inchat: Ein Nachrichten-Netzwerk für unabhängige, private und schwer verfolgbare Kommunikation, entwickelt für Desktop- und Mobilgeräte

René Bernhardsgrütter
rene.bernhardsgruetter@inchat.org
www.inchat.org

10. September 2013

Abstract

Auf Desktop- und Mobilgeräten sollte unkompliziert privat kommuniziert werden können, sodass Dritte weder Inhalte mitlesen noch relevante Metadaten erfassen können. Spionage-Programme wie PRISM speichern sehr viele Metadaten und Kommunikationsinhalte. Dies verletzt die Privatsphäre vieler Menschen. Mit inchat soll ein Chat-Protokoll geschaffen werden, das Nachrichten stark verschlüsselt und durch ein Netzwerk transportiert, um Absender und Ziel für Dritte zu verschleiern. Es wird ein Konzept für ein Nachrichtennetzwerk präsentiert, das End-zu-End-Verschlüsselung erzwingt, für Desktop- sowie Mobile-Endgeräte ausgelegt ist und Nachrichten über mehrere Server leiten kann, sodass das Verfolgen der Kommunikation möglichst aufwändig ist. inchat ermöglicht auf diese Weise private Chat-Konversationen. Durch die Client/Server-Architektur des Netzwerkes können zudem komfortable Funktionen wie Offline-Benachrichtigung oder Account-Synchronisation geboten werden.

1 Einleitung

Freie Menschen sollten in der Lage sein, problemlos privat zu kommunizieren. Grossangelegte Spionage-Programme wie PRISM der NSA[1] oder TEMPORA des GCHQ[2] erlauben, Metadaten zu erfassen [1, 2] und Kommunikation in Echtzeit auszuwerten [3]. Dadurch werden Menschen, welche die eigene Privatsphäre und diejenige ihrer Kommunikationspartner schützen wollen, dazu gedrängt, sich gegen solche Angriffe zu wehren. Bisher sind die meisten frei verfügbaren elektronischen Kommunikationswerkzeuge nicht End-zu-End-verschlüsselt, es ist einfach, Metadaten abzufangen oder beides trifft zu. Lösungen wie PGP[3] und Tor[4] erlauben zwar verschlüsselte beziehungsweise anonyme Kommunikation, sind aber oftmals umständlich zu bedienen oder für die Mobilnutzung ungeeignet.

Um sichere Kommunikation auch in Alltagssituation breit verfügbar zu machen, muss ein gemeinsam genutztes, sicheres Protokoll weit verbreitet sein. Dies soll mit *inchat* erreicht werden, indem ein einfach zu verwendendes Protokoll und benutzerfreundliche Client-Anwendungen[5] mit angenehmer Optik und Haptik entwickelt werden.

inchat fokussiert auf folgendes:

- Unabhängige Infrastruktur.
- Private Kommunikation durch End-zu-End-Verschlüsselung.
- Schwer oder nicht verfolgbare Nachrichtentransferwege[6].
- Einfache Installation, Konfiguration und Verwendung der Client-Software.

[1] http://www.nsa.gov/
[2] http://www.gchq.gov.uk/
[3] http://www.openpgp.org/
[4] https://www.torproject.org/
[5] Ein Client bezeichnet eine Software-Anwendung, mit welcher der Endbenutzer interagiert.
[6] Im Sinne von verstecken der Metadaten.

- Unterstützung mobiler Geräte, welche nur begrenzt über Bandbreite, Rechenleistung und Energie verfügen.
- Unterstützung moderner Messenger-Funktionalitäten wie Gruppenkonversationen mit mehreren Teilnehmern, Nachrichten-Caches, die Nachrichten zwischenspeichern für den Fall, wenn ein Client offline ist, Synchronisation mehrerer Clients, etc..

Dieses Projekt hat zum Ziel, eine Infrastruktur mit Referenzimplementation für unabhängige, private und schwer verfolgbare Kommunikation zu schaffen. inchat soll von möglichst vielen Menschen genutzt werden und langfristig (mobile) Instant Messengers und E-Mail ersetzen.

2 Netzwerkarchitektur

Um eine unabhängige Netzwerkinfrastruktur zu ermöglichen, die gegen politische und technische Angriffe genügend resistent ist, dürfen keine zentralen Stellen existieren, die das Netzwerk kontrollieren könnten, da diese Angriffe aller Art leicht zugänglich wären und grossen Schaden anrichten könnten. Zudem soll jedermann einen eigenen Server für das Netzwerk betreiben dürfen. Dies muss auf technischer Basis vollständig unabhängig möglich sein.

Sobald genug Menschen und Firmen von inchat profitieren, wird es auf politischer Basis nicht möglich sein, das Netzwerk weltweit abzuschalten. Zudem wird die Netzinfrastruktur mit jedem neuen inchat-Server gegen Attacken aus dem Internet weniger anfällig, denn je mehr Server das Netzwerk tragen, desto aufwändiger wird es, dieses zu überlasten.

Das Netzwerk ist Client/Server-orientiert. Die Server übermitteln Nachrichten untereinander und stellen diese, wenn nötig, den Clients zu. Im Detail erfüllen die Server folgende Anforderungen:

- Jeder Server des Netzwerkes muss alle eingehenden Nachrichten verarbeiten.
- Jeder Server des Netzwerkes muss einen Teil der *Distributed Hash Table* (DHT) mittragen. In der DHT werden Adress-, Schlüssel- und Hostinginformationen gespeichert.
- Ein Server des Netzwerkes kann *Accounts* hosten.

Accounts sind im Prinzip Postfächer, vergleichbar mit denjenigen bei E-Mail-Servern. Es können also Nachrichten einem Account zugestellt werden. Zusätzlich werden aber auch noch öffentlich verfügbare Informationen des Benutzers abgelegt, wie beispielsweise Profilbild, Benutzername und der Public-Key. Alle nichtöffentlichen Daten (Nachrichten, Kontaktlisten, etc.) sind auf dem Server jederzeit verschlüsselt.

Jeder Benutzer benötigt einen eigenen Account um Nachrichten empfangen zu können. Jeder Account muss auf einem Server gehostet sein. Wenn ein Benutzer eine Nachricht an einen anderen Benutzer verschickt, wird die Nachricht durch das Netzwerk geroutet[7] und schliesslich zu demjenigen Server übertragen, der den Account des Empfängers hostet. Dieser Server stellt die Nachricht dem Account zu und informiert alle Clients, die mit diesem Account verbunden sind. Benutzer sind mit ihren Accounts nicht fix an einen Server oder Hostname gebunden, denn jeder Account kann auf einen anderen Server übertragen werden. Dadurch soll einerseits ein kapitalistisch orientierter Wettbewerb unter sich konkurrierenden Serverbetreibern, welche kommerzielle Interessen verfolgen, etabliert werden. Andererseits soll aber auch auf freier Basis Account-Hosting angeboten werden können. Beispielsweise könnten Sponsoren Server mit Account-Plätzen gratis zur Verfügung stellen.

In der DHT wird eingetragen, welcher Server auf welcher IP-Adresse beziehungsweise welchen Hostname mit zugehörigem Port erreichbar ist und welcher Client auf welchem Server gehostet ist. Wenn ein Server eine Nachricht weiterleiten muss, dieser aber die Adressdaten des Zielservers nicht kennt, kann die DHT danach abgefragt werden. Alle Informationen in der DHT werden redundant gespeichert, denn die Server werden als nicht zuverlässig angesehen.

3 Nachrichtentransfer

Der Nachrichtentransfer wird mithilfe eines Beispiels geschildert.

Abbildung 1 stellt einen Netzwerkausschnitt mit *Alice* und *Bob* dar. Alice' Account-Server ist *B*, der von Bob ist *H*. Alice möchte nun Bob eine Nachricht schi-

[7] Siehe Kapitel *Nachrichtentransfer*.

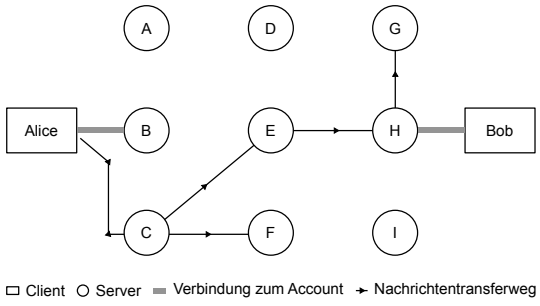

Abbildung 1: Schematische Darstellung eines Netzwerkausschnittes. *Alice* und *Bob* sind zu ihren jeweiligen Accounts auf Server *B* (Alice) beziehungsweise *H* (Bob) verbunden. Alice sendet Bob eine Nachricht via Netzwerk.

cken. Ihr Client kennt alle oder zumindest viele Server[8] [9]. Alice kennt auch Bob und den öffentlichen Schlüssel von dessen Schlüsselpaar. Möglicherweise weiss sie auch, auf welchem Server der Account von Bob gehostet ist, dies ist allerdings nicht zwingend notwendig[10].

Nun erstellt Alice einen *Transfer Graph*. Dieser Graph enthält zufällig ausgewählte Server aus dem Netzwerk, über welche die Nachricht geschickt werden soll. Es ist zu beachten, dass der erste Server, bei Abbildung 1 ist dies *C*, nicht derjenige Server sein sollte, der den eigenen Account hostet, denn sonst könnte es vorkommen, dass der Server nachvollziehen könnte, an wen die Nachricht geschickt werden soll[11]. Jeder Server leitet jede empfangene Nachricht an alle festgelegten Empfänger weiter. Wenn ein Server eine Nachricht erhält und als nächster Empfänger ein Account eingetragen ist, den der Server selbst hostet, so stellt er die Nachricht diesem Account zu. Eine Nachricht kann auch an mehrere weitere Empfänger weitergeleitet werden, wenn dies so vom ursprünglichen Absender bestimmt wurde. Es muss aber beachtet werden, dass pro *n* Verzweigungen bei einem Server, diesem *n* unterschiedliche Nachrichten zugestellt werden müssen, denn der Server muss jedem Zweig die passende Nachricht weiterleiten[12]. Diese separaten Nachrichten können alle in einer grösseren Nachricht zusammengefasst werden, es muss also nur eine Verbindung hergestellt werden, die zu übertragende Datenmenge kann sich aber pro Verzweigung verdoppeln.

Nachdem Alice den Transfer Graph erstellt hat, komprimiert, signiert und verschlüsselt sie die Nachricht rekursiv. Die Nachrichten werden verschachtelt, sodass eine Nachricht mehrere andere Nachrichten enthalten kann[13]. Dies ist bei jeder Verzweigung des Transfer Graphs erforderlich. Sollte der Transfer Graph geschlossen sein, muss dieser zuerst aufgebrochen werden[14]. Schlussendlich wird eine einzelne Nachricht generiert, die dem ersten Server *C* zugestellt wird. Dieser entschlüsselt und prüft den Inhalt. Der Nachrichteninhalt besteht aus zwei weiteren Nachrichten, adressiert an den Server *E* beziehungsweise *F*. *C* schickt also *E* die eine, *F* die andere Nachricht. Nachdem *F* die Nachricht von *C* erhalten hat, stellt er fest, dass es lediglich eine *Trash*-Nachricht ist und verwirft diese. *E* hingegen erhält eine neue Nachricht, die an *H* weitergeleitet werden muss, daher veranlasst *E* dies. Bei *H* werden zwei Nachrichten empfangen, die eine wird dem Account von Bob zugestellt, die andere muss zu dem Server *G* übertragen werden.

Nachdem Bob die Nachricht empfangen hat, kann automatisch eine Bestätigungsnachricht an Alice geschickt werden[15]. Bobs Account würde dafür den soeben beschriebenen Algorithmus verwenden.

3.1 Schwer verfolgbare Kommunikation

Die Verschleierung der Metadaten, sodass für Überwacher nicht sofort klar wird, wer mit wem wann kommuniziert, soll erreicht werden, indem Nachrichten stets über mindestens zwei Server transportiert werden. Der erste Server leitet die Nachricht an einen weiteren Server weiter, dadurch wird die IP-Adresse des Absenders entfernt. Auf dem zweiten Server könnte sich nun bereits der Ziel-Account befinden. Der hosten-

[8] *Server kennen* bedeutet, dass Empfänger-ID und der benötigte Public-Key lokal vorhanden sind, dass der Client ohne Netzwerkverbindung Nachrichten für einen Empfänger verschlüsseln kann.

[9] In den Anfangszeiten von inchat werden die Clients noch alle Server kennen, später, wenn es sehr viele Server gibt, kennen die Clients lediglich zwischen einigen hundert und wenigen tausend Servern.

[10] Wenn der Server mit dem Account nicht bekannt ist, kann auf einem Server einfach an die Account-ID weitergeleitet werden. Der Server, der dies behandelt, sucht dann selbstständig den Zielserver, welcher den Account hostet und leitet die Nachricht an diesen weiter.

[11] Dies wäre möglich, wenn Alice und Bob auf dem gleichen Server ihre Accounts hosten, Alice in dem Transfer Graphen die Nachricht gleich an erster Stelle an den Account von Bob schicken möchte und dies als erstes zum hostenden Server schickt. Es wäre vom Serverbetreiber nachvollziehbar, dass von Alice eine Nachricht an Bob geschickt wurde.

[12] Damit der nächste Empfänger die Nachricht entschlüsseln kann und auch die sonstigen Daten wie Adresse, etc., korrekt sind.

[13] Alice verschlüsselt dabei jede Nachricht so, dass jeder Server eine Schicht entschlüsseln kann. Server *C* kann den Inhalt der Nachricht an Server *E* also nicht entschlüsseln.

[14] Da dies vollständig clientseitig passiert, ist es prinzipiell irrelevant für das Protokoll. Standardmässig sollten wegen der Sicherheit auch geschlossene Graphen verwendet werden.

[15] Ein solches Verhalten muss vom Client umgesetzt werden, denn der hostende Server kennt den ursprünglichen Absender nicht.

de Server könnte die IP-Adresse nicht zurückverfolgen. Dennoch könnte bei geringem Kommunikationsaufkommen und Überwachung aller Internetanschlüsse leicht analysiert werden, welche Teilnehmer miteinander kommunizieren, indem man verfolgt, von welchem Server zu welchem Zeitpunkt Datenströme übertragen werden und wie lange diese sind. Die möglichen Kommunikationspartner beschränkten sich nur noch auf alle Benutzer, die mit den beiden Servern verbunden sind.

Um dies zu erschweren, durchläuft eine einzelne Nachricht von Alice zu Bob jeweils zufällig zwischen zwei und zehn Hops[16]. Je mehr Hops eingesetzt werden, desto schwerer verfolgbar wird eine Konversation, jedoch dauert die Übermittlung im Mittel länger und es muss mehr Rechenleistung aufgewendet werden, was auf Mobilgeräten problematisch sein könnte.

Damit Überwacher nicht sofort erkennen können, dass eine bestimmte Nachricht von Server *X* via Server *Y* zu Server *Z* transportiert wird, wird bei jeder Nachricht mit einer Wahrscheinlichkeit von 0.5 ein Verzögerung vor dem Versenden zwischen 0 und 50 ms[17] eingefügt. Zudem kann von jedem Server zufällig Trash-Content zu jeder Nachricht hinzugefügt werden. Eine Nachricht kann also in der Grösse in alle Richtungen variieren. Dies soll erschweren bis verhindern, dass der Nachrichtenverlauf durch Traffic-Analysis nachvollzogen werden kann. Je mehr Nachrichten über einen Server geroutet werden, desto geringer wird die Wahrscheinlichkeit, dass zuverlässig festgestellt werden kann, welche Nachricht wohin geleitet wurde. Jeder Server verschickt bei wenig Routing-Traffic zudem auch selbst Trash-Nachrichten an andere Server oder Clients. Die jeweiligen Ziele verwerfen diese automatisch.

Die grösstmögliche Sicherheit erreichte man allerdings, indem alle Nachrichten an alle Benutzer gesendet würden. Das hätte aber ein immenses Datenaufkommen zur Folge, zu viel für Mobilgeräte. Wenige Daten müssten transportiert werden, wenn die Nachrichten direkt zwischen den Benutzern ausgetauscht würden, dann könnte aber direkt verfolgt werden, wer wann mit wem kommuniziert. Dieses Konzept stellt folglich bezüglich der Verfolgbarkeit einen Kompromiss dar, der zu mässig viel Datenaufkommen führt.

Differenziert muss allerdings die Anbindung der Clients an deren Accounts auf Servern betrachtet werden. Wenn ein Überwacher alle Clients überwacht, kann verhältnismässig einfach analysiert werden, wer mit wem kommuniziert, denn es muss lediglich aufgezeichnet werden, wann Client *X* eine gewisse Datenmenge sendet und Client *Y* diese oder eine ähnlich grosse Datenmenge empfängt. Je länger dies bei allen Teilnehmern protokolliert wird, desto genauer kann bestimmt werden, wer mit wem kommuniziert. Um einen solchen Angriff zu verhindern, müssten auch von und zu den Clients viele Daten übertragen werden, was bei Mobilgeräten standardmässig unzumutbar ist. Sollte dies dennoch gewünscht werden, könnte man zwischen Client und Account-Server Daten mit einer konstanten Bandbreite austauschen. Ein Überwacher könnte nun nicht mehr unterscheiden, ob eine Nachricht oder Datenmüll zwischen Server und Client übertragen wird [4]. Es wäre auch nicht mehr feststellbar, ob ein Client mit anderen Kommunikationspartnern kommuniziert oder nicht.

Auf diese Funktionalität wird anfangs verzichtet, da primär innerhalb des Netzwerkes Metadaten verschleiert werden sollen.

4 Nachrichtenformat

In diesem Kapitel wird beschrieben, wie das Nachrichtenformat aufgebaut ist.

Listing 1: inchat-Nachricht in Textdarstellung.

```
version       = "0.1"
recipient     = 0x... // bytes
cipherSuite   = "RSA2048_AES256_GZIP"
keyParameters = 0x... // bytes
integrity     = 0x... // bytes
authorization = 0x... // bytes
content       = 0x... // bytes
delay         = 50 // in Millisek.
trash         = 86
```

Das inchat-Protokoll ist binär. Es basiert auf MessagePack[18]. Jede Nachricht besteht aus mehreren *Feldern*, die jeweils einen Bezeichner und den zugehörigen Wert enthalten.

[16] Als Hop zählt jede Verarbeitung einer Nachricht durch einen Server. Im Beispiel von Alice und Bob wurden insgesamt fünf Hops durchlaufen.

[17] Die Länge der Verzögerung wird innerhalb des angegebenen Rahmens zufällig bestimmt.

[18] http://msgpack.org

version Enthält die Protokoll-Version.

recipient Enthält die ID des nächsten Empfängers. Es muss sich dabei um eine Server- oder Account-ID handeln. Die ID kann in der DHT nachgeschlagen werden. Der recipient-Wert ist ein SHA-1-Hash des Public-Keys des Accounts beziehungsweise des Servers[19].

cipherSuite Beschreibt die verwendeten Cipher. Standardmässig wird RSA3072_AES256_GZIP unterstützt. Die verschiedenen Werte, getrennt durch die Unterstriche, beschreiben von links nach rechts: Verschlüsselungsverfahren für das keyParameters-Feld, Verschlüsselungsverfahren für das integrity-, das authorzation- und das content-Feld, Kompressionsverfahren, das auf das content-Feld vor der Verschlüsselung angewendet wird. Letzteres kann auch *NULL* lauten, dann wird der Inhalt des content-Feldes nicht komprimiert.

keyParameters Enthält das verwendete Schlüsselmaterial für die verwendete Cipher der Felder integrity, authorzation, content und delay.

integrity Dieses Feld dient der Integritätsprüfung. Es enthält einen SHA-256-Hash von den Feldern version, recipient, cipherSuite, keyParameters, authorization (wenn vorhanden), content, delay und trash. Es wird alles zusammen zu einem Hash-Wert verrechnet. Die Felder keyParameters, authorization, content, delay und trash sind beim Zeitpunkt der Hash-Berechnung unverschlüsselt. Das integrity-Feld wird verschlüsselt übertragen. Dadurch kann sichergestellt werden, dass kein Feld manipuliert wurde. Sobald etwas verändert wird, ergibt sich ein anderer Hash-Wert des integrity-Feldes. Da dieser selbst nur verschlüsselt übertragen wird, kann der Hash-Wert nicht unbemerkt verändert werden.

authorization Dieses Feld ist nur bei denjenigen Nachrichten vorhanden, die beim Account-Server des Zieles dem Benutzer zugestellt werden. Im authorization-Feld wird der signierte Hash-Wert aus dem integrity-Feld übertragen. Auf diesem Weg wird sichergestellt, dass der Kommunikationspartner eines Benutzers bekannt ist.

content Enthält den eigentlichen Nutzinhalt. Der Empfänger muss den Inhalt entschlüsseln, dekomprimieren (abhängig von den Angaben in dem Feld cipherSuite) und den Inhalt interpretieren. Server erwarten in jedem Fall eine oder mehrere neue inchat-Nachrichten, also wieder in dem hier beschriebenen Nachrichtenformat, oder die Anweisung, die Nachricht zu verwerfen. Clients erwarten die eigentliche Nachricht zusammen mit einigen weiteren Daten (Absendezeitpunkt, etc.).

delay Dieses Feld enthält eine Anzahl Millisekunden, die der empfangende Server vor dem Weiterleiten der Nachricht warten muss. Dieser Wert wird jeweils vom Client bestimmt, damit möglichst wenig Entscheidungsfreiheit an die involvierten Server abgegeben werden muss. Es kann mit statistischen Methoden überprüft werden, ob ein Server die vom Client vorgegebene Verzögerungszeit einhält.

trash Das trash-Feld enthält unterschiedliche Informationen. Bei Nachrichten, die ein Server S_1 weiterleiten muss, enthält das Feld zuerst eine Zahl N. Der S_1 interpretiert diese und ersetzt sie mit einem String von N aneinandergereihten T-Buchstaben. Damit werden Trash-Bytes erzeugt um Traffic-Analysis zu erschweren. Wenn der S_1 die vom Client bestimme Anzahl von N Ts nicht erzeugt, stimmt der Hash-Wert der gesendeten Nachricht nicht und sie wird vom nächsten Server S_2 verworfen.

5 Verschlüsselung und Authentifizierung

Es wurde bereits einiges zu den Verschlüsselungs- und Authentifizierungsverfahren im Kapitel *Nachrichtenformat* erwähnt, hier soll nun ergänzend darauf eingegangen werden. Es wird wieder anhand des Nachrichtenformats beschrieben.

Standardmässig kommen folgende Verschlüsselungsverfahren zum Einsatz:

- RSA-3072 mit OAEP-Padding für asymmetrische Verschlüsselung.

[19] Wenn sich das Schlüsselpaar eines Benutzers ändert, kann die ID im Netzwerk aktualisiert werden.

- RSA-3072 mit PSS-Padding für Signierung.
- AES-256 im CBC-Modus, zusammen mit PKCS#7-Padding für symmetrische Verschlüsselung.

Das Feld keyParameters wird asymmetrisch, die Felder integrity, authorization, content und delay symmetrisch verschlüsselt.

Nachdem eine Nachricht zusammengestellt wurde, wird sie via TLS übertragen. Es wird das Diffie-Hellman Key-Exchange-Verfahren verwendet, der Zielserver authentifiziert sich jeweils mit den RSA-3072-Schlüsseln, der Datenstrom wird mit AES-256 verschlüsselt. Dies wird auf der gesamten Route durch das Netzwerk so gehandhabt, auch vom sendenden Client zum ersten Server. Durch den TLS-Tunnel ist für Evesdropper nicht mehr unterscheidbar, ob es sich bei den übertragenen Daten um inchat- oder andere TLS-verschlüsselte Kommunikation handelt. Durch die Verwendung des Diffie-Hellman-Key-Exchange-Verfahrens wird Perfect Forward Secrecy [5] zwischen den Servern hergestellt.

6 Netzwerk-Performance und -Finanzierung

Im Vergleich zu klassischen Messenger-Protokollen wie XMPP[20] oder E-Mail beziehungsweise SMTP[21] wird mit inchat sehr viel Mehraufwand erzeugt. Die durch das Weiterleiten erzeugte Netzwerklast ist einerseits gewollt, da dadurch schwerer nachvollziehbar ist, welche Nachricht welchen Weg durch das Netzwerk nimmt, andererseits ist es auch ein Kostenproblem, denn die verwendeten Server müssen finanziert werden. Aus Serverbetreiber-Sicht sind auch die Accounts ein Problem, denn diese benötigen Speicherplatz. Allerdings erfordern komfortable Funktionalitäten wie Synchronisierung, Offline-Cache und der Anspruch an die Möglichkeit der realistischen Nutzung auf Mobilgeräten dies. Um ein solches Netzwerk dennoch bezahlbar zu betreiben, soll möglichst viel Last auf verschiedene private Firmen und Staaten beziehungsweise staatliche Behörden verteilt werden.

Da komfortable Verschlüsselung, kombiniert mit dem Schutz der Metadaten private Kommunikation ermöglicht und inchat unabhängig ist, wird die Verwendung dessen für Firmen und staatliche Behörden ab einem gewissen Verbreitungsgrad des Protokolls sehr lukrativ. Es ist denkbar, dass solche Organisationen die eigenen Nachrichten aus Datenschutzbedenken möglicherweise selber hosten möchten, wie dies mit privaten E-Mail-Servern oftmals der Fall ist. Eine Begründung könnte sein, dass keinem fremden Serverbetreiber vertraut werden möchte und dass erreicht werden soll, dass die Kommunikationsaktivität einzelner Mitarbeitenden möglichst geheim gehalten werden kann. Wenn ein Account auf einem eigenen Server gehostet wird, kann die Aktivität[22] von keinem anderen Server erfasst werden. Vor allem werden Firmen und staatliche Behörden grössere Accounts anlegen und unter Umständen spezialisierte Funktionalitäten nutzen wollen, beispielsweise Gruppenkonten erstellen oder grössere Nachrichten lokal versenden können. Dies wäre in erster Linie mit einem selbst betriebenen und konfigurierbaren Server umsetzbar.

Da auch diese Server die in Kapitel *Netzwerkarchitektur* beschriebenen Anforderungen erfüllen müssen, leisten sie *Öffentlichkeitsarbeit* wenn sie Nachrichten weiterleiten. Dadurch wird die Netzwerklast verteilt.

Anhand des Beispiels von WhatsApp[23] soll berechnet werden, wie viele Server etwa notwendig sind, um ein solches Netzwerk zu betreiben. WhatsApp wird zurzeit von rund 300 Mio. monatlich aktiven Benutzern eingesetzt [6], es werden täglich etwa 30 Mrd. Nachrichten versendet [7]. Nun soll dies mit inchat bewältigt werden. Es wird von folgenden Annahmen ausgegangen:

- Ein inchat-Server verarbeitet 80 Nachrichten pro Sekunde[24].
- Eine Nachricht wird im Mittel von 4 Servern behandelt (Anzahl Hops).

$$C_{Message} = 30 * 10^9 Messages$$

[20] https://tools.ietf.org/html/rfc6120

[21] https://www.ietf.org/rfc/rfc2821.txt

[22] Gemeint im Sinne der Nutzungshäufigkeit, der Anzahl empfangene Nachrichten, etc..

[23] https://www.whatsapp.com/

[24] Eine Entschlüsselung des keyParameters-Feldes mit RSA-3072 dauert bei 2.6 GHz etwa 30 ms. Es wird angenommen, dass eine Nachricht bei im Mittel innert 50 ms verarbeitet ist. Bei vier CPU-Kernen können somit 80 Nachrichten pro Sekunde verarbeitet werden. Zudem werden verwendete TLS-Verbindungen offen gehalten, sodass nicht für jede Nachricht eine neue TLS-Verbindung erstellt werden muss.

$$S_{Sec} = 80 \frac{Messages}{Sec}$$

$$S_{Day} = S_{Sec} * 60 * 60 * 24 = 864000 \frac{Messages}{Day}$$

$$C_{Hops} = 4$$

$$C_{Server} = \left\lceil \frac{C_{Messages}}{S_{Day}} * C_{Hops} \right\rceil = 17362$$

Um die 30 Mrd. Nachrichten pro Tag zuzustellen, einschliesslich der Routing-Mehraufwände, werden etwa 17500 Server benötigt.

Diese Zahlen sind rein theoretisch und basieren zentral auf ungetesteten Annahmen, werden also bei realen Bedingungen mit grosser Wahrscheinlichkeit signifikant abweichen. Es konnte aber gezeigt werden, dass auch ein immenses Nachrichtenaufkommen, vergleichbar mit demjenigen von WhatsApp, realistisch bewältigt werden kann. Die Benutzer müssen sich zwar selbst um einen Account-Server kümmern oder selbst einen solchen betreiben, der notwendigerweise erzeugte Mehraufwand wird jedoch vom Netzwerk getragen.

7 Komfortfunktionen

In diesem Kapitel soll auf die in der *Einleitung* erwähnten modernen Messenger-Funktionalitäten eingegangen werden.

Grundsätzlich können mehrere Clients mit einem Account verbunden sein. Wenn sich etwas im Account verändert, zum Beispiel eine neue Nachricht zugestellt wird, werden alle mit dem Account verbundenen Clients darüber informiert. Dadurch ergibt sich beispielsweise die Synchronisation über mehrere Clients automatisch.

Sollte kein Client mit einem bestimmten Account verbunden sein, können Nachrichten dennoch zugestellt werden. In einem solchen Fall speichert der Server die Nachricht in dem Account und informiert den Client beim nächsten Verbinden über die neue Nachricht.

Gruppenchats werden grösstenteils clientseitig implementiert. Prinzipiell werden inchat-Nachrichten einfach an mehrere Accounts geschickt. Diese Nachrichten werden speziell gekennzeichnet, sodass die Clients aller Gruppenmitglieder die Nachricht korrekt einordnen können. Statusänderungen, wie zum Beispiel Mitgliederzuwachs, werden via *Systemnachrichten*[25] allen Gruppenmitglieder signalisiert.

8 Kompromittierende Server

Serverbetreiber könnten veranlassen, dass ihre Server Nachrichten verwerfen, die eigentlich weitergeleitet werden müssten. Damit dies möglich unattraktiv ist, können solche Server blockiert werden.

Ein Hosting-Server S_H kann seinen Accounts Serverlisten zur Verfügung stellen. Ein Server auf der Serverliste heisst S_L. Jeder Server S_L wird von S_H regelmässig kontrolliert, indem S_H Nachrichten via S_L an sich selbst sendet[26]. So kann die Zuverlässigkeit und die Latenz überprüft werden. Sollte S_L Nachrichten verwerfen, stellt dies S_H fest und teilt es seinen Clients mit, sodass diese die nicht korrekt funktionierenden Server nicht mehr verwenden. Sollte sich der Account von einem Kommunikationspartner auf einem kompromittierten Server befinden, wird dies beim Client als *Verbindungsproblem* oder ähnlich signalisiert.

9 Lizenzierung

Die Referenzimplementation, voraussichtlich in Java geschrieben[27], wird unter der *General Public License, Version 3*[28], lizenziert. Die GPL wird der LGPL vorgezogen, da Closed-Source-Software in sicherheitskritischen Bereichen prinzipiell nicht vertrauenswürdig ist, denn es kann nicht jederzeit kontrolliert werden, ob vom Hersteller Hintertüren eingebaut wurden, wie dies beispielsweise bei Microsoft passiert ist [8]. Daher soll die Referenzimplementation der Kernbibliothek *inchat-common* nur als Open-Source-Software verbreitet werden dürfen. Natürlich kann das inchat-Protokoll auch proprietär oder unter einer (anderen) Open-Source-Lizenz implementiert werden.

[25] Systemnachrichten sind inchat-Nachrichten, die von den Clients interpretiert werden. Beispiel: Ein neues Mitglied ist der Gruppe beigetreten oder der Name der Gruppe wurde geändert.

[26] Es ist zu beachten, dass S_L nicht erfahren darf, wann er getestet wird. Eine Testnachricht muss also zuerst immer über einen Drittserver und erst dann an S_L gesendet werden, selbiges von S_L zurück zu S_H.

[27] Die Referenzimplementation soll auf Android, Linux und Windows funktionieren, daher wurde Java als Sprache gewählt.

[28] https://gnu.org/licenses/gpl-3.0.txt

10 Fazit

inchat bietet primär private und schwer verfolgbare Echtzeittextkommunikation. Ausgelegt für Desktop- und Mobile-Geräte aller Kategorien kann das Protokoll in vielen Bereichen Anwendung finden. Das Netzwerk ist dezentral aufgebaut und unabhängig von politischen Interessen. Jedermann kann eigene Server betreiben und nutzen. Durch die Client/Server-Infrastruktur können komfortable Funktionen relativ einfach entweder client- oder serverseitg implementiert werden. Das inchat-Protokoll muss dafür nicht oder kaum angepasst werden.

Literatur

[1] G. Greenwald, *NSA collecting phone records of millions of Verizon customers daily*. The Guardian, 2013, http://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order.

[2] E. MacAskill, J. Borger, N. Hopkins, N. Davies, J. Ball, *GCHQ taps fibre-optic cables for secret access to world's communications*. The Guardian, 2013, http://www.theguardian.com/uk/2013/jun/21/gchq-cables-secret-world-communications-nsa.

[3] G. Greenwald, *XKeyscore: NSA tool collects 'nearly everything a user does on the internet'*. The Guardian, 2013, http://www.theguardian.com/world/2013/jul/31/nsa-top-secret-program-online-data.

[4] N. Ferguson, B. Schneier, *Practical Cryptography*. John Wiley & Sons, 2003.

[5] *Diffie–Hellman key exchange*. Wikipedia, 2013, https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange.

[6] L. Gannes, *The Quiet Mobile Giant: With 300M Active Users, WhatsApp Adds Voice Messaging*. AllThingsD, 2013, http://allthingsd.com/20130806/the-quiet-mobile-giant-with-300m-active-users-whatsapp-adds-voice/.

[7] *Twitter / WhatsApp: new daily record*. WhatsApp Inc., 2013, https://twitter.com/WhatsApp/status/344966710241161216.

[8] G. Greenwald, E. MacAskill, L. Poitras, S. Ackerman, D. Rushe, *How Microsoft handed the NSA access to encrypted messages*. The Guardian, 2013, http://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data.