5117	8124	1447	115780	169908
$\bar{d}_g$	4079	2419	9011	2969	146984	149652

Tabelle 5.2: Zeit in ms um einen DSA Schlüssel zu generieren und das gestutzte Mittel des jeweiligen Browsers.

man nur eine Internetseite aufrufen. Beides bringt seine Vor- und Nachteile mit sich: Die Desktop-Version benötigt zusätzlich zu dem meist geöffneten Browser Ressourcen und verursacht einen gewissen Mehraufwand durch die Installation. Dadurch erkauft man sich aber ein Stück weit mehr Sicherheit (siehe 5.3 Programmsicherheit). Der Konkurrent hingegen bringt für den Betreiber und Benutzer einige Vorteile: Als da wären die längere Verweildauer auf der entsprechenden Betreiberseite, was einher geht mit einem Mehr von Werbeeinnahmen und eine höhere Kundenbindung. Für den Benutzer oder Kunden entfällt selbstverständlich die Installation der Anwendung und somit kann er auf jedem Computer die selbe Umgebung benutzen. Auch die Nachteile einer klassischen Webanwendung, wie dass Informationen nur übermittelt werden können, wenn die Webseite im Vordergrund ist, können durch HTML5 beseitigt werden (siehe Zusammenfassung & Ausblick 6).

Für einen direkten Vergleich wurde Firefox 23 und Pidgin 2.10.6 gewählt. Die Wahl für die Desktop Anwendung fiel auf Pidgin[2], da sie auf allen Betriebssystemen verfügbar und weit verbreitet[3] ist.

5.2.1 Geschwindigkeit

Die Geschwindigkeit wurde in drei Kategorien gemessen: Senden/Empfangen von Klartext-/Geheimtext-Nachrichten und Generierung eines Schlüssels. Die Dauer für die Erstellung des Schlüssels wurde bei Pidgin mittels Stoppuhr ermittelt und betrug 4 Sekunden. Damit fast 70 Prozent langsamer als der schnellste Browser im vorangegangenen Test mit 2,4 Sekunden. Wie entscheidend die Implementierung des Algorithmus ist, zeigt sich daran, dass Jitsi[4], ein weiterer Desktop Client, für die selbe Aufgabe einen Wert von unter einer Sekunde liefert.

Die Geschwindigkeit beim Senden und Empfangen von verschlüsselten sowie unverschlüsselten Nachrichten glich sich bei beiden Anwendungen. Alle Kombinationen blieben unter einer Sekunde. Um diese Werte zu ermitteln wurden die Zeitstempel in Sekunden der Debug-Meldungen ausgewertet, als Nachrichten zwischen Pidgin und JSXC versendet wurden. Da die Zeiten im unteren Ende des Messfensters liegen, war eine genauere Zeitmessung beim Transport der einzelnen Anwendung mit dieser Messmethode unmöglich. Die weitere Untersuchung ist in meinen Augen auch unnötig, bedingt durch die absolut zufriedenstellende Verzögerung von unter einer Sekunde.

[2] `http://www.pidgin.im/`

[3] Ermittelt von `http://www.mytopdozen.com/Best_XMPP_Client_Software.html` mittels verschiedener Kriterien, wie der Beliebtheit im Netz.

[4] `https://jitsi.org/`

Vergleicht man alle Geschwindigkeiten sind sich beide Anwendungen ebenbürtig.

5.2.2 Handling

Da sich die Webanwendung an den Gestaltprinzipien einer Desktop-Anwendung orientiert, sind beide Programme ähnlich in der Bedienung. Man kann Chatfenster minimieren oder schließen und den Roster verstecken, indem man das Browser-Fenster verkleinert. Auch sind die Begriffe und Symbole angelehnt an bekannte Konzepte.

5.2.3 Funktionsumfang

JSXC bietet einen eher minimalistischen Funktionsumfang, da die Entwicklung noch ganz am Anfang steht. So haben stärker ausgereifte Produkte eine Vielzahl von Eigenschaften die JSXC noch fehlen. Es gibt verschiedene Benachrichtigungen, Dateiversand, Hervorhebung von Links, Audio/Video (A/V) Gespräche, Multi-User-Chat (MUC), vCards, Buddy-Alarme, Gruppen innerhalb des Rosters, unterschiedliche Einstelloptionen und vieles mehr was eine Desktop-Anwendung JSXC voraus hat. Durch die einfache Erweiterbarkeit von JSXC dürfte sich das in Zukunft aber ändern. So gibt es schon ein Plugin für A/V-Gespräche, siehe Zusammenfassung & Ausblick 6.

5.3 Programmsicherheit

Dieses Kapitel beleuchtet verschiedenste Sicherheitsaspekte des Chats und deckt auf, wo noch Handlungsbedarf besteht oder inwiefern man sich schützen kann. Dabei wird von einer sauberen Umgebung ausgegangen. Das bedeutet es werden keine Sicherheitslücken in den Browsern oder Schadsoftware, wie zum Beispiel Keylogger, auf dem benutzen PC behandelt.

5.3.1 Zufallszahlen

Für jede kryptographische Anwendung sind Zufallszahlen wichtig, wie sie ein kryptographisch sicherer Zufallsgenerator (engl. cryptographically secure pseudo-random number generator (CSPRNG)) erzeugt. Glücklicherweise haben alle Browser in ihrer neusten Version einen solchen Generator, so dass die Zufälligkeit ausreichen sollte. Weitere Verbesserungen könnte man einbauen indem man mehr oder weniger zufällige Ereignisse mit in die Berechnung der Zahl aufnimmt. Wie zum Beispiel Verweildauer auf einer Seite, die Mausbewegung oder die Zeit zwischen zwei Tastenanschlägen. Auch wäre es denkbar einen externen Service wie `random.org` zu verwenden, welcher atmosphärisches Rauschen als Grundlage für Zufallszahlen verwendet. Auch wenn sich das erst einmal gut anhört, wäre man von einem Drittdienst abhängig und niemand kann garantieren, dass der Service nicht manipuliert ist. Aus diesem Grund bleibt der Web-Chat erst einmal bei den Generatoren der Browser.

5.3.2 Code injection

Mit dem Begriff „code injection“ meint man das Einschleusen von Fremdcode in die eigene Anwendung. Diese Gefahr besteht für jede Anwendung, ist für den verschlüsselten Chat aber besonders gefährlich. Denn was bringt die beste verschlüsselte Übertragung, wenn die Inhalte im Klartext vom Bildschirm abgelesen werden können? Dies ist prinzipiell durch JS möglich, da keinerlei Kapselung besteht. Das heißt, jeder ausgeführte Code auf einer Seite kann auf alle Elemente der selbigen zugreifen, was natürlich auch für den Chatverlauf gilt.

Um die Auswirkungen so gering wie möglich zu halten wird nur ein Verlauf von 10 Nachrichten pro Gespräch gespeichert, um im Falle einer Infizierung nicht unnötig Informationen preiszugeben. Der nächste Schritt ist die Absicherung von Einfallstoren. So wäre es fatal, wenn empfangenes HTML

nicht dargestellt sondern eingebunden würde. Weitere typische Fehlerquellen sind alle Formulare in die ein Benutzer etwas eingeben kann, wie eine Suchmaske oder ein Gästebuch[5]. Diaspora hat davon natürlich eine Fülle und alle zu kontrollieren ist arbeits- und zeitintensiv. Daher ist ein sauberes Grundsystem die Grundlage für die Sicherheit von JSXC.

Fremdcode kann leider nicht nur durch Formulare eingeschleust werden, sondern auch direkt über den Server. Damit ist klar, dass ein geschützter Server eine weitere Sicherheitssäule darstellt.

Genau wie ein geschützter Übertragungsweg für die Auslieferung des JS-Codes. Daher ist die Verwendung von TLS Pflicht um zum Beispiel Man-in-the-middle Attacken auszuschließen.

Welche Möglichkeiten gibt es noch um „code injection“ zu verhindern? Moderne Browser unterstützen die Content-Security-Policy (CSP), welche die Regulierung von JavaScript-Code erlaubt. Die Regeln werden über den Header mitgeliefert und erlauben die Sperrung von Skripten innerhalb des HTML-Codes und von Eval-Funktionen[6]. Damit werden XSS Attacken wirksam gebremst bzw. verhindert.

Eine weitere Idee wäre es allen JavaScript Quelltext in einer Datei zu komprimieren[7] um diese anschließend zu signieren. Diese Technik ist momentan noch nicht praktikabel da kein Browser[8] ein derartiges Verfahren implementiert hat. Aus mehreren Gründen ist dies auch schwer umzusetzen. Zum Einen werden in neuen Webseiten viele verschiedene Skripte dynamisch geladen und somit müsste für jedes Skript eine eigene Signatur erzeugt werden, was aber nicht auf dem Server geschehen dürfte, da sonst ein Angreifer sein Skript selbst signieren könnte. Zum Anderen muss der Browser auch wissen, dass die Skripte signiert sind. Dies kann aber wiederum der Server nicht leisten, da der Angreifer diesen Mechanismus auch abschalten könnte. Somit müsste jede Seite signierte Skripte verwenden, was nicht umsetzbar ist, oder es müsste einen Dritten geben der diese Information bereitstellt. Denkbar wäre unter Umständen das die DNS-Server diese Aufgabe übernehmen würden, da sie vom Browser ohnehin kontaktiert werden müssen. Ob so eine Technik je einsetzbar ist wird jedoch die Zukunft zeigen.

5.3.3 Denial-of-Service (DOS)

DOS Angriffe haben zum Ziel eine bestimmte Leistung zu blockieren, dazu werden häufig so viele Anfragen gestellt, bis der Service-Rechner nicht mehr zu erreichen ist.

Um JSXC zu testen wurden 200 Zeichen mehrere hundert Mal an die Anwendung gesendet und das Einzige was passiert ist, ist dass der Angreifer vom XMPP-Server blockiert wurde. 200 Nachrichten auf einmal konnte der Chat problemlos bewältigen, womit sie, bei einem korrekt konfigurierten XMPP-Server, DOS-fest ist.

5.3.4 Web-Storage

Wie in den vorangegangenen Kapiteln erläutert, werden zahlreiche Informationen im Speicher des Browsers hinterlegt. Dazu zählen unter anderem der DSA Schlüssel und Parameter zur XMPP Verbindungswiederherstellung (siehe 3.2.4).

Man unterscheidet dabei zwischen passiven und aktiven Angreifern. Ein aktiver hat Zugriff auf den Speicher zur Laufzeit des Chats und führt damit ein Schadskript auf der Seite aus[9]. Andere Möglichkeiten bestehen nicht, da die Spezifikation Zugriff außerhalb der Domain verbietet: „User agents must have a set of local storage areas, one for each origin.“ [8]. Eine Person hingegen, die den gleichen Browser nutzt wie das Opfer, zum Beispiel in einem Internetcafé, wird als passiver Angreifer bezeichnet.

[5] Diese spezielle Form von „code injection“ ist unter dem Begriff Cross-Site-Scripting (XSS) bekannt.
[6] Eval-Funktionen führen Text als Code aus und sind somit potentiell gefährlich.
[7] Es ist nicht das Komprimieren in typische Archivformate gemeint, sondern das Verkleinern des Quelltextes durch zum Beispiel die Ersetzung von langen Variablennamen durch kurze.
[8] Firefox unterstützt die Signierung von Code, aber nur für die Erweiterung der Rechte eines Skripts.
[9] Näheres zu dieser Angriffsmethode finden Sie in Abschnitt 5.3.2

Beide Typen können dabei in die Privatsphäre ihres Opfers eindringen, indem sie die letzten 10 Nachrichten eines jeden Kontakts oder den kompletten Roster einsehen. Darüber hinaus können sie alleine mit dem gestohlenen DSA-Schlüssel zunächst nichts anfangen. Mit einem fremden Schlüssel können zwar die eigenen Nachrichten signiert werden, dies hat aber keine Auswirkungen auf die Sicherheit, denn bereits verschlüsselte Nachrichten können nicht wiederhergestellt werden (siehe 2.3.3) und für einen Identitätsdiebstahl benötigt man die XMPP Anmeldedaten des Schlüsseleigentümers. Erst nachdem sie einen zweiten Schlüssel eines Freundes des Opfers gestohlen haben, könnten sie einen MitM-Angriff durchführen.

Über diese Optionen hinaus hat der aktive Angreifer weitere Möglichkeiten. Einerseits kann er, wie in Abschnitt 5.3.2 erläutert, Mitteilungen mitlesen und versenden. Andererseits kann er dies auch von einem fremden Rechner, denn alles was für eine Wiederherstellung notwendig ist, befindet sich im Speicher des Browsers. Dieses Angriffsszenario ist dennoch von geringer Wahrscheinlichkeit, da das Opfer nach der Übernahme vom XMPP-Server abgewiesen wird. Danach würde es natürlich versuchen sich erneut anzumelden, was zu einer Abmeldung des Angreifers führen würde. Daraus wird ersichtlich, dass das Problem des stillen Mitlesens und ggf. Mitschreibens schwerwiegender ist.

Um die Daten des Benutzers auch an öffentlichen Computern zu schützen bietet JSXC die Möglichkeit vor der Anmeldung festzulegen, dass alle Daten nach einer Sitzung gelöscht werden. Damit wird dem Chatter größtmögliche Sicherheit geboten.

Da diese Sicherheit vom Handeln des Benutzers abhängig ist, wurde auf die Speicherung von sehr sensiblen Daten, wie dem Passwort, verzichtet. Zwar wäre in den meisten Fällen eine Speicherung des Passwort-Hashes möglich (siehe 2.3.2), aber selbst dieser eröffnet die Möglichkeit sich am Server zu authentifizieren. Aus diesem Grund wurde auf die Annehmlichkeit einer Verbindungswiederherstellung bei einem Timeout verzichtet.

6 Ausblick & Zusammenfassung

6.1 Ausblick

Inzwischen wurde durch seine unabhängige Funktionsweise der Diaspora Client in einer Owncloud[1] App benutzt, welche mit einigen nützlichen Funktionen aufwarten kann. So wurde durch die neue „Page Visibility“- und „Notification“-API in HTML5 eine Benachrichtigung möglich, auch wenn sich die Webseite gerade nicht im Vordergrund befindet. Dazu wird am rechten unteren Bildschirmrand ein Benachrichtigungsfenster eingeblendet, wie man es unter anderem von seinem E-Mail Client kennt.

Des weiteren wurde durch WebRTC ein Video- und Audio-Chat implementiert auf Grundlage des Jingle[2] Protokolls, welches es in Zukunft ermöglichen sollte auch mit anderen Anwendungen als JSXC Gespräche zu führen.

Damit sind die Möglichkeiten noch nicht erschöpft: Ein verschlüsselter Dateiversand ist ebenso möglich wie Gruppenchats, Gruppentelefonate oder Statusmeldungen.

6.2 Zusammenfassung

Diese Arbeit hat einen Einblick in die Funktionsweise eines XMPP-Clients gewährt. Dabei wurde auf die verschiedenen Authentifizierungsmöglichkeiten am Server mit Hilfe von SASL, XMPP Stanzas und die Funktionsweise von OTR und SMP eingegangen. Anschließend wurde die Umsetzung in die Client-seitige Skriptsprache JavaScript erläutert unter zu Hilfenahme der neuen HTML5 Technik Web-Storage. Um verschiedene Browser miteinander zu vergleichen wurden einige Tests durchgeführt, wie die Erstellung eines DSA-Schlüssels, mit dem Ergebnis, dass im Gegensatz zu mobilen Exemplaren alle Desktop-Varianten für JSXC geeignet sind. Auch im direkten Vergleich mit Pidgin als Desktop-Derivat zeigten sich keine Nachteile für die Anwendung. Bei der Überprüfung der Programmsicherheit stellte sich heraus, dass Code injection die größte Gefahr für die Sicherheit darstellt, die aber durch TLS und ein sicheres Grundsystem minimiert werden kann.

Im Hinblick zur Zielsetzung muss man sagen, dass JSXC alle Anforderungen erfüllt hat. Es ist ein flexibler XMPP-Chat mit OTR Unterstützung, der sich nahtlos in Diaspora integriert und trotzdem leicht portierbar geblieben ist.

[1] Owncloud ist eine quelloffene Cloud-Anwendung, welche das zentrale Speichern von Daten, Terminen und Kontakten ermöglicht. `http://owncloud.org/`

[2] Jingle ist eine XMPP-Erweiterung welche die Aushandlungen von Peer-to-Peer Media-Verbindungen standardisiert. Spezifikation und Definition unter `http://xmpp.org/extensions/xep-0166.html`

Literatur

[1] K. Zeilenga A. Melnikov. *Simple Authentication and Security Layer (RFC4422)*. Network Working Group. Juni 2006. URL: http://tools.ietf.org/html/rfc4422.

[2] Chris Alexander und Ian Goldberg. »Improved user authentication in off-the-record messaging«. In: *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. WPES '07. Alexandria, Virginia, USA: ACM, 2007, S. 41–47. ISBN: 978-1-59593-883-1. DOI: 10.1145/1314333.1314340. URL: http://doi.acm.org/10.1145/1314333.1314340.

[3] Mihir Bellare, Ran Canetti und Hugo Krawczyk. »Keying Hash Functions for Message Authentication«. In: *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '96. London, UK, UK: Springer-Verlag, 1996, S. 1–15. ISBN: 3-540-61512-1. URL: http://dl.acm.org/citation.cfm?id=646761.706031.

[4] Nikita Borisov, Ian Goldberg und Eric Brewer. »Off-the-record communication, or, why not to use PGP«. In: *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. WPES '04. Washington DC, USA: ACM, 2004, S. 77–84. ISBN: 1-58113-968-3. DOI: 10.1145/1029179.1029200. URL: http://doi.acm.org/10.1145/1029179.1029200.

[5] Fabrice Boudot, Berry Schoenmakers und Jacques Traoré. »A Fair and Efficient Solution to the Socialist Millionaires' Problem«. In: *Discrete Applied Mathematics* 111 (2001), S. 2001.

[6] *Candy Chat*. URL: http://candy-chat.github.io/candy/ (besucht am 19. Aug. 2013).

[7] *Crypto Cat*. URL: https://crypto.cat/ (besucht am 19. Aug. 2013).

[8] Ian Hickson, Hrsg. *Web Storage*. Recommendation. W3C. Juli 2013. URL: http://www.w3.org/TR/webstorage/.

[9] *Jappix Mini*. URL: https://mini.jappix.com/ (besucht am 19. Aug. 2013).

[10] H. Krawczyk, M. Bellare und R. Canetti. *HMAC: Keyed-Hashing for Message Authentication (RFC2104)*. Network Working Group. Feb. 1997. URL: http://tools.ietf.org/html/rfc2104.

[11] P. Leach und C. Newman. *Using Digest Authentication as a SASL Mechanism (RFC2831)*. HISTORIC. Network Working Group. Mai 2000. URL: http://tools.ietf.org/html/rfc2831.

[12] A. Melnikov. *Moving DIGEST-MD5 to Historic (RFC6331)*. Internet Engineering Task Force (IETF). Juli 2011. URL: http://tools.ietf.org/html/rfc6331.

[13] C. Newman u. a. *Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms (RFC5802)*. Internet Engineering Task Force (IETF). Juli 2010. URL: http://tools.ietf.org/html/rfc5802.

[14] *Off-the-Record Messaging Protocol version 3*. cypherpunks. URL: http://www.cypherpunks.ca/otr/Protocol-v3-4.0.0.html.

[15] Ian Paterson u. a. *Bidirectional-streams Over Synchronous HTTP (BOSH) (XEP-0124)*. XMPP Standards Foundation. Juli 2010. URL: http://xmpp.org/extensions/xep-0124.html.

[16] *Prodromus*. URL: http://raphael.kallensee.name/journal/prodromus-ein-minimaler-javascript-xmpp-client/ (besucht am 19. Aug. 2013).

[17] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core (RFC6120)*. Internet Engineering Task Force (IETF). März 2011. URL: http://xmpp.org/rfcs/rfc6120.html.

[18] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence (RFC6121)*. Internet Engineering Task Force (IETF). März 2011. URL: http://xmpp.org/rfcs/rfc6121.html.

[19] R. Shirey. *Internet Security Glossary, Version 2 (RFC4949)*. Network Working Group. Aug. 2007. URL: http://tools.ietf.org/html/rfc4949.

[20] Andrew C. Yao. »Protocols for secure computations«. In: *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*. SFCS '82. Washington, DC, USA: IEEE Computer Society, 1982, S. 160–164. DOI: 10.1109/SFCS.1982.88. URL: http://dx.doi.org/10.1109/SFCS.1982.88.

[21] K. Zeilenga. *The PLAIN Simple Authentication and Security Layer Mechanism (RFC4616)*. Network Working Group. Aug. 2006. URL: http://tools.ietf.org/html/rfc4616.

[22] *ZXMPP*. URL: https://bitbucket.org/ivucica/zxmpp/ (besucht am 19. Aug. 2013).

Begriffserklärungen

Adapter siehe *Wrapper*

Add-On Add-On's sind Erweiterungen für bestehende Software. Damit können unter anderem Browser in ihrer Funktionalität erweitert werden.

Cookie Datei welche im Browser gespeichert werden kann.

Drop-Down Menü Ein Menü welches nach unten hin aufklappt.

Fingerprint Prüfsumme eines Schlüssels.

Kontrollzeichen Kontrollzeichen sind nicht-druckbare Schriftzeichen, wie Tabulator, Zeilenvorschub oder Umbruch.

Roster Bei XMPP spricht man bei der Freundschaftsliste von Roster.

Stanza Bruchstücke der XMPP Kommunikation werden Stanza genannt.

Tab Ein Reiter im Browser, welcher dazu dient, mehrere Seiten gleichzeitig offen zu halten.

Template Ein Template ist eine meist seitenübergreifende konsistente Design Vorlage.

TLS TLS ist der Nachfolger von SSL und dient der verschlüsselten Übertragung von Webseiten.

Tooltip Stellt Extra-Informationen beim Überfahren eines Elementes bereit.

Wrapper Wrapper sind eine Zwischenschicht beim Programmieren, welche den bestehenden Funktionsumfang umhüllt und gegebenenfalls erweitert. Dies ist manchmal nötig um das spätere Austauschen von Komponenten zu erleichtern.

Abkürzungsverzeichnis

A/V Audio/Video

AES Advanced Encryption Standard

AJAX Asynchronous JavaScript and XML

AKE Authenticated Key Exchange

API Application programming interface

BOSH Bidirectional-streams Over Synchronous HTTP

CPA chosen-plaintext attack

CSP Content-Security-Policy

CSPRNG cryptographically secure pseudo-random number generator

CSS Cascading Stylesheets

DOS Denial-of-Service

DSA Digital Signature Algorithm

FF Mozilla Firefox

HMAC Hash-Message Authentication Code

HTML Hypertext-Markup-Language

IE Internet Explorer

Jid Jabber-Id

JS Javascript

JSXC JavaScript Xmpp Chat

LAMP Linux-Apache-MySQL-PHP

MitM Man-in-the-middle

MUC Multi-User-Chat

NS Namespace

OTR Off-the-Record Messaging

Rid Request-Id

RoR Ruby-on-Rails

SASL Simple Authentication and Security Layer

SCRAM Salted Challenge Response Authentication Mechanism

Sid Session-Id

SMP Socialist millionaire protocol

TLS Transport Layer Security

XEP XMPP Extension Protocols

XML Extensible Markup Language

XMPP Extensible Messaging and Presence Protocol

XSS Cross-Site-Scripting