HÖGSKOLAN
I GÄVLE

AKADEMIN FÖR TEKNIK OCH MILJÖ
Avdelningen för industriell utveckling, IT och samhällsbyggnad

Utveckling av ett krypterat meddelandesystem

David Pettersson

2018

Examensarbete, Grundnivå (högskoleexamen), 15 hp
Datavetenskap
Dataingenjörsprogrammet

Handledare: Anders Hermansson
Examinator: Niclas Björsell

Abstract

Today there exists many messaging applications and new ones are released almost daily. However, there are not as many messaging applications where both the client application and the server application are open source and can be self hosted. These messaging applications are good at solving the problem that many, up to several million people can communicate with each other. However, there is a smaller more focused use case with a messaging system consisting of only one message flow, where all messages are visible to all devices. This smaller use case means that less metadata is required. Uses for such a messaging system may be to receive notifications from servers or to take personal notes. The message system is synchronized between all devices. The goal of this project is to create an End-to-End encrypted messaging system that provides authentication, data integrity and confidentiality. Research has been investigated regarding weaknesses in some encryption techniques, these weaknesses have been taken into account when creating the messaging system. The purpose why this messaging system needs to be created is because an existing system that meets certain essential requirements for maintaining maximum personal integrity is missing. A prototype has been created that consists of a server application and a graphical computer application. Messages sent in the messaging system are End-to-End encrypted. Communication between clients and the server is also encrypted and authenticated.

Nyckelord: End-to-End encryption, OpenPGP, Private messaging system, Open-source

Sammanfattning

Idag finns det en uppsjö av meddelandeapplikationer och det kommer nya nästan varje dag. Dock finns det inte lika många meddelandeapplikationer där både klient-applikationen och serverapplikationen är öppen källkod och kan köras på egna privata servrar. Dessa meddelandeapplikationer är bra på att lösa problemet att många, upp till flera miljoner personer ska kunna kommunicera med varandra. Dock finns det ett mindre användarfall med ett mindre meddelandesystem som består av enbart ett meddelandeflöde, där alla meddelanden är synliga för alla enheter. Detta mindre användarfall gör att mindre metadata behövs. Användningsområden för ett sådant meddelandesystem kan vara bl.a. att få notifikationer från servrar eller att skriva anteckningar. Meddelandeflödet synkroniseras mellan alla enheter. Målet med detta arbete är att skapa ett End-to-End krypterat meddelandesystem som uppfyller autentisering, dataintegritet och sekretess. Forskning har granskats gällande svagheter i vissa krypteringstekniker, dessa svagheter har tagits i åtanke vid skapandet av meddelandesystemet. Syftet varför detta meddelandesystem behöver skapas är därför att något befintligt system som uppfyller vissa viktiga krav för att upprätthålla den personliga integriteten saknas. En prototyp har skapats som består av en serverapplikation och en grafisk datorapplikation. Meddelanden som skickas i meddelandesystemet är End-to-End krypterade. Kommunikation mellan klienter och server är även den krypterad och autentiserad.

Nyckelord: End-to-End kryptering, OpenPGP, Privat meddelandesystem, Öppen källkod

Ordlista

End-to-End kryptering

Det betyder att data krypteras och dekrypteras av klienterna själva och att eventuella servrar emellan inte har tillgång till några dekrypteringsnycklar.

Man-in-the-middle attack

Det är en typ av attack där en attackerare sätter sig emellan klienterna för att t.ex. kunna läsa, modifiera eller stoppa trafik mellan två andra parter, utan deras vetskap.

Autentisering

Säkerställer att någon är den de påstår sig vara.

Dataintegritet

Garanterar att data är oförändrad.

Personlig integritet

Rätten för en människas privatliv.

Sekretess

Data kan ej läsas av obehöriga.

Digital signatur

En signatur av ett meddelande för att uppnå bl.a. autentisering och dataintegritet.

Fingeravtryck

Fingeravtryck skapat av en kryptografisk hashfunktion. En hashfunktion är en funktion med varierande storlek av ingångsvärde men med bestämd storlek av utgångsvärdet (hashen). En kryptografisk hashfunktion är gjord så att det är mycket svårt att från ett hash finna vilket ingångsvärde som använts.

Fingeravtryck kan användas för dataintegritet.

Innehållsförteckning

1 Introduktion

Det är viktigt att kryptera meddelanden som transporteras över osäkra nätverk såsom Internet om den personliga integriteten vill bevaras. Det visade sig tydligt efter att Edward Snowden år 2013 avslöjade den massavlyssning som genomförs av USA. Många påstår att de inte har något att dölja så varför kryptera meddelanden överhuvudtaget? Edward Snowden har ett bra argument mot det. ”Att säga att personlig integritet inte spelar någon roll eftersom du inte har något att dölja är samma sak som att säga att du inte bryr dig om yttrandefrihet därför du inte har något att säga” [1].

Idag lanseras det nya meddelandeapplikationer på rullande band och trots att många av dessa meddelandeapplikationer implementerar End-to-End kryptering finns det ofta ändå vissa brister som kan försämra den personliga integriteten. En sådan brist är att metadata är tillgänglig i klartext vilket kan vara information såsom tidstämplar, hur många meddelanden som skickats och vilka som kommunicerat med vilka. En annan brist är att applikationerna inklusive serverapplikationerna inte är öppen källkod. Med stängd källkod är det svårt att verifiera vad ett program faktiskt gör samt att det inte går att modifiera källkoden.

De flesta meddelandeapplikationer är gjorda för att lösa problemet att många (miljontals) personer skall kunna kommunicera med varandra, men det finns ett mindre användarfall med ett liknande problem att lösa fast som är mer nischat. Nämligen ett meddelandesystem som är tänkt att användas av några enstaka enheter med endast ett meddelandeflöde där alla meddelanden som skickas är ämnade till alla. Användarfall för det kan vara ett meddelandeflöde där personliga anteckningar skrivs ned, där automatiska notifikationer skickas från servrar eller för att skicka länkar till sig själv. Ett annat användarfall kan vara ett privat meddelandeflöde inom familjen. Detta meddelandesystem ska vara synkroniserat mellan alla enheter vilket skulle kunna vara en individs mobiltelefon, bärbara dator eller stationära dator. Metadata som kan vara ett krav för att dagens moderna meddelandeapplikation skall fungera (t.ex. för att kunna skicka meddelanden till rätt person) behövs inte alls i detta mindre användarfallet eftersom alla meddelanden som skickas ska till alla enheter.

1.1 Bakgrund

Dagens meddelandeapplikationer använder ofta End-to-End kryptering men de kräver ändå att viss metadata finns tillgängligt i klartext så att servrarna kan skicka meddelanden till rätt enheter. Denna metadata är ett problem för den personliga integriteten eftersom den kan avslöja vilka enheter som kommunicerat med varandra vid vilken tidpunkt.

Ett annat vanligt problem är att klientapplikationerna och serverapplikationerna inte är öppen källkod, vilket kan försämra den personliga integriteten. Detta eftersom

det då inte går att veta vilken kod som faktiskt körs, hur krypteringen går till och vilken metadata som existerar och hur den hanteras.

Det finns två olika sätt kommunikation kan ske för ett meddelandesystem. Antingen att alla enheter är uppkopplade och enbart då kan skicka meddelanden till varandra, alltså en typ av chatt eller snabbmeddelanden. Det andra sättet är att alla enheter kan skicka meddelanden när som helst asynkront, även fast ingen direkt anslutning mellan avsändare och mottagare kan öppnas. Det första fallet är enklare att designa ett system för därför krypteringen blir inte lika komplicerad eftersom en handskakning kan utföras direkt för att utbyta eventuella nycklar med varandra. Dock blir det väldigt begränsat att bara kunna skicka meddelanden när alla enheter är uppkopplade samtidigt.

1.2 Syfte

Syftet med detta arbete är att skapa ett privat meddelandesystem som är skräddarsytt för att användas av ett begränsat antal enheter. Detta behöver göras därför det saknas lösningar för det idag som uppfyller vissa viktiga krav för att upprätthålla den personliga integriteten så mycket som möjligt. Dessa krav är:

- End-to-End kryptering
- Dataintegritet och autenticitet
- Minimera mängden metadata
- Öppen design som möjliggör implementationer med öppen källkod
- Autentisering av klienter och server

1.3 Frågeställningar

Arbetet svarar på följande frågor:

1. Hur kan End-to-End kryptering mellan klienter uppnås?
2. Hur kan dataintegritet och autentisering mellan klienter uppnås?
3. Hur kan sekretess, dataintegritet och autentisering uppnås mellan klient och server?

1.4 Avgränsningar

För detta meddelandesystem behövs åtminstone en klient och en server. Två klienter hade varit önskvärt, en klient för en dator och en klient för Android. Dock har en klient för en dator prioriterats och ingen klient för Android har skapats.

2 Teori

I följande avsnitt kommer olika typer av krypteringstekniker förklaras och aktuell forskning att presenteras.

2.1 Asymmetrisk och symmetrisk kryptering

Asymmetrisk kryptering är en teknik som använder sig av ett nyckelpar uppdelat i en öppen (publik) och en hemlig (privat) nyckel. Kryptering sker med den öppna nyckeln och dekryptering sker med den hemliga nyckeln. Digitala signaturer skapas med den hemliga nyckeln och kontrolleras med den öppna nyckeln. Den öppna nyckeln kan spridas öppet, det är praktiskt omöjligt att från den öppna nyckeln beräkna fram den hemliga nyckeln.

Symmetrisk kryptering använder samma nyckel för kryptering och dekryptering. Asymmetrisk kryptering löser problemet att det är svårt att överföra en hemlig nyckel och samtidigt bibehålla den hemlig, eftersom den öppna nyckeln kan spridas utan att det utgör någon säkerhetsrisk. Däremot är asymmetrisk kryptering långsam och har problem att kryptera större mängder data till skillnad mot symmetrisk kryptering som inte har dem problemen. Därför genereras en nyckel kallad sessionsnyckel som används för symmetrisk kryptering, den nyckeln krypteras med asymmetrisk kryptering med mottagarens öppna nyckel. Mottagaren kan sedan dekryptera sessionsnyckeln med sin hemliga nyckel och därefter dekryptera meddelandet med symmetrisk kryptering. För att uppnå dataintegritet och autenticitet signeras meddelandet med avsändarens hemliga nyckel innan meddelandet krypteras.

2.2 OpenPGP

PGP står för Pretty Good Privacy och är en proprietär krypteringsprogramvara skapad av Phil Zimmermann år 1991 [2]. Det fanns vissa patent och upphovsrättsproblem med PGP. Det ledde till att en öppen standard skapades år 1997 kallad OpenPGP. OpenPGP är en specifikation som kan implementeras fritt utan att betala licensavgifter. OpenPGP används för att säkert skicka meddelanden över osäkra nätverk såsom Internet. Det möjliggör End-to-End kryptering genom att kombinera asymmetrisk och symmetrisk kryptering. Tillit till någon komponent annan än sändarens och mottagarens datorer behövs inte. OpenPGP garanterar även dataintegritet vilket betyder att meddelandet är oförändrat samt autenticitet genom att använda digitala signaturer.

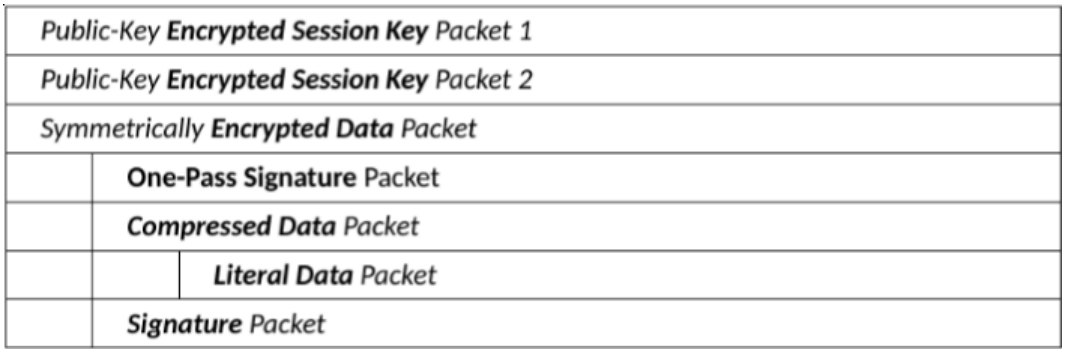
År 2017 genomfördes en genomgående granskning av OpenPGP som hittade vissa sårbarheter av OpenPGP och förbättringsförslag föreslogs [3]. Granskningen visade att två av dem populäraste OpenPGP programvarorna GnuPG och Symantec PGP inte var sårbara mot attackerna då de varit noggranna med implementationerna.

I maj 2018 publicerades en attack mot användandet av OpenPGP och S/MIME för email kallad Efail [4]. Det har blivit många skriverier kring Efail, och falsk information om att OpenPGP protokollet är trasigt har spridits. Det visade sig att OpenPGP protokollet inte är trasigt av Efail, utan det är implementationer av OpenPGP av vissa emailklienter det handlar om [5]–[7].

När ett meddelande krypteras med OpenPGP till flera personer så krypteras inte hela meddelandet flera gånger utan istället skapas ett sessionsnyckelpaket för varje mottagare som radas upp efter varandra som innehåller sessionsnyckeln som använts för att kryptera meddelandet (figur 2.1, *Public-Key Encrypted Session Key Packet*). Dessa paket krypteras var för sig med mottagarnas öppna nycklar [8]. Mottagarens nyckel-id läggs till för varje sessionsnyckelpaket vilket gör att metadata om vilka som är mottagarna av ett meddelande läcker ut. För att undvika denna metadata har vissa OpenPGP implementationer stöd för att inte avslöja nyckel-id, med GnuPG kan flaggan `--hidden-recipient` användas för detta.

OpenPGP definierar inte om data först skall signeras och sedan krypteras eller tvärtom [8]. Det finns fördelar och nackdelar med båda metoderna. En fördel med att först signera meddelandet och sedan kryptera (som i figur 2.1) är att mottagaren först måste dekryptera OpenPGP meddelandet för att se vem som skapat signaturen. Det är denna metod GnuPG använder. Om meddelandet först krypteras och sedan signeras så blir signaturen publik, och vem som är avsändare läcker då ut. En annan nackdel med att skapa signaturen efteråt är att signaturen kan plockas bort och bytas ut av t.ex. en attackerare. En fördel med att skapa signaturen efteråt är att mottagaren kan kontrollera signaturen direkt, för att på så vis kunna avbryta utifall signaturen inte är korrekt.

OpenPGP har inget inbyggt skydd mot *replay-attacker*. En *replay-attack* är en typ av attack där en attackerare avlyssnar eller stoppar trafik. Attackeraren kan då antingen kasta bort eller fördröja vissa meddelanden utan att mottagaren märker det eller återskicka samma meddelande flera gånger. Skydd mot detta behöver hanteras manuellt.



Figur 2.1: Exempel struktur OpenPGP meddelande [9]

2.3 Secure Sockets Layer (SSL)

SSL utvecklades av Netscape för att lägga till HTTPS protokollet till deras webbläsare Netscape Navigator. SSL finns i tre versioner, den första versionen publicerades aldrig därför det fanns allvarliga säkerhetsbrister i protokollet och den andra versionen som släpptes 1995 visade sig också ha säkerhetsbrister. Den tredje och sista versionen av SSL var en total omskrivning av protokollet och släpptes 1996. År 2014 visade det sig att SSL 3.0 var sårbar av en attack kallad POODLE som är en typ av man-in-the-middle attack [10]. Det är idag starkt rekommenderat att inte använda någon version av SSL överhuvudtaget. År 2011 publicerade Internet Engineering Task Force (IETF) ett dokument som förbjuder användandet av SSL 2.0 [11]. Ett liknande dokument publicerades år 2015 gällande SSL 3.0 [12].

2.4 Transport Layer Security (TLS)

TLS är en vidareutveckling av SSL 3.0, TLS kan anses vara det viktigaste krypteringsprotokollet som existerar idag eftersom det är så vidspritt. TLS är ett kryptografiskt kommunikationsprotokoll som är en öppen standard för att öppna en krypterad kanal mellan två datorer. Målet med TLS är att kunna uppnå autentisering, dataintegritet och sekretess [13]. Autentiseringen är frivillig men används oftast av servern genom användandet av asymmetrisk kryptering och digitala certifikat. Det är möjligt att använda autentisering av båda parter, vilket betyder att om certifikat används för autentisering så skickar båda parter varsitt certifikat i handskakningen. Det finns fyra versioner av TLS, den första versionen kom 1999 och den senaste versionen kallad TLS 1.3 godkändes[1] i mars 2018 av IETF [14] och förväntas nu bli den nya standarden. TLS 1.3 innebär förändringar, bl.a. så tas många osäkra krypteringsprimitiver bort. Statiska RSA och Diffie-Hellman chiffersviter tas även bort. Med statiska chiffersviter menas att samma nyckelpar återanvänds. Borttagandet av de statiska chiffersviterna gör att TLS 1.3 blir den första versionen att stödja *forward secrecy* (*forward secrecy* beskrivs i nästa avsnitt) för alla chiffersviter med asymmetriskt baserade nyckelutbyten. En annan nyhet är att det bara krävs en tur och retur (1-RTT, round-trip time) för handskakningen istället för två som det krävdes tidigare, vilket gör att fördröjningen av handskakningen halveras. Det läggs även till stöd för att skicka krypterad data redan med första handskakningspaketet (0-RTT), dock med vissa förlorade säkerhetsaspekter (såsom *forward secrecy*).

År 2017 granskades Diffie-Hellman Ephemeral (DHE) varianterna av TLS vilket visade att det underliggande protokollet bakom TLS med DHE förser en säker etablering av autentiserade och sekretessbelagda kanaler [15].

[1] Beslutet att godkänna TLS 1.3 kom efter fyra år och det var det tjugoåttonde förslaget av TLS 1.3 som godkändes.

Några anser att det finns två svagheter med SSL/TLS. Det första är att handskakningen kräver mycket resurser på grund av all kryptering och dekryptering, det andra problemet är sårbarheten med illvilliga servrar. De kom år 2015 med ett nytt förslag på ett protokoll kallat TSSL som använder sig av TLS i kombination med en förtroendemodell [16]. Förtroendemodellen bygger på att innan klienten gör en förfrågan till en server så avgörs serverns tillförlitlighet genom att bl.a. dela in varje server i tre olika tillitsnivåer. Nivåerna är okänd server, pålitlig server samt opålitlig server. Om det är en pålitlig server kan den förminskade versionen av TLS handskakningen användas för att på så vis förbruka mindre resurser.

För att en TLS kanal ska öppnas krävs att båda enheterna är uppkopplade, eftersom en handskakning behöver utföras. Problemet med det är att asynkron kommunikation inte blir möjlig, vilket kan vara ett problem om ett meddelande skall skickas till en enhet som inte är online. Däremot kan TLS vara lämpad för kommunikation mellan enheter och servern, för att uppnå autentisering, integritet och sekretess med *forward secrecy*.

2.5 Forward secrecy

Forward secrecy är en egenskap inom kryptografi som betyder att tillfälliga krypteringsnycklar skapas ofta vilket gör att även om en krypteringsnyckel läcker ut så är omfånget i tid begränsat för användningen av den krypteringsnyckeln [17]. Signal Protocol (Signal Protocol skapat för Signal Private Messenger som beskrivs i nästa avsnitt) uppfyller *forward secrecy* för varje meddelande, vilket betyder att om krypteringsnyckeln för ett meddelande knäcks så har attackeraren ingen nytta av det för att dekryptera tidigare krypterade meddelanden. TLS har stöd för *forward secrecy* om det konfigureras rätt med vilka chiffersviter som används, med TLS 1.3 finns det bara *forward secrecy* stödda chiffersviter att välja på vid asymmetriskt baserade nyckelutbyten.

OpenPGP har inte stöd för *forward secrecy* eftersom vanligtvis används samma OpenPGP nyckelpar för flera meddelanden. För att uppnå *forward secrecy* behöver tillfälliga nycklar utbytas genom att till exempel använda Diffie-Hellman protokollet. För att uppnå *forward secrecy* med OpenPGP skulle nyckelpar behöva bytas ut regelbundet. Det finns ett förslag publicerat på IETF som bygger på det, att använda kortvariga nyckelpar för att minimera skadan om en hemlig nyckel läcker ut [18].

2.6 Befintliga applikationer

2.6.1 Telegram

Telegram är en molnbaserad meddelandeapplikation med öppen källkod för klienterna men med stängd källkod för serverapplikationen [19]. Telegram använder ett

eget krypteringsprotokoll för End-to-End kryptering kallat MTProto [20]. End-to-End kryptering är inte påslaget från början utan det kan användas genom att använda funktionen kallad Secret Chat, dock finns den funktionaliteten enbart för tvåpersonskommunicering (en-till-en). Telegram har inte stöd för End-to-End kryptering för gruppkonversationer.

Krypteringsprotokollet MTProto har fått mycket kritik och flera svagheter har hittats [21], [22]. En svaghet är att ingen autentisering görs mellan klienterna utan all tillit läggs hos servrarna, vilket gör att servrarna kan utföra en man-in-the-middle attack. En annan svaghet var att föråldrade krypteringsprimitiver används.

2.6.2 Signal Private Messenger

Signal Private Messenger är en meddelandeapplikation skapad av Open Whisper Systems. Målet med Signal Private Messenger är att göra en End-to-End krypterad meddelandeapplikation användarvänlig och lättillgänglig för allmänheten. Signal Private Messenger använder ett krypteringsprotokoll kallat Signal Protocol [23]. Signal Protocol är kryptografiskt säkert [24]. Signal Protocol anses vara implementerat i ett flertal allmänt kända applikationer, bl.a. WhatsApp, Facebook Messenger och Google Allo [24].

Klienterna som används för Signal Private Messenger är öppen källkod men är beroende av proprietär programvara från Google för att hantera notifikationer. Serverprogramvaran är även den öppen källkod dock går det inte att köra serverprogramvaran på egna servrar och kommunicera med dem offentliga servrarna. Det är möjligt att köra serverprogramvaran på en egen server och skapa ett privat kommunikationsnätverk, dock skulle servern och klienternas källkod behövas modifieras kraftigt. Eftersom källkoden är skriven att prata med Open Whisper Systems servrar samt med Googles servrar.

En avgrening kallad LibreSignal skapades från Android klienten för Signal Private Messenger med målet att vara helt Google-fri tillskillnad mot den officiella klienten. Den 5:e maj 2016 kommenterade Moxie Marlinspike som är grundaren av Open Whisper Systems på LibreSignals GitHub hemsida. Där skrev han följande [25]:

> *"I'm not OK with LibreSignal using our servers, and I'm not OK with LibreSignal using the name "Signal." You're free to use our source code for whatever you would like under the terms of the license, but you're not entitled to use our name or the service that we run."*
>
> *"We're barely able to support our own apps, and having to support products outside of our control would make our lives even more difficult."*

Efter det är LibreSignal projektet nedlagt. Fem dagar efter det, den 10:e maj 2016 skrev Moxie Marlinspike ett blogginlägg där han förklarar varför han tycker att cent-

ralisering är att föredra idag, han menar att det går betydligt snabbare att implementera nya funktioner om alla system kontrolleras av samma organisation. Han skrev följande i sitt blogginlägg [26]:

> *"When someone recently asked me about federating an unrelated communication platform into the Signal network, I told them that I thought we'd be unlikely to ever federate with clients and servers we don't control. Their retort was "that's dumb, how far would the internet have gotten without interoperable protocols defined by 3rd parties?"*
>
> *I thought about it. We got to the first production version of IP, and have been trying for the past 20 years to switch to a second production version of IP with limited success. We got to HTTP version 1.1 in 1997, and have been stuck there until now. Likewise, SMTP, IRC, DNS, XMPP, are all similarly frozen in time circa the late 1990s. To answer his question, that's how far the internet got. It got to the late 90s."*

En nackdel med centralisering är att användarna måste lita på dem som kontrollerar servrarna.

I januari 2018 hittades vissa svagheter av applikationer som använder Signal Protocol vid gruppkonversationer [27]. Signal Protocol har egentligen inga speciella funktioner för gruppkonversationer utan samma protokoll som används vid tvåpersonskommunicering används, samma meddelande skickas flera gånger, en gång till varje medlem i gruppen. Svagheten som upptäcktes handlade om hur hanteringen går till för att lägga till nya medlemmar i en gruppkonversation. I Signal Private Messenger anses varje medlem i gruppen vara administratör vilket betyder att alla medlemmar kan bjuda in nya medlemmar till gruppkonversationen. Felet innebär att det var möjligt för användare som inte var medlem i gruppkonversation ändå kunde bjuda in sig själva till gruppkonversationen. Dock krävs det att inbjudningsmeddelandet skickas till någon som är medlem i den önskade gruppkonversationen samt att gissa rätt på ett 128-bitars gruppidentifikationsnummer. WhatsApp som använder Signal Protocol visade sig även de vara sårbara mot detta, och till och med mer sårbara än Signal Private Messenger. WhatsApp hanterar gruppinbjudningar lite annorlunda, deras egna servrar används som administratörer för gruppinbjudningar och dessutom är inbjudningsmeddelanden inte End-to-End krypterade vilket dem är med Signal Private Messenger. Detta betyder alltså att WhatsApp eller någon som hackat deras servrar kan lägga till användare till vilken grupp som helst [27].

2.7 Don't roll your own crypto

Något som är känt inom kryptering är att sträva efter att använda kända, testade och välgranskade krypteringsalgoritmer [28]. Det är allmänt känt som *don't-roll-your-own-crypto* (självklart är det en annan sak att skapa egna krypteringsalgoritmer enbart för att lära sig, men detta arbete handlar inte om det). Detta eftersom det är så svårt

att veta om det finns någon svaghet eller inte. Desto längre tid en krypteringsalgoritm inte blivit bruten desto säkrare kan den anses vara, men det går aldrig att vara helt säker ändå. Nya krypteringsalgoritmer måste givetvis skapas av någon, men då ska det helst vara ett flertal professionella kryptografer inblandade och gärna ännu fler som hjälper till med granskningen.

Ett exempel på en testad och välgranskad krypteringsalgoritm är Advanced Encryption Standard (AES). I januari år 1997 utlyste National Institute of Standards and Technology (NIST) att de var ute efter en ny krypteringsalgoritm för att ersätta en tidigare mindre säker krypteringsalgoritm kallad Data Encryption Standard (DES) [29]. Femton olika krypteringsalgoritmer från olika länder tävlade under mer än tre år om vilken som var den bästa krypteringsalgoritmen. Slutligen korades Rijndael som vinnare och utsågs då till Advanced Encryption Standard [30].

3 Metod och genomförande

Detta kapitel beskriver hur arbetet har genomförts. Avsnitt 3.1 beskriver hur tidigare forskning samlats in. Avsnitt 3.2 beskriver vilka verktyg som använts. Avsnitt 3.3 beskriver vilken typ av kryptering som valts. Avsnitt 3.4 beskriver utvecklingen av prototyp.

3.1 Insamling av tidigare forskning

Litteratur samlades in från databaserna IEEE Xplore, Science Citation Index, BASE och Google Scholar. Söksträngarna som användes var ”signal private messenger”, ” signal protocol”, ”openpgp OR pgp”, ”end-to-end encryption”, ”transport layer security OR ssl OR tls”, ”mtproto AND security”, ”forward secrecy”.

3.2 Verktyg

Go (även kallat Golang) är ett programmeringsspråk skapat år 2009. Go är ett statiskt typat programspråk. Några av Go′s egenskaper är:

- Garbage collection
- Kompilerat språk med snabb kompileringstid
- Statiska binärer
- Inbyggda strukturer för hantering av parallella program
- Stort standardbibliotek

Go har flera egenskaper som lämpar sig bra för nätverksprogrammering. En sådan egenskap är *goroutines* som kan jämföras som trådar fast mycket mer lättviktigt, samt *channels* som gör kommunikation mellan *goroutines* enkelt. Go har även ett utökat standardbibliotek som har stöd för OpenPGP. Go har inget inbyggt stöd för att skapa grafiska gränssnitt (GUI) tillskillnad mot t.ex. Java som har bl.a. Swing och JavaFX. Däremot finns det ett flertal bibliotek som kan användas för att skapa grafiska gränssnitt med Go. Go har valts som programmeringsspråk för att utveckla prototyperna därför Go ansågs lämpa sig bra för projektets typ av program. Att Go inte har nativt stöd för att skapa grafiska gränssnitt är inte något stort problem eftersom det finns bibliotek för det.

Som GUI ramverk valdes Qt som är ett plattformsoberoende GUI ramverk skrivet i C++. Det finns många bindningar för Qt till andra programmeringsspråk inklusive Go.

3.3 Val av kryptering

Jag valde TLS för kommunikation mellan klient och server eftersom TLS bidrar med egenskaperna autentisering, dataintegritet och sekretess. En viktig detalj är att både klienten och servern har varsitt certifikat. Klienten litar bara på ett specifikt certifikat som tillhör servern och servern litar bara på certifikat från tillåtna klienter. Med rätt konfigurering av TLS kan även *forward secrecy* uppnås.

För kommunikation mellan klienter skulle Signalprotokollet passat bra dock skulle det tagit för lång tid att lära sig det och implementera, för Signalprotokollet behöver mycket även hanteras på serversidan. OpenPGP har varit andrahandsvalet efter Signalprotokollet. OpenPGP bidrar inte med *forward secrecy* vilket Signalprotokollet gör, dock kan nya OpenPGP nycklar genereras med jämna mellanrum för att minimera tidsramen för varje nyckelpar. OpenPGP har valts för att uppnå autentisering, dataintegritet och sekretess för kommunikation mellan klienter. OpenPGP är standardiserat och finns lättillgängligt för dem flesta programmeringsspråk.

3.4 Utveckling av prototyp

Till detta system behövs åtminstone en klient och en server. Som klient vale jag att utveckla en datorapplikation. En klient för Android hade lika gärna kunnat skapats men en datorapplikation valdes på grund av personliga skäl (mer nytta av datorapplikation).

Båda klienten och servern kommer att behöva en databas för att lagra data såsom TLS certifikat och meddelanden i. Som databassystem valde jag att använda SQLite eftersom SQLite är enkelt att bädda in i ett program och har tillräckligt hög prestanda för detta system.

Under utveckling har Git använts för versionshantering.

3.5 Källkod

Källkod för arbetet finns tillgänglig på begäran.

4 Resultat

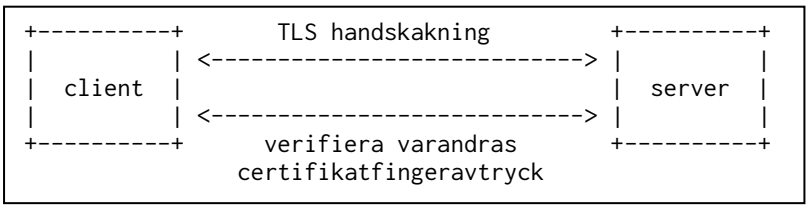
En klient i form av en datorapplikation och en serverapplikation har skapats.

4.1 Kommunikation mellan klient och server

Den första kommunikationen som sker är den mellan klient och server. För kommunikation mellan klient och server används TLS vilket bidrar med dataintegritet, sekretess, autentisering och *forward secrecy* vid rätt konfigurering. Vanligtvis med TLS så skickar inte klienten något certifikat utan det är bara serverns certifikat som verifieras av klienten. Det brukar även vara så att det är certifikatkedjor som verifieras. För detta system behöver både klienten och servern skicka varsitt certifikat så att de kan verifiera varandra. Verifieringen använder inte certifikatkedjor utan klienter och servern väljer specifikt vilka certifikat de litar på.

4.1.1 Öppna anslutning

Klienten öppnar en anslutning till servern vilket gör att en TLS handskakning genomförs. Efter TLS handskakningen verifierar både klienten och servern om fingeravtrycket från den andra partens certifikat stämmer överens med vad som förväntas, om inte så stängs anslutningen direkt.



Figur 4.1: Översikt öppning av anslutning

Konfigurationen för TLS (figur 4.2) är viktig. TLS behöver konfigureras eftersom:

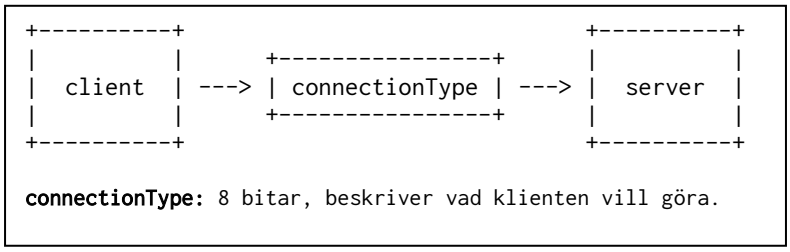
- Gamla TLS eller SSL versioner inte skall tillåts
- Bara chiffersviter med *forward secrecy* skall användas
- Certifikat krävs från både servern och klienten

```
config := &tls.Config{
   MinVersion:              tls.VersionTLS12,
   CipherSuites:            []uint16{tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384},
   Certificates:            []tls.Certificate{*serverCert},
   ClientAuth:              tls.RequireAnyClientCert,
   InsecureSkipVerify:      true,
   VerifyPeerCertificate:   verifyClientCertificate,
   SessionTicketsDisabled:  true,
}
```

Figur 4.2: Server TLS konfigurering

4.1.2 Öppen anslutning

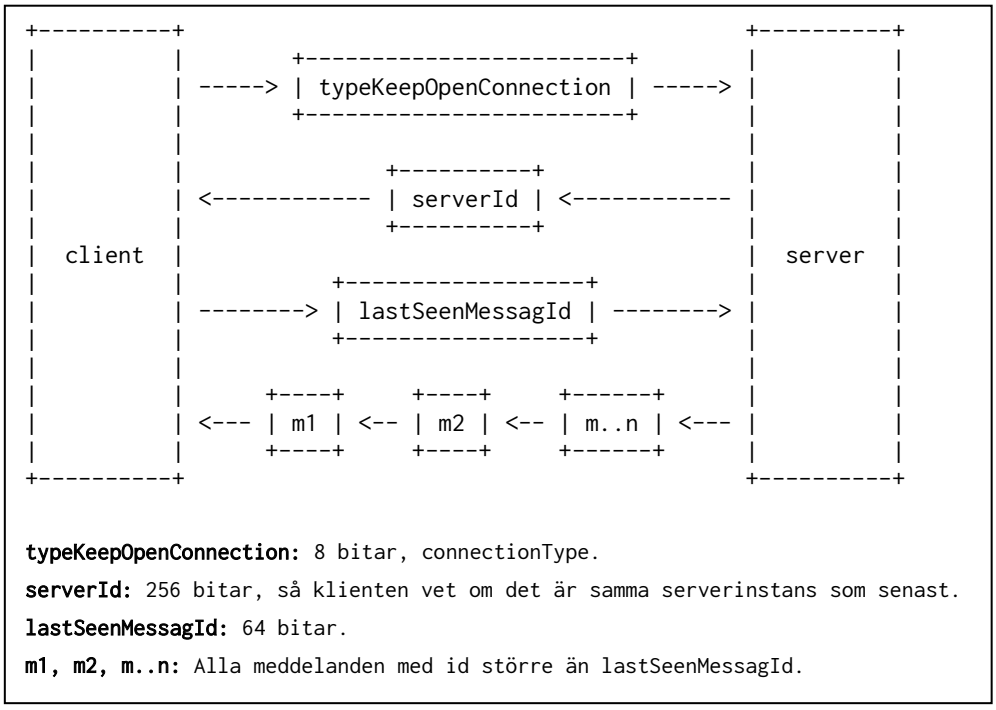
Efter att anslutningen mellan klienten och servern är öppen så kan klienten utföra några olika saker (figur 4.3).



Figur 4.3: Klienten väljer

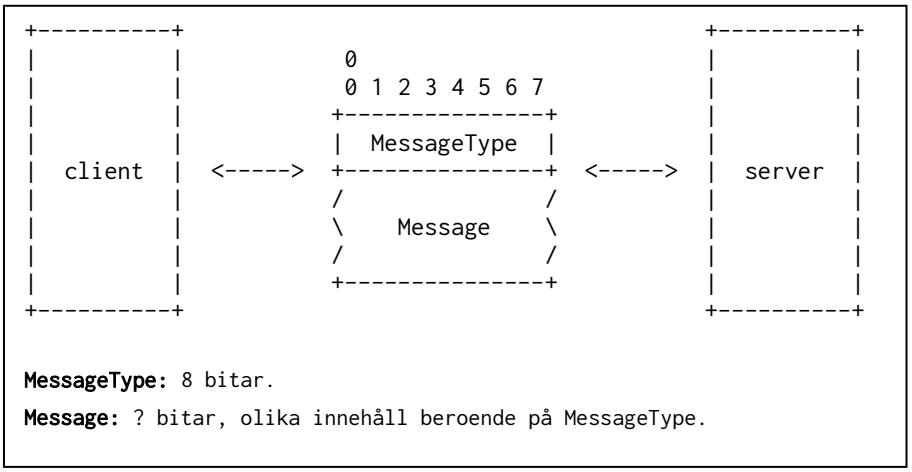
En vanlig sak som klienten vill göra är att öppna en anslutning och låta den anslutningen vara öppen. Denna anslutning (kallad *keepOpenConnection*) kan användas för att skicka nya meddelanden till servern samt att mottaga meddelanden från andra klienter via servern. Anslutningen används även för att mottaga uppdateringar om meddelande-id från servern.

När klienten öppnar en anslutning till servern vill klienten hämta hem meddelanden som klienten inte har sett ännu, utifall sådana meddelanden existerar. Klienten skickar då vilket id det nyaste meddelandet klienten sett har så servern vet vilka meddelanden den ska skicka (figur 4.4).



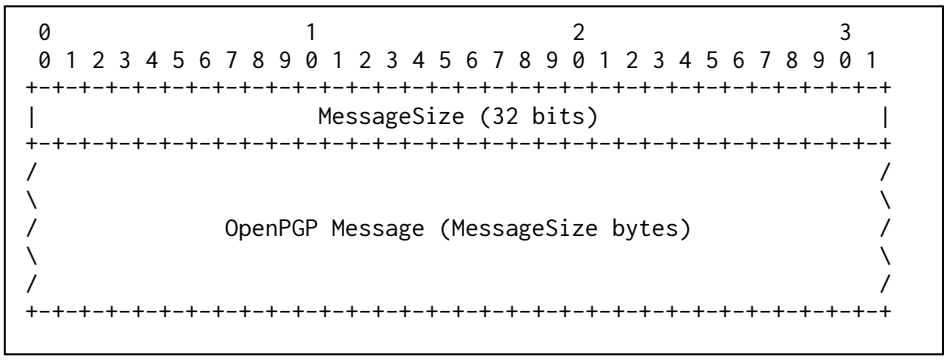
Figur 4.4: Klienten öppnar keepOpenConnection

Det finns flera olika typer av meddelanden som kan skickas mellan klienten och servern vid en öppen anslutning, därför behöver meddelandetyp skickas med (figur 4.5).



Figur 4.5: Meddelandetyp

En av meddelandetyperna är ett vanligt meddelande som ska visas av klienterna. Innehållet av ett sådant meddelande är ett signerat och krypterat OpenPGP meddelande (figur 4.6). *OpenPGP Message* är krypterat med andra klienters publika OpenPGP nycklar vilket gör att servern inte kan dekryptera eller ändra innehållet i meddelandet, eftersom servern inte har tillgång till någon av de privata OpenPGP nycklarna.

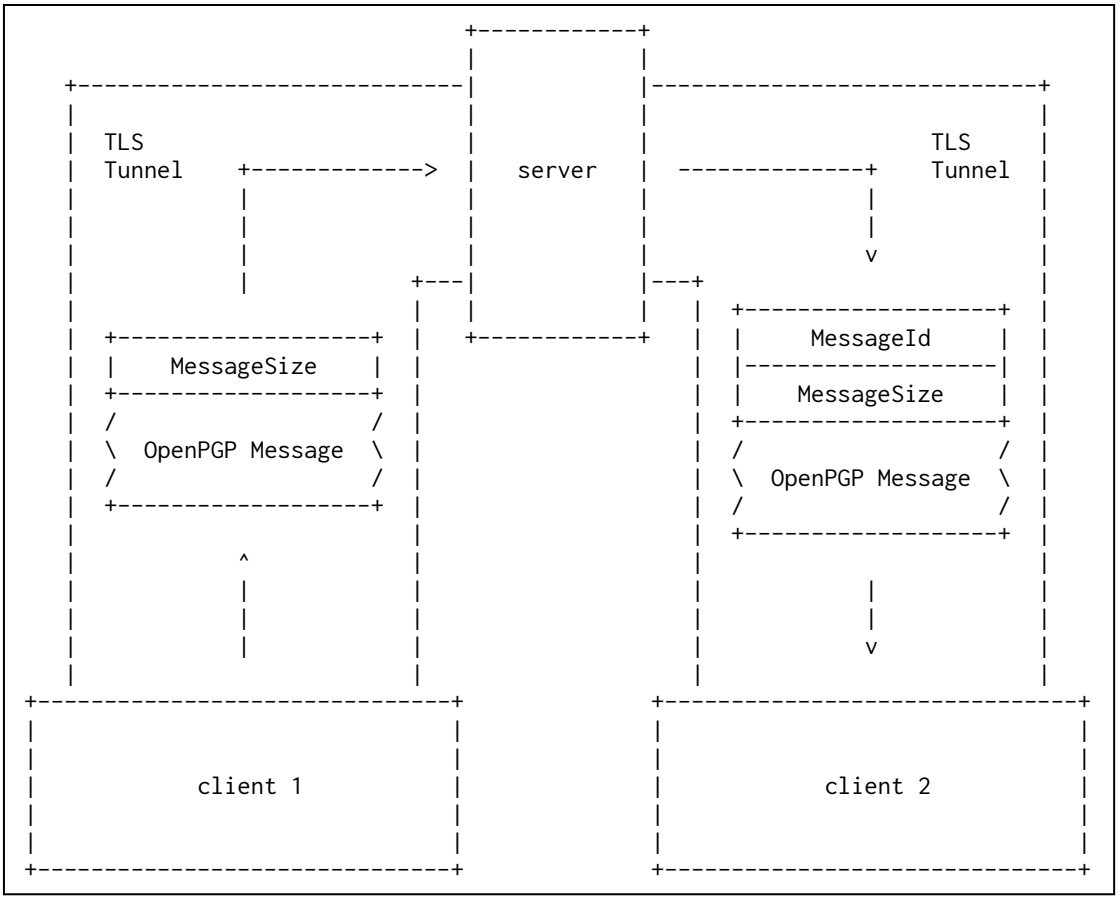


Figur 4.6: OpenPGP meddelande, vad servern ser

Servern spar alla inkommande meddelanden i sin databas och vidarebefordrar det till andra anslutna klienter om sådana finns. Den enda metadata som servern lägger till för varje meddelande är ett uppräknande 64-bitars unikt id-nummer. Detta nummer skickas till klienterna för varje meddelande de tar emot, så att de vid nästa nya anslutning till servern kan tala om vilket det senaste meddelandet de sett är.

4.2 Kommunikation mellan klienter

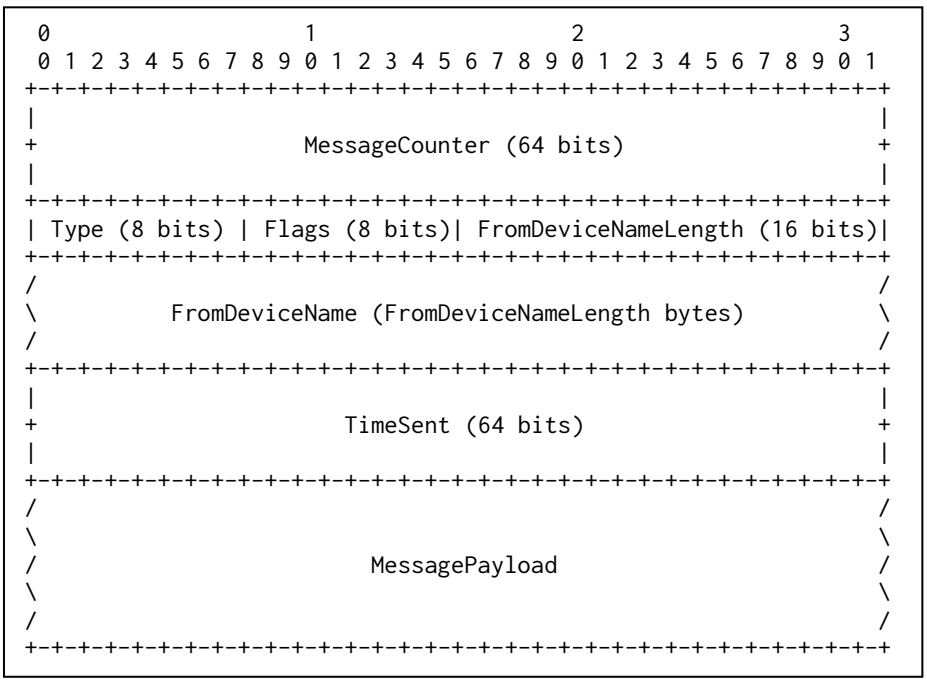
Kommunikation mellan klienter sker via servern, servern agerar mellanhand åt klienterna (figur 4.7).



Figur 4.7: Överblick kommunikation mellan klienter

Trots att all kommunikation är krypterad mellan klienten och servern med TLS så behöver OpenPGP användas för att uppnå End-to-End kryptering mellan klienterna. Detta gör att klienterna behöver signera och kryptera meddelandena själva. Servern spar alltså OpenPGP meddelanden i sin databas som sedan autentiserade klienter kan ladda ned.

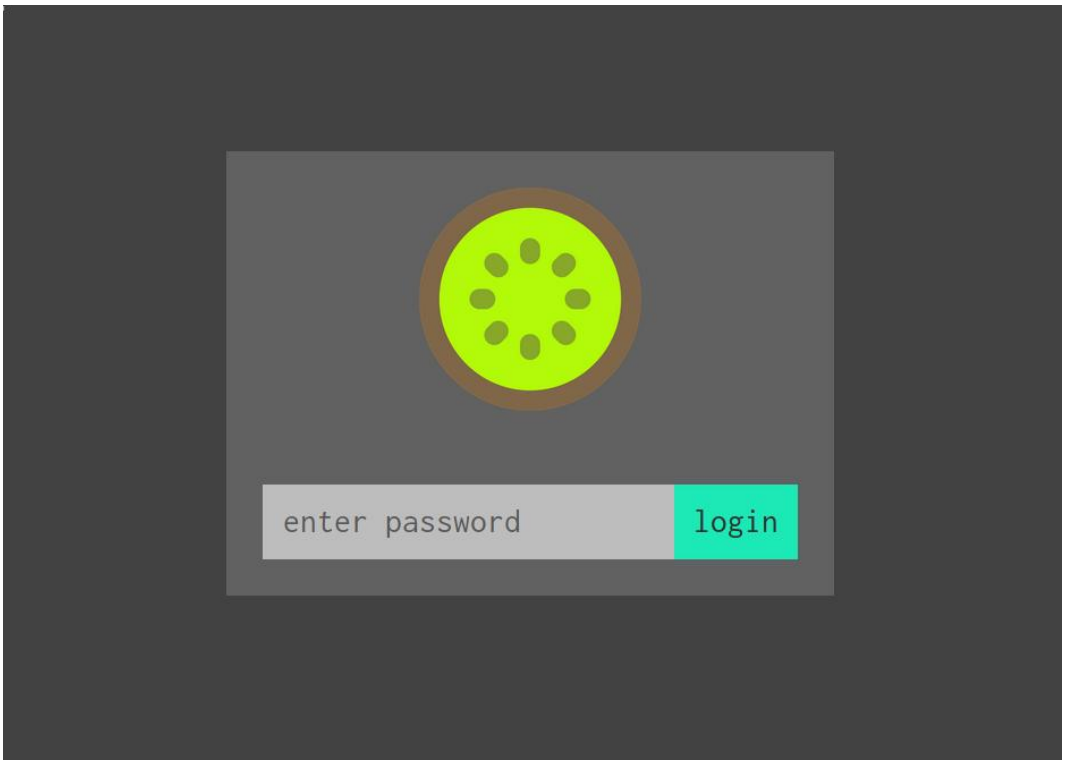
OpenPGP Message (figur 4.7) innehåller nyttolasten och krypterad metadata som behövs av klienterna (figur 4.8). Eftersom OpenPGP inte har skydd mot *replay-attacker* så används en räknare för att motverka det (*MessageCounter*). För varje meddelande som skickas av en klient så ökas räknaren med ett, det gör att mottagaren kan kontrollera så att meddelandet som tas emot har ett räknarvärde som är ett större än det förra meddelandet som togs emot av den avsändaren.



Figur 4.8: Innehåll OpenPGP Message

4.3 Serverprototyp

Servern konfigureras genom ett webbaserat GUI där användaren behöver logga in med ett lösenord (figur 4.9), lösenordet finns hashat (med bcrypt) i serverns databas. Ingen sessionshantering för webbinloggningen finns på serversidan utan istället används en cookie som är krypterad och signerad.



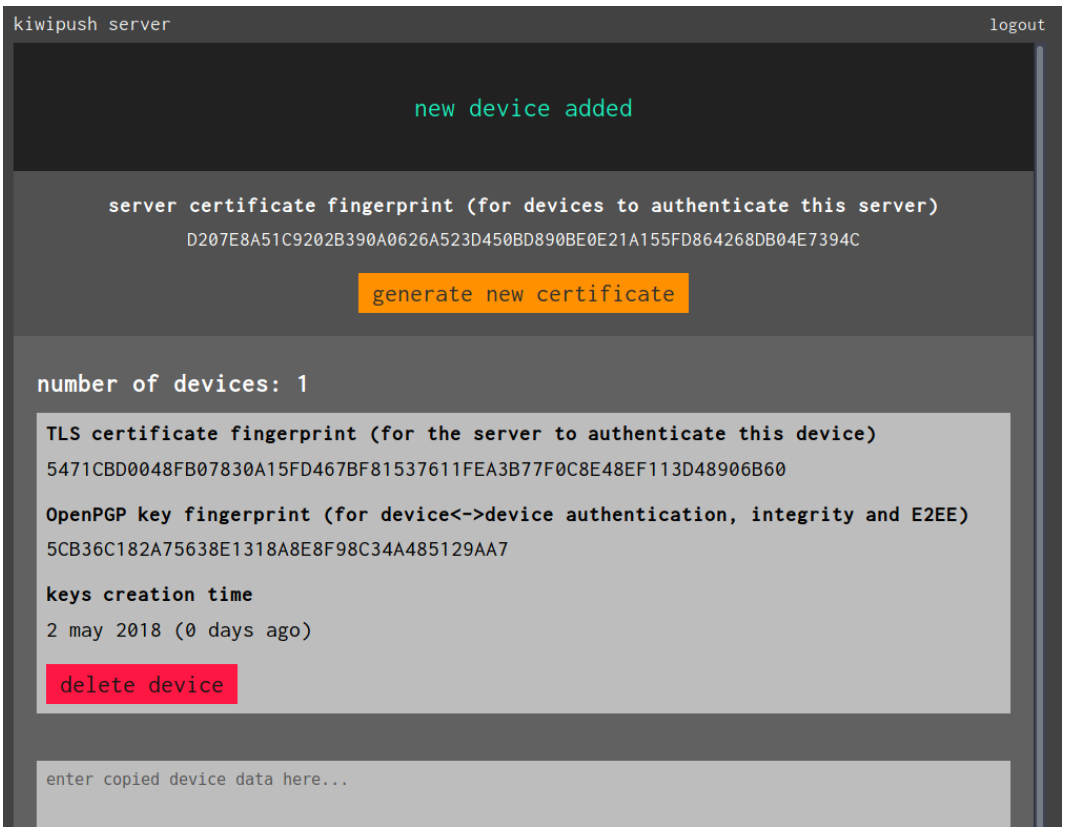
Figur 4.9: Inloggning serverapplikation

Konfigurering som kan utföras på servern är (figur 4.10):

- Generera ett nytt nyckelpar med tillhörande TLS certifikat
- Lägga till och ta bort klienter
- Ändra vilken port servern och webbservern lyssnar på
- Ändra inloggningslösenordet

Serverns GUI behöver även visa fingeravtrycket för dess TLS certifikat, så att klienter kan kopiera och verifiera det. För att lägga till klienter används ett textfält som förväntar sig ett TLS fingeravtryck och den publika OpenPGP nyckeln hopslaget och BASE64 kodat (figur 4.11).

Figur 4.10: Inloggad på serverns webb GUI

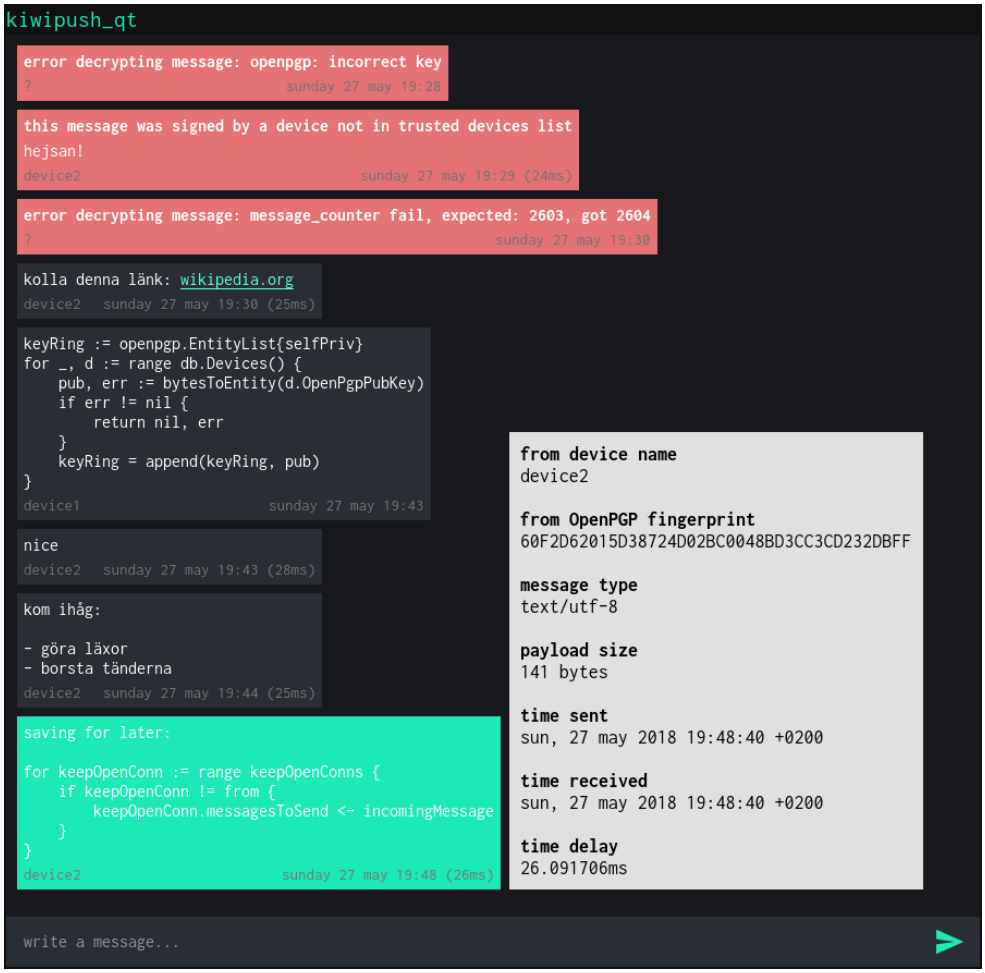


Figur 4.11: Klient tillagd till servern

4.4 Klientprototyp

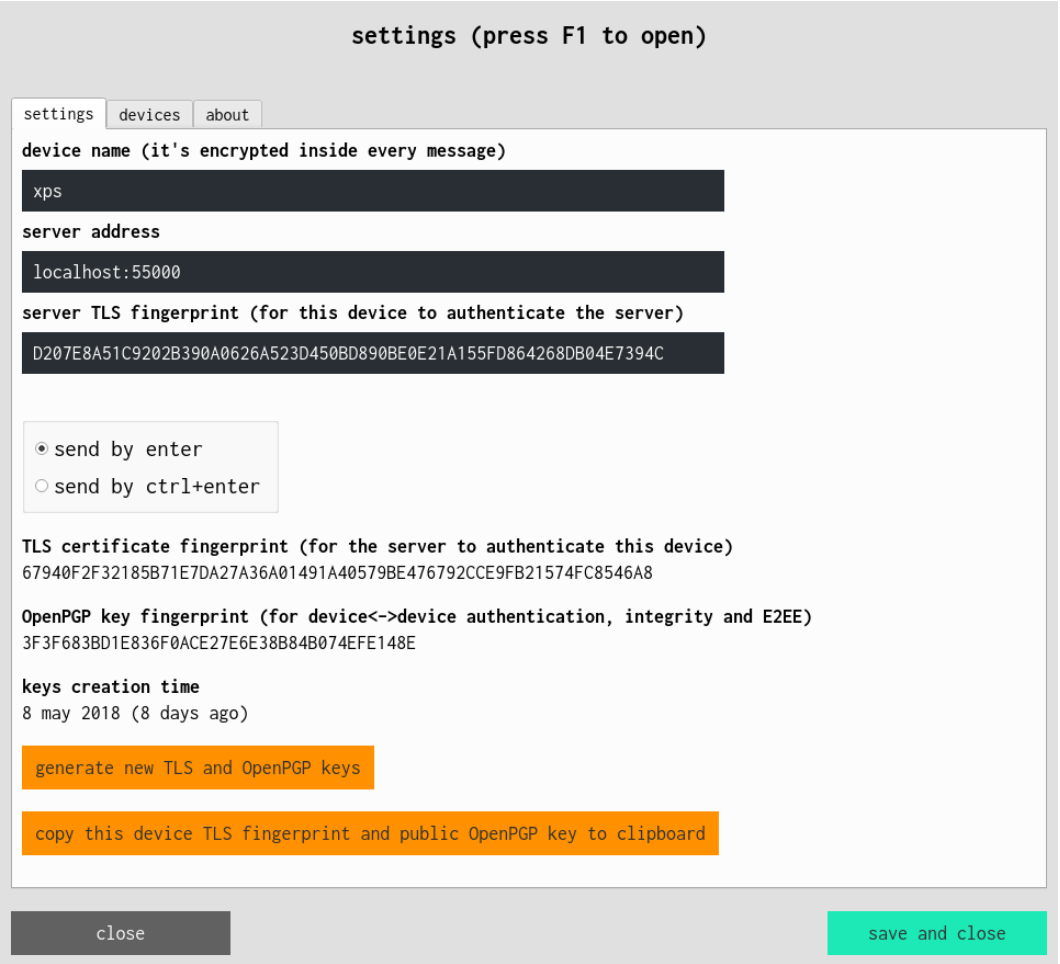
När ett meddelande skickas från servern till klienten så försöker klienten först att dekryptera det. Meddelandet måste dekrypteras innan signaturen kan kontrolleras eftersom meddelandena är signerade först och sedan krypterade. Om klienten misslyckas att dekryptera meddelandet så visas det hos klienten som ett felmeddelande (*incorrect key*, figur 4.12).

Om klienten lyckas att dekryptera meddelandet så kontrolleras att signaturen är korrekt samt om signaturen är skapad av någon av nycklarna som klienten har i sin nyckelring. Tillslut kontrolleras även meddelanderäknaren för att upptäcka eventuella *replay-attacker*.



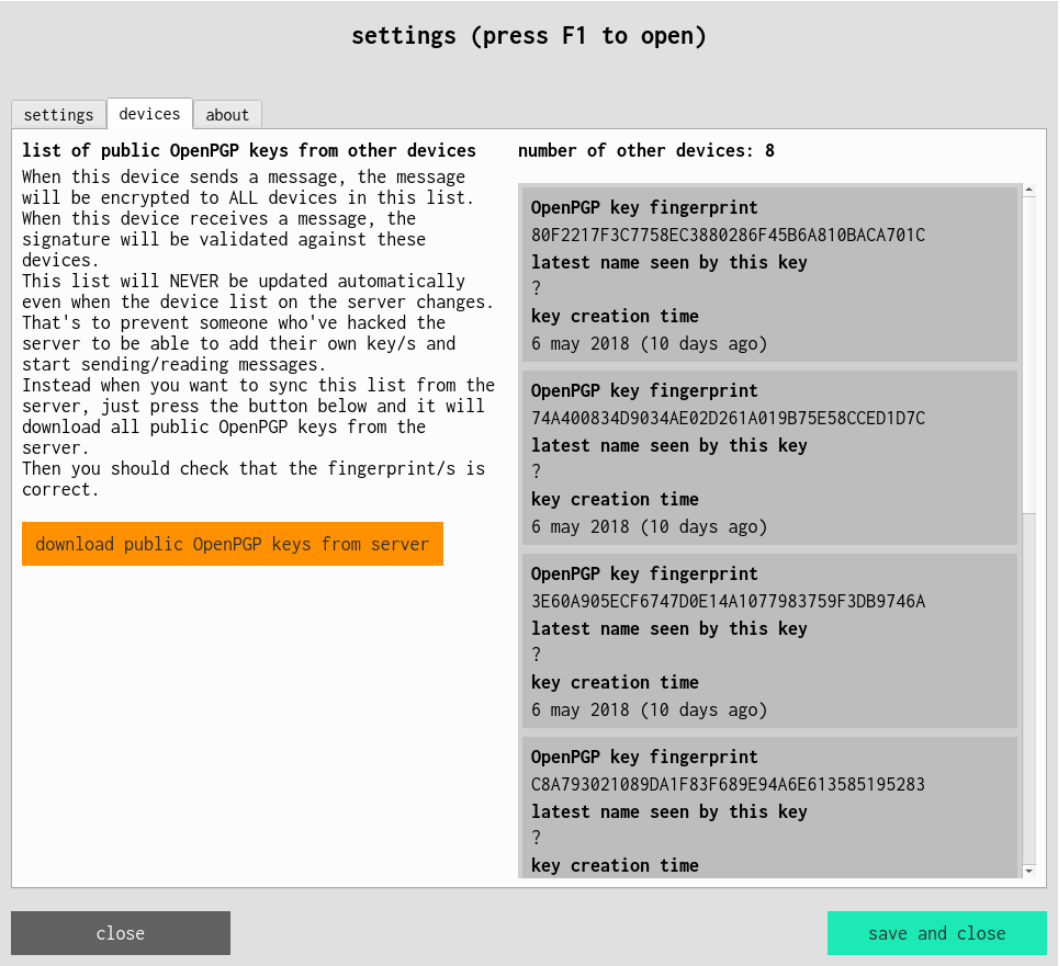
Figur 4.12. Klientapplikation med flera olika meddelanden.

Klienterna behöver konfigurera (figur 4.13) vilken adress servern har och välja värdet av fingeravtrycket för serverns TLS certifikat, så klienten kan veta att den pratar med den förväntade servern. Klienten kan exportera klientens fingeravtryck för TLS certifikatet samt den publika OpenPGP nyckeln BASE64 kodat, vilket sedan används för att lägga till klienten till servern.



Figur 4.13: Klientkonfiguration

Varje klient behöver ladda ned dem öppna OpenPGP nycklarna som är tillagda till servern (figur 4.14). Dessa nycklar används sedan av klienterna när de ska kryptera meddelanden och när de ska verifiera signaturer av inkommande meddelanden.



Figur 4.14: Klientkonfiguration

5 Diskussion och slutsatser

Sammanfattningsvis var målet att skapa ett meddelandesystem bestående av åtminstone en klient och en server där systemet uppfyller vissa krav. Ett system har skapats som uppfyller grundkraven angivna under frågeställningarna.

Nedan ges svar till frågeställningarna från avsnitt 1.3.

1. Hur kan End-to-End kryptering mellan klienter uppnås?

   Genom att använda OpenPGP. Klienter lägger till sig själva hos servern vilket gör att deras öppna OpenPGP nycklar sparas där. Klienter laddar hem andra klienters öppna OpenPGP nycklar från servern. Innan meddelanden skickas till servern krypteras dem med dem andra klienternas publika OpenPGP nycklar.

2. Hur kan dataintegritet och autentisering mellan klienter uppnås?

   Meddelanden signeras med OpenPGP innan dem krypteras. När en klient mottager ett meddelande kontrolleras att signaturen är korrekt samt att signaturen är skapad av någon av nycklarna som de laddat ned från servern.

3. Hur kan sekretess, dataintegritet och autentisering uppnås mellan klient och server?

   Genom att använda TLS där både klienten och servern har varsitt certifikat. Servern och klienten väljer specifikt vilka certifikatfingeravtryck de litar på. Direkt efter TLS handskakningen kontrolleras fingeravtrycket, om det inte stämmer överens med vad som förväntas avbryts anslutningen.

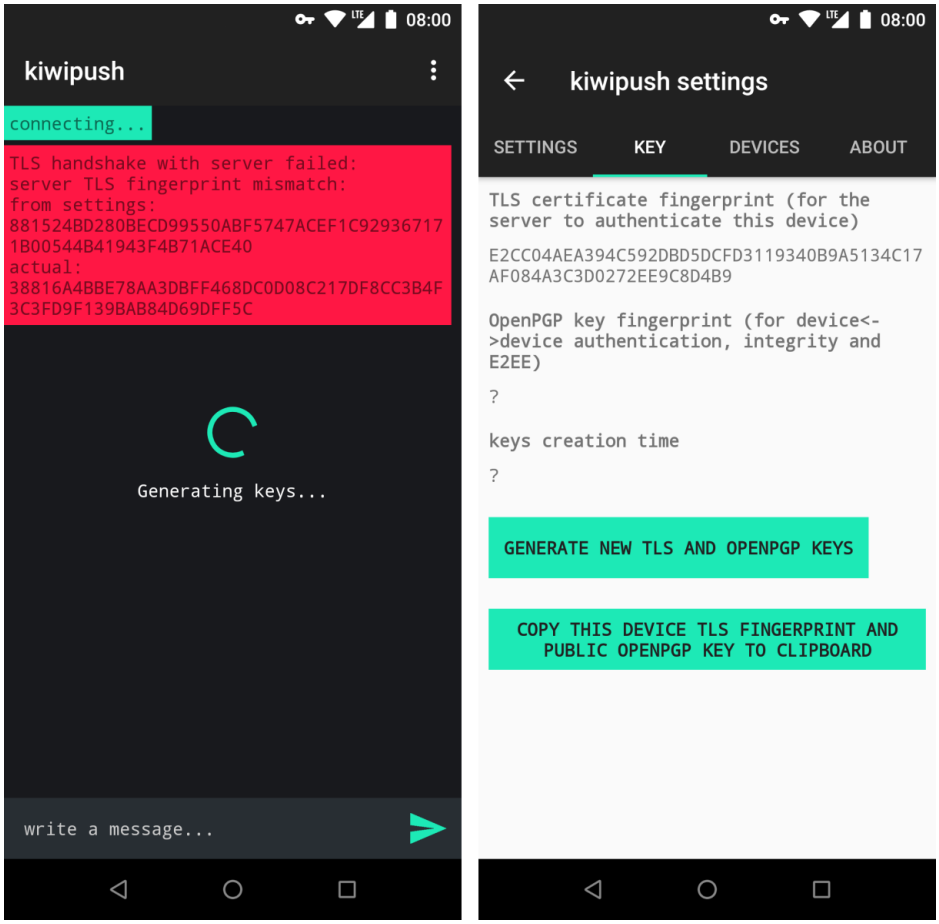
Ett annat mål var att minimera mängden metadata (i klartext) som tillhandahålls hos servern. Den enda metadata som sparas av servern är att för varje inkommande meddelande så sammankopplar servern ett unikt uppräknande id-nummer till det meddelandet. Utöver det har även servern tillgång till alla öppna OpenPGP nycklar, vilket är en funktion som erhålls av servern så att klienterna kan ladda hem andra klienters öppna OpenPGP nycklar. Jag ser inget problem med att servern har tillgång till dessa öppna OpenPGP nycklar, det är det som är grejen med asymmetrisk kryptering. De öppna (publika) nycklarna kan spridas öppet utan någon egentlig säkerhetsrisk, det är i alla fall tanken. Sen är det även så att klienterna själva kontrollerar signaturerna för varje inkommande meddelande, och för att skapa en signatur krävs den hemliga (privata) OpenPGP nyckeln vilket servern inte har tillgång till. Klienterna kan även kontrollera att fingeravtrycket för nycklarna stämmer överens.

TLS och OpenPGP är båda kända och välgranskade protokoll. De är även standardiserade vilket gör att de flesta programmeringsspråk har på ett enkelt sätt stöd för

dem. Signalprotokollet hade varit en intressant ersättare för OpenPGP i detta system, vilket är välgranskat men dock inte lika standardiserat ännu. Om Signalprotokollet hade varit standardiserat och mer lättillgängligt hade jag nog valt det. Dock hade det antagligen tagit längre tid att sätta sig in i hur det fungerar och vad som behöver göras på serversidan osv. Hade jag fått börja om från början med detta arbete med lika mycket tid idag hade jag fortsatt att välja OpenPGP, hade jag däremot haft längre tid (säg ett halvår) så hade jag åtminstone i början undersökt Signalprotokollet mer och provat utifall någon lösning varit möjlig.

Valet att använda Go som primärt programmeringsspråk var lite av en chansning, mestadels på grund av hur det skulle gå med Qt som GUI ramverk. Nu i efterhand anser jag att jag gjorde rätt val, Go passar väldigt bra för nätverksprogrammering tycker jag. Go har även i sitt utökade standardbibliotek stöd för OpenPGP.

Projektet hade avgränsningen att prioritera en datorapplikation före en Android applikation. Dock har en prototyp för Android påbörjats den senaste veckan (figur 5.1). Prototypen har inte stöd för att skicka eller ta emot meddelanden ännu. För Android applikationen experimenterar jag med Android NDK som gör det möjligt att exekvera nativ kod. Det gör att utöver Java så kan Go användas i vissa delar av applikationen vilket gör att en del kod från datorapplikationen kan återanvändas.



Figur 5.1: Prototyp för Android

Sett ur ett större perspektiv skulle detta arbete kunna leda till förbättrad psykisk hälsa eftersom användaren kan vara säker på hur data och metadata hanteras. Jag anser det vara etiskt fel att spionera på människors privatliv. Detta projekt kan orsaka större personliga kostnader ekonomiskt eftersom det kan vara dyrare att driva en egen server jämfört med att använda gratistjänster. Det är dock ofta så att om någonting är gratis så är användaren och dess privatliv själva produkten. Miljömässiga aspekter kring detta arbete anser jag vara irrelevanta.

Referenser

[1] E. Snowden, ”SuddenlySnowden comments on Just days left to kill mass surveillance under Section 215 of the Patriot Act. We are Edward Snowden and the ACLU’s Jameel Jaffer. AUA.”, *Reddit*, 2015. [Online]. Tillgänglig vid: https://www.reddit.com/r/IAmA/comments/36ru89/just_days_left_to_kill_mass_surveillance_under/crglgh2/. [Åtkomstdatum: 09-apr-2018].

[2] P. Zimmermann, ”Why I Wrote PGP”, *Phil Zimmermann’s Home Page*, 1991. [Online]. Tillgänglig vid: https://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html. [Åtkomstdatum: 09-apr-2018].

[3] A. Barenghi, N. Mainardi, och G. Pelosi, ”A Security Audit of the OpenPGP Format”, *2017 14th International Symposium on Pervasive Systems, Algorithms and Networks &2017 11th International Conference on Frontier of Computer Science and Technology &2017 Third International Symposium of Creative Computing (ISPAN-FCST-ISCC), Pervasive Systems, Algorithms and Networks &2017 11th International Conference on Frontier of Computer Science and Technology &2017 Third International Symposium of Creative Computing (ISPAN-FCST-ISCC), 2017 14th International Symposium on, ISPAN-FCST-ISCC*, s. 336, 2017.

[4] D. Poddebniak *m.fl.*, ”Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels (draft 0.9.1)”, *EFAIL*, maj-2018. [Online]. Tillgänglig vid: https://efail.de/efail-attack-paper.pdf.

[5] A. Yen, ”ProtonMail Blog”, *Statement from PGP developers about eFail*, maj-2018. [Online]. Tillgänglig vid: https://protonmail.com/blog/pgp-efail-statement/.

[6] A. Yen, ”ProtonMail Blog”, *No, PGP is not broken, not even with the Efail vulnerabilities*, maj-2018. [Online]. Tillgänglig vid: https://protonmail.com/blog/pgp-vulnerability-efail/.

[7] R. J. Hansen, ”GnuPG Mailing list”, *Efail press release*, maj-2018. [Online]. Tillgänglig vid: https://lists.gnupg.org/pipermail/gnupg-users/2018-May/060334.html.

[8] Internet Engineering Task Force (IETF), ”OpenPGP Message Format”, *RFC 4880*, nov-2007. [Online]. Tillgänglig vid: https://tools.ietf.org/html/rfc4880. [Åtkomstdatum: 24-apr-2018].

[9] S. More, ”Java Privacy Guard - The OpenPGP Message Format and an Implementation in Java”, *Bachelor's Thesis, Germany, Graz University of Technology*, 2015.

[10] Red Hat, ”POODLE: SSLv3 vulnerability (CVE-2014-3566)”, *Red Hat Customer Portal*, 2014. [Online]. Tillgänglig vid: https://access.redhat.com/articles/1232123. [Åtkomstdatum: 09-apr-2018].

[11] Internet Engineering Task Force (IETF), ”Prohibiting Secure Sockets Layer (SSL) Version 2.0”, *RFC 6176*, mar-2011. [Online]. Tillgänglig vid: https://tools.ietf.org/html/rfc6176. [Åtkomstdatum: 09-apr-2018].

[12] Internet Engineering Task Force (IETF), ”Deprecating Secure Sockets Layer Version 3.0”, *RFC 7568*, juni-2015. [Online]. Tillgänglig vid: https://tools.ietf.org/html/rfc7568. [Åtkomstdatum: 09-apr-2018].

[13] S. Turner, ”Transport Layer Security”, *IEEE Internet Computing, Internet Computing, IEEE, IEEE Internet Comput.*, nr 6, s. 60, 2014.

[14]Internet Engineering Task Force (IETF), ”Protocol Action: ’The Transport Layer Security (TLS) Protocol Version 1.3’ to Proposed Standard (draft-ietf-tls-tls13-28.txt)”, *Internet Engineering Task Force (IETF)*, 21-mar-2018. [Online]. Tillgänglig vid: https://www.ietf.org/mail-archive/web/ietf-announce/current/msg17592.html. [Åtkomstdatum: 09-apr-2018].

[15]T. Jager, F. Kohlar, S. Schaege, och J. Schwenk, ”Authenticated Confidential Channel Establishment and the Security of TLS-DHE”, *JOURNAL OF CRYPTOLOGY*, vol. 30, nr 4, s. 1276–1324, okt. 2017.

[16]M. A. Kaljahi, A. Payandeh, och M. B. Ghaznavi-Ghoushchi, ”TSSL: improving SSL/TLS protocol by trust model”, *SECURITY AND COMMUNICATION NETWORKS*, vol. 8, nr 9, s. 1659–1671, juni 2015.

[17]Y. Suga, ”SSL/TLS Servers Status Survey about Enabling Forward Secrecy”, *2014 17th International Conference on Network-Based Information Systems, Network-Based Information Systems (NBiS), 2014 17th International Conference on, Network-Based Information Systems (NBiS), 2013 16th International Conference on*, s. 501, 2014.

[18]I. Brown, ”draft-brown-pgp-pfs-03 - Forward Secrecy Extensions for OpenPGP”, *Internet Engineering Task Force (IETF) Internet Draft*, okt-2001. [Online]. Tillgänglig vid: https://tools.ietf.org/html/draft-brown-pgp-pfs-03. [Åtkomstdatum: 09-apr-2018].

[19]Telegram, ”Telegram FAQ”, 09-apr-2018. [Online]. Tillgänglig vid: https://telegram.org/faq#q-why-not-open-source-everything. [Åtkomstdatum: 09-apr-2018].

[20]Telegram, ”MTProto Mobile Protocol”, 09-apr-2018. [Online]. Tillgänglig vid: https://core.telegram.org/mtproto. [Åtkomstdatum: 09-apr-2018].

[21]J. Jakobsen och C. Orlandi, ”On the CCA (in)security of MTProto”, i *SPSM 2016 - Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices, co-located with CCS 2016*, 2016, s. 113–116.

[22]G. Couprie, ”Telegram, AKA “Stand back, we have Math PhDs!””, *Unhandled expression*, 2013. [Online]. Tillgänglig vid: https://unhandledexpression.com/2013/12/17/telegram-stand-back-we-know-maths/. [Åtkomstdatum: 09-apr-2018].

[23]Open Whisper Systems, ”Technical information. Specifications. XEdDSA and VXEdDSA. X3DH. Double Ratchet. Sesame.”, *Signal >> Documentation*, 09-apr-2018. [Online]. Tillgänglig vid: https://signal.org/docs/. [Åtkomstdatum: 09-apr-2018].

[24]K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, och D. Stebila, ”A Formal Security Analysis of the Signal Messaging Protocol”, *2017 IEEE European Symposium on Security and Privacy (EuroS&P), Security and Privacy (EuroS&P), 2017 IEEE European Symposium on, EUROS-P*, s. 451, 2017.

[25]M. Marlinspike, ”Please add LibreSignal to f-droid”, *GitHub*, 05-maj-2016. [Online]. Tillgänglig vid: https://github.com/LibreSignal/LibreSignal/issues/37#issuecomment-217211165. [Åtkomstdatum: 09-apr-2018].

[26]M. Marlinspike, ”Reflections: The ecosystem is moving”, *Signal blog*, 10-maj-2016. [Online]. Tillgänglig vid: https://signal.org/blog/the-ecosystem-is-moving/. [Åtkomstdatum: 09-apr-2018].
[27]P. Rösler, C. Mainka, och S. Jörg, ”More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema”, *3rd IEEE European Symposium on Security and Privacy (EuroS&P 2018)*, jan. 2018.
[28]B. Schneier, ”Amateurs Produce Amateur Cryptography”, *Schneier on Security*, 2015. [Online]. Tillgänglig vid: https://www.schneier.com/blog/archives/2015/05/amateurs_produc.html. [Åtkomstdatum: 24-maj-2018].
[29]NIST Computer Security Resource Center, ”Announcing Development of a Federal Information Processing Standard for Advanced Encryption Standard”, 02-jan-1997. [Online]. Tillgänglig vid: https://csrc.nist.gov/news/1997/announcing-development-of-fips-for-advanced-encryp. [Åtkomstdatum: 09-apr-2018].
[30]NIST, ”Commerce Department Announces Winner of Global Information Security Competition”, 02-okt-2000. [Online]. Tillgänglig vid: https://www.nist.gov/news-events/news/2000/10/commerce-department-announces-winner-global-information-security. [Åtkomstdatum: 09-apr-2018].