Exploration of the criteria for designing an accessible secure messenger

Mr M.J.W Sellman

Goldsmiths

University of London

A thesis submitted for the degree of

Bachelors of Science (Honors) Computer Science

May 2017

Acknowledgements

Personal

I would like to thank my supervisor, Dr. Ida Pu, for her valuable help and advice during the course of the project. Her help and advice was invaluable at many stages and the regular meetings she set up helped inspire me in my work.

Institutional

Abstract

Currently, there are few open source secure messengers which are developer friendly to individuals with a limited security background. This dissertation explores the requirements which would be needed for developing a secure messaging service. The aim of this research is to explore what criteria and challenges must be considered in order to provide alternatives for services such as WhatsApp and WeChat if they were to become compromised. This involves the creation of a secure system through the use of encryption over an Internet Relay Chat server with an emphasis on maintaining the confidentiality of messages. Through the use of AES encryption and Diffie-Hellman key generation this is explored. Having designed and implemented a prototype, this was then tested for security and user experience. The success of the user focused development testing was shown to be highly correlated to having a strong computer science background, while the results from the security testing showed a robust system.

Contents

List of Figures

List of Abbreviations

IRC	Internet Relay Chat
DH	Diffie Hellman Merkle Key Exchange
AES	Advanced Encryption Standard
RSA	Rivest, Shamir, Adleman
PKI	Public Key Infrastructure
KDF	Key Derivation Function
IDS	Intrusion Detection System
XMPP	Extensible Messaging and Presence Protocol
Nmap	Network Mapper

1 Introduction

Contents

1.1 Overview of the System

This research investigates the requirements that should be made in designing an open source secure chat client. Specifically this work looks into implementing encryption for an Internet Relay Chat (IRC) messenger in order to ensure security for users. The project will explore the value of confidentiality in developing user-friendly decentralised encryption of the client side messenger while sending messages over an insecure network.

1.1.1 Security focus

In this research the preservation of confidentiality is investigated by focusing upon the utilisation of the Diffie-Hellman generation and key exchange [**Diffie**, **1979**] in addition to AES symmetric key encryption and the securing of a Linux server. There are currently few fully open source encrypted IRC chat clients that can provide the user with end-to-end encryption [**Arlolra**, **2015**]. This project seeks to redress that issue by making a decentralised system that can be expanded and open to customisation by users.

1.2 Motivation

In 2015 the IT security market was valued at around $75 billion dollars [**Morgan**, **2015**] which can be attributed to how important individuals and institutions believe it is to mitigate security risks. Confidence in privacy and security are essential for many industries that deal with sensitive information. For example, security is a vital part of managing health data due to the personal nature of the information [**Bodkin**, **2017**], while security for maintaining civil rights by ensuring citizens have the means to communicate privately is also of great importance [**Warren**, **1890**]. Due to the progression of security attacks, it is becoming better realised how security weaknesses can result in undesirable and potentially financially costly problems [**Campbell**, **2003**]. It is because of this greater awareness that computer security is becoming an increasingly popular subject of discussion, despite a shortage of skilled security professionals [**Bednarz**, **2015**].

This project is focused upon the decentralization of a secure means of messaging. Although many services with user bases of millions are useful, if they are compromised few options are available as a replacement. A solution is for users to create their own secure messengers, however the learning curve for this is steep as many principles must be learned and implemented reliably. This work seeks to understand what requirements would be necessary to ensure that inexperienced users can adopt an independent means of communication. The intention is to develop an open source chat client which lies between the two extremes of entirely user facing, and independently developed. The learning curve for users can become less steep such that users can work from a basic secure chat client before further customizing the client to their needs. This ensures that if some messengers become compromised by governmental or legal means, alternative options are accessible to individuals in need [**Sparrow**, **2017**].

1.3 Research Question

The research question asks:

What considerations should be made when developing a secure open source chat messenger.

This question hopes to focus on the needs of the user when it comes to the ease of use and accessibility in creating a cryptographic program.

This can be split into sub sections that provide a clearer focus and smaller scope for the project duration.

The first sub-question is: What kind of audience and demographic should be considered as users?
This will help find out who would want to use the program and why. It will in turn help to make a system that will be catered for their needs.

The second sub-question is: What kind of hardware system would this research be developed for?
This will clarify the hardware suitability for the software. Which can then be better

suited for optimisation.

The third sub-question is: What kind of developments can be made to expand this project?
This will include future work that can be done on the project in order to create a coherent system.

These research questions are broad but have been specified and narrowed by the approaches used, and also by limiting the range of options explored. Each question in turn works to address the main research question and thus shapes the direction of the research. The methodology of using a descriptive study followed by a prescriptive study means that having gained an initial understanding, interactions can provide data which will help answer these questions.

1.4 Methodology

There are multiple ways to preserve confidentiality through cryptography. In order to narrow the scope of this research, and create a clear direction, a framework has been used to help structure this project.

The first step begins with an investigation into the brief technical background of popular messengers in order to understand the range of technical approaches used. The second step has been in exploring how to implement the theory discovered and developed in the initial step. The next section of work having designed the research is to implement the theory fully by developing and iterating over any bugs or project challenges. The prototype is then tested against users to gather feedback. The data gathered is analysed. Finally the processes and data gathered through this project are reflected upon to summarise findings and seek improvements or future developments for this project.

The figure below details the time frames developed for this project

1.5 Aims

This dissertation will explore one of the options of the secure encryption of messages in an Internet Relay Chat. The aim of this project is to better understand and explore cryptography in order to make it accessible to others. The emphasis of this project is based upon the insecurity of the connection between the client and the server. By creating a system that can easily be expanded without the steep learning curve normally incurred, it allows a decentralised, secure system to be developed, essentially giving security a greater level of accessibility.

1.5.1 Academic contribution

The value of making this type of software accessible is to allow people who do not have access to a program with encryption to make their own. This is useful as companies like WhatsApp and Signal are targets for both hackers and governments [**Sparrow**, **2017**]. It is hoped by making an easily expandable system that can be built upon, this project will help create an open-source simple encryption and decryption process allowing others to build their own systems more easily. Understanding the requirements will be a key part of reaching the goal of creating a novel and valuable system.

1.6 Thesis Structure

This chapter has outlined a broad summary of the work while investigating the aims and key questions. These questions have been considered with a methodology which suits the requirements. Chapter 2 describes the background of the research in the area of Secure Messaging, particular attention is placed upon popular messengers as these are the focus of the research and provide relevant value. In Chapter 3 the system design is described and expanded upon in order to communicate the project methods used. In chapter 4 the implementation of the system is elaborated upon. Chapter 5 consists of description of the testing techniques and testing results. Finally, a conclusion and a direction of future work is given in Chapter 6.

2 Background

2.1 Introduction

Chapter one defined the investigation and set the scope of the project, Chapter two follows on in more detail and explores this field has approached this area. This section also details the considerations made by others when combining usability with security for optimisation of either aspect. User facing user facing applications are focused on particularity as they provide relevance.

2.2 Encryption

Encryption is the act of protecting a message to avoid it being read by an individual who is not the intended recipient. Encryption turns a plaintext message into ciphertext. This is the message after encryption but before decryption. There is a focus on three main areas, diffusion, confusion and completeness [**Shannon**, **1949**].

Encryption is necessary to secure the client, as the message leaves the client to reach the server, and can be intercepted and analysed. In an insecure server a intercepted message can be modified, read or destroyed. It is because of this encryption is necessary, hiding information from attackers or observers.

Diffusion - if a single bit of the ciphertext were to change then, statistically, half of the bits in the cipher would also change. The same would also be true of the ciphertext, where a one bit change would change half the plaintext. This creates an effect called permutation which would prevent differential analysis [**Shannon**, **1949**].

Confusion - each single bit of the message plaintext should rely on several parts of the key. If a single letter of the plaintext is changed, then the entire key should be different. This makes exhaustive searches much harder for an attacker [**Shannon**, **1949**].

Completeness - every bit of the ciphertext should depend upon every bit of the key. Therefore one change to the ciphertext should change a random aspect of the key. This prevents key differentiation and the comparisons of similar keys [**Shannon**, **1949**].

2.2.1 Symmetric encryption

AES Encryption

AES stands for Advanced Encryption Standard [**Daemen**, **2003**] and is a lightweight symmetric encryption algorithm. Symmetric cryptography enables the same cryptographic keys to be used by both the sender and recipient, the same keys are used to encrypt and decrypt. In practice, these keys are a shared secret between the two parties A and B. However, as both parties need to know these keys (they must also be shared securely) this is a major disadvantage of using symmetric encryption [**Daemen**, **2003**].

AES works by a network of substitution and permutation. This network is a series of linked mathematical operations, taking a block of plaintext from a message, alongside the symmetric key and applying alternating layers of subsitution and permutation to produce the ciphertext. This process transforms the input units of plaintext into ciphertext over a series of rounds, 10 rounds for 128-bit keys or 14 rounds for 256-bit keys [**Daemen**, **2003**].

Decryption includes reversing the above process using the appropriate key. Once again the key needs to be the same for encrypting as it is for decrypting.

AES was chosen for my project because it is fast, reliable and can be used on a variety of platforms. Symmetric key transmission can be implemented by producing keys using the Diffie-Hellman protocol [**Diffie**, **1976**]. However, AES can be vulnerable to timing attacks [**Bernstein**, **2005**]. However timing attacks require 200 million plaintext messages in order to accurately carry out. Therefore the risk is low enough to be used in this project.

A timing attack is the process of analyzing the time taken to execute cryptographic algorithms in order to leak information [**Bernstein**, **2005**].

2.2.2 Asymmetric encryption

RSA Encryption

RSA stands for Rivest, Shamir and Adleman, the individuals who created the security standard. It was first publicly described in 1977, however, Clifford Cocks, who worked for GCHQ described a similar system in 1973, but it was not disclosed to the public until 1997 [**Cocks**, **2008**].

RSA uses a system of prime numbers, a number n would need to be created by multiplying two prime numbers, n and n_2 together. Prime numbers are necessary because they create a strong one-way function. This is a function that can easily be computed, but is difficult to reverse when only the end result is known.

The totient of this number n is then found and is multiplied by a random integer + 1. Then select and encrypt and decrypt prime from the two factors of $(Totient \cdot n_a) + 1$.

Then to encrypt follow the below equation:

$$t_c = t_p \cdot p_e mod(P_1 \cdot P_2)$$

To decrypt follow the below equation:

$$t_p = t_c \cdot p_d mod(P_1 \cdot P_2)$$

RSA was not chosen despite its asymmetry and would solve the problem of key transmission, it is slower than AES. RSA is limited to one block, whereas AES can use block chaining. Block chaining is the process whereby each bit of plaintext is compared with the previous ciphertext block using the logic XOR, or exclusive or, before it is encrypted. In this way, the property of diffusion is enforced. RSA carries this out more slowly than AES; in a messaging application speed is a vital aspect of its function. Therefore AES was selected over RSA [**Firegun**, **2015**].

Diffie-Hellman Key Exchange

Diffie-Hellman is a key exchange protocol used to generate a common secret key between two parties over an insecure connection. From this secret a key can be made that is known to both parties and can be used for symmetric encryption. This is achieved via a one-way function using a prime number and the primitive root modulo of the prime number chosen. This creates an easy to compute, hard to reverse function that is required for making secure keys [**Diffie**, **1976**].

Diffie-Hellman conveniently solves the problem of symmetric key transmission through the use of common secrecy. Whereas more traditional forms of symmetric key transmission use couriers, diplomatic bags and other forms of transport. Diffie-Hellman enables two strangers who have never met to exchange data securely over the internet without ever needing to meet or know information about one another. However, this raises the problem of authentication. Authentication is the provision of authority to a user in order to make changes to a system [**Lamport**, **1981**].

Key authentication is solved by allowing a user to send a signature of their private key to be authenticated rather than a password. This helps prevent man-in-the-middle attacks because instead of a useful password, an attacker would only gain a useless one-time signature.

2.3 Server Security

The focus of the research is the creation of a secure client. To make a fully secure system security must be designed in every level. One layer of security could be circumvented, broken or removed and may not stop even an amateur attacker. It is preferred to incorporate several layers of security in order to stop determined opponents. Therefore if one layer fails there are multiple layers of defence that will protect the system and prevent unauthorised access. Because of this safeguards have been chosen to complement one another and build up a secure system.

2.3.1 Firewall

A firewall is a network security system used to monitor the input and output to and from a server. A firewall is used to create a barrier between a trusted internal network and wider internet. This makes the firewall the first line of defence between the server and external input.

There are two kinds of firewalls, a network firewall, which is used to defend multiple servers, networks and filter and monitor traffic. The second kind of firewall is a host based firewall. This provides security for a device and controls communications for one host. Due to the nature of the server, the host-based firewall will be the focus.

Host-based firewalls limit the number of ports available for access. This limits the number of services that can be attached to the device, which in turn lowers the potential attack surface. A server with fewer ports open and less services available will accept less connections and therefore will be more difficult to attack. This restriction will, as stated above, be the first line of defence against malicious attacks.

2.3.2 Public Key Infrastructure

Public Key Infrastructure, also known as PKI, is a system that can manage certificates within a server. This is useful for authenticating and verifying individuals who want to connect to the server. It is normally done alongside Secure Socket Layer (SSL) or Transport Layer Security (TLS), which makes a system more secure through the proper authentication of individuals. This will also help prevent man-in-the-middle attacks on clients using the server.

PKI also gives clients a strong Key Derivation Function (KDF) ivolving the creation of one or more secret keys from a master secret value. The secret key could be created in a variety of ways, but mainly a securely kept password, pseudo-random number or user generated random number. The KDF would then be used for hashing as an approach to verification or a component of a key agreement protocol.

2.3.3 Service auditing

Server auditing is an administrator task of checking the privilege of users currently on the server system. This ensures users logged into the system would not have rights or privileges that put the server security at risk. In order to effectively use the audit, its necessary for the administrator to understand the attack surface of the server; enabling knowledge of potential attacks, where they can occur and how they can be contained as well as knowing how risks can be mitigated and defences strengthened.

Service auditing will include the annual examination of practices, procedures and technical controls. This is achieved in order to keep updates current for software on the server to fix possible bugs. The field of cyber security is changing rapidly, therefore it is vital to examine what security controls are in place and how they could be regularly updated.

2.3.4 Intrusion Detection System

An Intrusion Detection Systems (IDS) monitors network traffic for malicious activity and checks files for unauthorised modification. Using an IDS would enhance the security of the server by allowing internal breaches to be detected.

IDS can take many forms, including anti-virus software or hierarchical systems. This is different from a firewall as firewalls look outward for malicious threats, but IDS looks inwards at potentially unwanted software and malware on the system.

2.4 Decentralised Messaging

2.4.1 Internet Relay Chat Servers

Messaging across the internet begun in Finland in 1998 with the launch of the Internet Relay Chat (IRC) [**Oikarinen**, **1988**]. The IRC is a application layer protocol and although it had begun in Finland it soon spread over the world. Its use has decreased and is now replaced by client facing systems such as WhatsApp, WeChat and other social media platforms that do not require technical knowledge and are catered towards the ease of user experience [**Sternberg**, **2011**].

IRC remains an important aspect of communication, not just historically, but because it has come from an era before third parties. It can be set up to create a secure service without giving up knowledge of the back-end infrastructure. It has been considered for the project because of this key feature.

The features of IRC include a chatroom called a channel. Channels can be given any name by the user who created them. In these channels users can post public messages or send private messages to specific users. Messages are sent to a server where they are publicly available to anyone who goes onto that channel [**Oikarinen**, **1988**].

In terms of the technical side of IRC, there are two main aspects, servers and clients. The servers are usually written in C, but they can be made in most programming languages. Clients are typically written in the same language and are made using a variety of programming languages, including Python, Java and C. The typical Operating System used for servers is Linux, though Windows and Mac OS can also be used for some server side software. Linux is generally chosen due to the high level of customisation that it allows.

The IRC can also use a identification protocol called Ident. However, this will not be used in this project because of a security flaw as usernames are accessible it can be used to gain a list of usernames from the server.

2.4.2 XMPP Servers

The Extensible Messaging and Presence Protocol (XMPP) is a communications protocol for messaging based on XML or the Extensible Markup Language. It enables instant messaging between two or more users via a server. XMPP was developed by the open source community for Jabber in 1999 and worked alongside IRC messaging, though the IRC continues to be used to the present day [**Pingdom**, **2012**].

In terms of the features available, XMPP allows users to send textual messages through a client to a server and from there to a client in a similar system to other messaging protocols. However, more recently XMPP has been replaced by services such as WhatsApp, Facebook messenger and other similar social media based messaging services which will be investigated later in Chapter 2.

The client side of XMPP can be made in a variety of languages to do a variety of tasks. Unlike IRC, XMPP clients can also work within a browser and android in addition to usual Operating Systems [**XMPP**, **2017**].

XMPP is important to this research because it is one possible option when creating the server for the messenger. It has the advantage of being decentralised meaning the XMPP server is isolated from third parties. Allowing for more secure connections to be made. However, a disadvantage of XMPP is it requires more back-end structure, alongside more third party distributions, a possible source of insecurity.

2.5 Centralised Messengers

To answer the research questions asked in Chapter 1 it is important to look at how others have tackled the same problems. This includes an analysis of hardware, software and user experiences in order to discover how these applications have been successful.

It is because these are centralised systems, if these companies were forced to turn off end-to-end encryption, the people who relied upon them would have few options for secure messaging. Although these applications make secure messaging accessible, they do not provide a decentralised framework.

2.5.1 WhatsApp

WhatsApp is one of the most popular messengers in the world, which recently reached over a billion users [**Statt**, **2016**]. Historically, an application that begun with almost no security for users, has built security into its software over time.

WhatsApp begun as a startup creating a messenger that would work over the on internet for a mobile platform as an alternative to SMS, thereby cutting costs to the user. In February of 2014 WhatsApp was bought by Facebook, the same year that Snowden revelations came to light [**Snowden**, **2014**]. Companies such as WhatsApp quickly discovered that security could become an easily exploitable selling point. In November 2014 end-to-end encryption was rolled out amongst their products [**WhatsApp**, **2016**].

In 2014 the Snowden revelations exposed misconduct amongst the NSA and GCHQ [**Snowden**, **2014**]. These agencies used malware and data collection to monitor the calls and messages of entire countries [**Snowden**, **2014**]. There is evidence that the NSA has broken through current HTTPS protocols [**Halderman**, **2015**], which uses public key cryptography similar to RSA, the encryption algorithm that was almost used in this project.

End-to-End encryption is the process of having information encrypted before it leaves a device and only decrypted on the device of the intended recipient. This includes the messages being encrypted as they pass through the server, which guarantees their secrecy.

WhatsApp achieves this is through two 30bit AES keys that join and make a single 60bit key between the sender and recipient [**WhatsApp**, **2016**].

From 2014 WhatsApp begun to work with Open Whisper Systems; which is the industry leader in the creation of secure messaging and is the creator of the Signal Protocol [**Moran**, **2011**]. Whisper systems is funded by a combination of grants and donations and all its products are free and open source [**Moran**, **2011**].

Despite this encryption, WhatsApp does not protect their messages metadata. Metadata are the activity logs of a message, they include data about the times, origins and recipients of messages. This is important because it allows an attacker to find valuable information about targets [**Opsahl**, **2013**]. Although the subject of the message is encrypted, the metadata is available to Facebook as the parent company [**Rastogi**, **2016**].

In terms of scalability, the advantage of WhatsApp has been Erlang, a programming language that allows a small number of engineers to cater to a large number of users [**OConnell**, **2014**]. Erlang was made in 1986 by a Swedish telecoms company called Ericsson. The language was specifically developed to support massive multi threading software that requires a high level of availability [**Wiger**, **2001**]. The language is developed in order to have order independent components. Meaning each component, or conversation in the software, is separate, allowing for high-level multitasking from separate threads; which is suited for developers who need to cater for a high number of users.

WhatsApp has a standard server configuration of 24 Logical CPUs and 100GB RAM, SSD. With this architecture, bottlenecks appeared around 425,000 messages. After a first round of fixes this increased to one million connections. Identification and resolution of similar bottlenecks allowed WhatsApp architecture to reach more than 2 million connections [**Rick**, **2012**]. An important consideration as social networking has the potential to expand quickly. Scaling was done in a secure, reliable manner that allowed WhatsApp to maintain its reliability and reputation matters when scaling [**Hoff**, **2014**].

To replicate something similar to WhatsApp would prove challenging. The challenges include creating tools, running tests and back-porting code, as well as encryption or design of the application. Overall the creation of a similar program to WhatsApp is would require a variety of specialist knowledge and skillsets.

2.5.2 WeChat

WeChat is a popular application based in China. In 2016 WeChat had 700 million active users, in recent years WeChat has been expanding into other Asian and Western countries [**Custer**, **2016**].

WeChat started as an application called Weixin and was invented by Xiaolong Zhang in 2011 [**Nian**, **2012**]. WeChat's growth has been spurred by the number of features attached to it. These features include contact-less payments, video calls and messaging increasing the value of finances. The app also has a "hookup" function that allows users to talk to nearby users of the app. Features like this have contributed to the fast growth experienced by WeChat.

WeChats growth has caused security concerns over the app. It has been demonstrated that data can be extracted using forensic tools [**Gao**, **2013**]. It has also been affected by malware multiple times, such as XCodeGhost malware [**Xiao**, **2015**]. While there have also been concerns about monitoring by the Chinese government [**Lien**, **2014**]. However these concerns have not reduced the popularity of the app [**Custer**, **2016**].

WeChat appears to use XCode. This is a development environment developed for MacOS that was first released in 2003 (WeChat itself was released in 2011). XCode appears to support most major programming languages, including C++, Java and Ruby. However, it does not appear to have the concurrency of threads that occur in WhatsApp [**Epalle**, **2015**]. Because WeChat is in China it is subject to the same censorship as the rest of the country [**Krebs**, **2015**] WeChat uses server architecture similar to WhatsApp, where Erlang is the server language.

2.5.3 Signal

Signal is an encrypted instant messenging platform that also allows voice calls. It is the successor to two programs, the first called RedPhone for voice calling and the second called TextSecure for instant messages. Both of these programs were developed in 2010 by Whisper Systems, the precursor to Open Whisper Systems [**Team**, **2012**]. All of these features are automatically encrypted [**Kolenkina**, **2017**]. In both programs keys used are generated and stored by the users rather than the server, placing security into the users responsibility. Signal also has a built in mechanism that verifies that man-in-the-middle attacks have not occurred and the application also allows users to compare key fingerprints or QR codes. It employs a mechanism to notify users of key changes [**Rottermanner**, **2015**].

In 2011 Whisper Systems was acquired by the social media company Twitter and the application RedPhone was shut down [**Greenberg**, **2011**]. TextSecure and RedPhone were released as free open source software in 2011 and 2012 respectively

[**Team**, **2012**].

After working at Twitter from 2011 to 2014, Marlinspike, the individual who made Whisper Systems begun Open Whisper Systems in January 2013. It was in February 2014 that Open Whisper Systems introduced the Signal Protocol which included end-to-end encryption as standard. Signal was the first IOS chat that enabled strong encryption for free for both messaging and voice calls [**Greenberg**, **2014**]. It was from here that Signal then spread to desktop, android and other clients [**Ermoshina**, **2017**].

A man in the middle attack is an attack where an attacker alters the messages between two parties directly communicating with each other. This is usually done through imitating a server in between two parties; resulting in messages passing through the attackers system on their way to the clients [**Tanmay**, **2013**].

Signals systems are written in a variety of languages, from Java for Android, and Objective C for IOS [**Moran**, **2011**]. The messages themselves pass through a server and use several kinds of encryption including Diffie-Hellman key Exchange, AES and the Double Ratchet Algorithm [**Frosch**, **2016**]. The Double Ratchet Algorithm is a key management protocol that was developed by the founder of Open Whisper Systems Moxie Marlinspike and Trevor Perrin in 2013 [**Marlinspike**, **2016**] which can be used as part of a cryptographic protocol to provide end-to-end encryption for a user. The advantage of the algorithm is it combines a ratchet based on Diffie-Hellman and a Key Derivation Function, hence the name double ratchet. After this initial exchange it manages an ongoing renewal of short lived session keys. This self-healing automatically stops an attacker from accessing clear text messages having compromised a session key [**Cohn-Gordon**, **2016**].

Although Signal is mostly based on the client-side in a decentralised system, there are servers involved. But unlike WhatsApp their metadata is protected using

Transport Layer Security or TLS, the successor to Secure Socket Layer, or SSL [**Marlinspike**, **2012**]. The metadata is deleted after 24 hours. Messages are kept no longer than it takes to send them to the user [**Whisper Systems**, **2017**]. Signal's servers are centralised and maintained by Open Whisper Systems and the server system is written in Java [**Marlinspike**, **2017**]. Although details of the server architecture is unknown, there is evidence of optimisation for speed and reliability for Signals user base [**Greenburg**, **2014**]. Another aspect of Signal that lowers the costs of server architecture, is the disappearing messages for memory. Not only is the metadata only kept for as long as it takes to send the message, but messages are not retained, this has the double advantage of less storage required and greater privacy for users [**Whisper Systems**, **2017**], [**Whisper Systems**, **2016**].

If Signal were to be replicated, it would be difficult for one person to replicate their level of encryption. The use of a double ratchet is effective but presents the problem of comprehension for users who want to create a simple easy to compute and hard to reverse function. In terms of the usability of the code available from Signal. Although the is framework robust it does not provide a simple framework that could be taken and remodelled [**Moran**, **2011**].

The relevance of this work to the research questions asked is what considerations have been made by past messaging protocols and how have they developed their software. The background research is also relevant for future work.

3 Design

3.1 Introduction

Chapter 2 explained the context of the different backgrounds researched. This chapter seeks to outline the design and implementation of previous research. This hopes to answer what kind of design and criteria is required for developing an open source and secure messenger and explore in greater depth the requirements and considerations in reading the research aim.

3.2 Server

3.2.1 Aims

The main aim of the servers creation is to support the client and cryptographic system. In order to make sure there are no weak points in the overall design some measures to protect the server will be taken. To provide a realistic model the server will also need to be globally accessible.

3.2.2 Considerations

The server will provide support to the client software and provide a base for users who have never met, to talk confidentially. The server will either use the daemon

IRCd-hybrid or ngIRCd. These server architectures are used to implement the IRC protocol. This enables conversations via users over the internet. Both IRC server software types are similar but IRCd-hybrid has been chosen as it is more widely used, is lightweight and perfect for small scale messaging.

The server will need to be globally accessible for the sake of realism, using public and private IP addresses, as well as editing the dynamic and static IP address. This will include creating a URL and port forwarding. These steps will make the server open to the internet and globally accessible.

In order to capitalise on the public nature of the server a direct connection to it will be set up. This will be through Secure Shell (SSH), allowing a user to operate network services securely over an unsecured connection. This will be necessary to enable it within the server. Port 22 will need to be open on the firewall to allow these direct connections, however, this will bring a number of security concerns.

To mitigate these security concerns, a number of features will be added to the server. These include configuring a host-based firewall to close open ports.It will be necessary to write and configure the firewall correctly and safely, and ensure its effectiveness. The opportunity will be taken to explore server-side architecture and similar models.

Fail2ban which will lock a user out if they fail to type the right password multiple times has been designed into the project, this will help prevent dictionary attacks against the server. A IDS called ClamAv will be set up to prevent clean up any viruses that may get onto the server. Furthermore, the server itself will be encrypted, preventing the data being accessed if the server is compromised. Service auditing will be guaranteed through cron, an inbuilt automatic updater for the raspberry pi operating system used on the server.

3.2.3 Justification of Design Decisions

Before the server could be designed, a comparison was required on how XMPP and IRC could be used. The number of users for both IRC and XMPP have varied over time. However XMPP has a greater number of users due to the success of applications such as MSN, Facebook Chat and WhatsApp, where the number of users exceeds 1 billion [**Statt**, **2016**]. In comparison, the IRC userbase has dropped since 2003.

The advantage of IRC is that it can be used for both one to one and many to many protocols which go through channels. It is a simpler protocol than XMPP called RFC 1459. This is a text based protocol with almost any socket based program being able to connect. However, a disadvantage is the lack of authentication, the only requirement to join is that a user presents a unique nickname to the server.

XMPP is a more general purpose mechanism. Clients need to connect to a local server in order to communicate to others on the same server. The client to server protocols have been standardised which makes it more secure. XMPP is also used for one to one chat but can be used in many to many chat in a multi-user chat room located on the server.

In terms of the architecture, XMPP depends on the number of users as to how powerful the server needs to be. IRC generally requires a smaller architecture in order to function correctly with a similar number of users. The cross platform availability of both XMPP and IRC is equal, both are available in one form or another on most Operating Systems with the same level of accessibility. Though the quality of service differs from software to software.

Overall both are equally suited to use as a messaging platform and despite some differences, because a lower level of hardware is required, IRC is lighter and simpler to use for both client and server, IRC was chosen. The choice between two services, IRCd-Hybrid and NgIRCd, proved difficult, but IRCd-Hybrid was chosen as it used less power and was a lighter system than NgIRCd. Both of these services are open source and available on Github [**Barton**, **2017**, **Miwob**, **2017**].

3.3 Client

3.3.1 Aims

The client will have three basic pieces of functionality. It will preserve the confidentiality of the messages, send those messages to another client and connect to the server.

3.3.2 Considerations

To connect to the server sockets are used that will require the port number for the server and either the address, either and IP or URL. Once the connection is secure a channel for the IRC protocol will need to be joined. The channel name will need to be written from the client to the server in order to function. Therefore a "write" method will be required for the sake of simplicity. This will allow the user to send messages to the server in the form of a 'command' and a 'message'. This will enable a number of features, including the ability to connect to other clients such as Hexchat or mIRC.

3.3.3 Justification of Design Decisions

A simple and minimalist design interface will allow user-friendly interaction and navigation. Since this is a proof of concept, only a simple console interface would be needed. Messages will be sent by allowing the user to type into the console when a prompt appears. The prompt should appear at the same time as information is being read from the server. It is possible to do this using either a timer or a boolean function to guarantee the concurrency of messages to and from the server.

If time is available then a clean and simple GUI should be included to allow the user more freedom and control in the application. A benefit of making the program easy to use and navigate is that the target demographic will be attracted to using the program. If the back end code and the front end design is easy to use and understand, then both developers and users will be attracted to try and use the system, thus making it more popular.

3.4 Encryption and Decryption

3.4.1 Aims

Keys will be exchanged using Diffie-Hellman and the plaintext will be encrypted and decrypted using AES. Key generation will need to be hidden from the user in order to gaurantee safety. In addition, the encryption will need to be relatively simple for developers to understand. Together they should help preserve the confidentiality of messages while they are between clients.

3.4.2 Considerations

Diffie-Hellman will be used in order to create the symmetric keys required for interaction between the two clients. These keys will be made on the client side for the sake of decentralisation, secrecy and convenience for the user. This removes the need for a public and private key infrastructure that would have been slower to encrypt messages. The prime numbers used for Diffie-Hellman would need to be strong in order to create secure keys [**Zimmermann**, **2015**].

The encryption itself will be made using AES. A 16 bit long initialisation vector (IV) will be made and converted to bytecode. The string will then be turned to bytecode and together with the key the IV and bytecoded message shall create the cipher.

The ciphertext and IV will be sent over the network to the second client to be decrypted. Decryption will then reverse the process. The received cipher will be stripped of the IV and key using the IV and key. Then the bytecode will be converted back into a string ready to be read by the second user.

3.4.3 Justification of Design Decisions

Encryption and decryption are a core part of this project. The key generation scheme used will be Diffie-Hellman, which was chosen for two reasons. Firstly, it is a secure form of symmetric key generation between two parties, although it could be scaled. Furthermore, it is a more elegant solution to creating a pair of symmetric keys than alternative methods, such as the server handing out user keys in a centralised manner.

AES has been chosen as the encryption and decryption standard for this project. This is because of the features mentioned in background research. But also because it is faster in both hardware and software and supports larger key sizes than another cipher such as 3DES.

In many secure systems, users are the weak point of the design. Therefore a user will need a low level of privilege, or control, over the user end of the program in order to stop them compromising themselves or others. In order to achieve this, key generation will be hidden from the users and inherited from another class. This makes key generation safer so, in a black box scenario, key generation is safe and users are protected.

3.5 General Design Considerations

Overall, the program should answer the research questions set out in Chapter 1. The program should have a high level of security, a simple code base and usability will be a priority.

In order to achieve this level of security it would be important to assign the lowest possible level of privilege to the user while they are using the whole program. There are many advantages to this, the first is that the user will find it difficult to break the program, thus making it more reliable. The second is that the user will only need to have a small amount of input when they run the program, this will make the program easier to use.

It is important to make the code base of the project easy to develop in future work. Meaning additions will be possible for anyone who would want to develop the code further. In order to achieve this the code will need to be cleanly made in relatively easy to understand blocks, as well as extensively commented with the information a developer might need to extend it, this will come with guides for expansion such as videos and documentation.

A program made as stated above should be able to explore whether any considerations that could be made when developing a secure and open source chat messenger. This, alongside an easy to develop and simple but secure code base should allow for a decentralised system that can be replicated without as much difficulty as is currently available, making it closer to the research aim.

3.6 The Diagrams

3.6.1 Block Diagram

The block diagram shows the passage of messages from client to server to client in a simple connection. In this example a message is written by Alice who then chooses the recipient, whether that is a private or public message. That message is then sent and the message is automatically encrypted using AES and Alice's secret key. That message is then sent to the server.

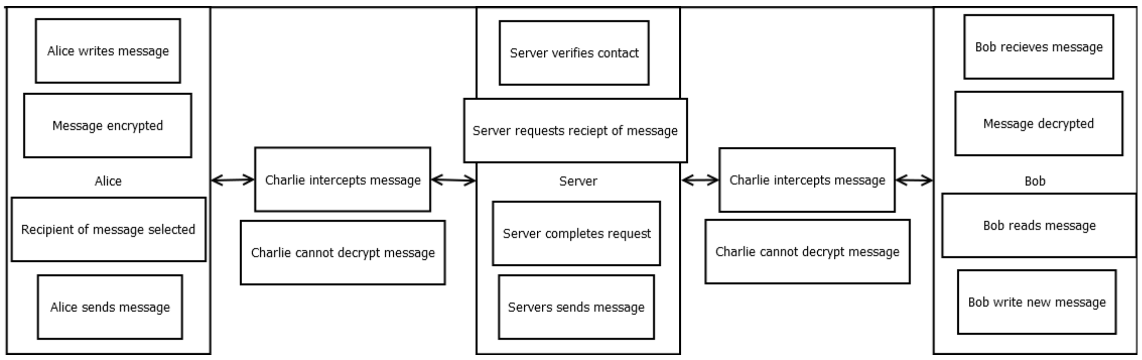


Figure 3.1: Sever-Client interaction

Inside the server the message is not decrypted but it is sent on to the recipient of the message, in this case Bob. Bob can then receive the message and is then able to decrypt and read it in secrecy using his key.

Figure 3.1 can show us the flow of conversation, as well as the functions of the users and the server. This helps us clarify the overall concepts through a concise description of the actions inside the server.

3.6.2 Flowcharts

Figure 3.3 is a flowchart of the Diffie-Hellman and AES processes. This illustrates the kind of processes that will be involved in the creation of encryption and decryption inside the program.

3.7 Documentation

There is value in designing documentation because it serves to fulfil a number of purposes.

1. It is important for understanding the code from a different point of view. This can involve looking at old code or collaboration from others.

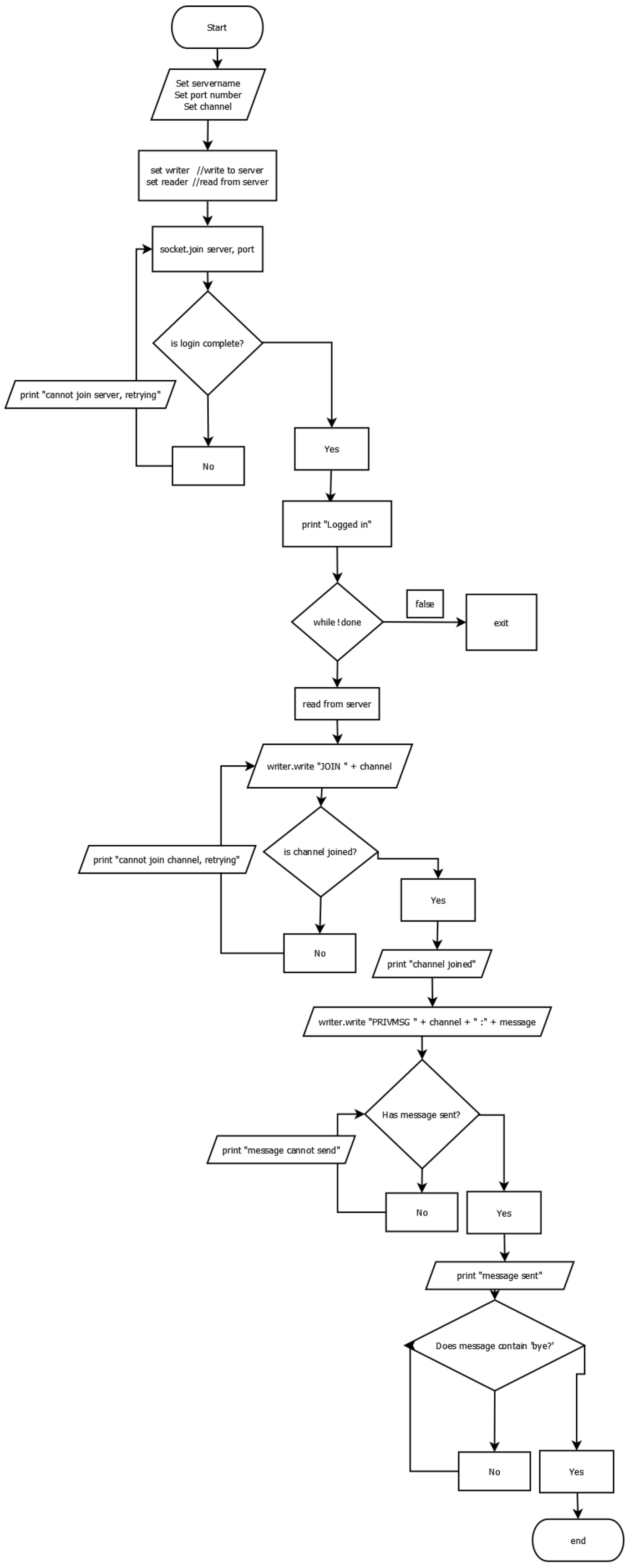

Start
Set servername
Set port number
Set channel
set writer //write to server
set reader //read from server
socket.join server, port
is login complete?
print "cannot join server, retrying"
No
Yes
print "Logged in"
while ! done
false
exit
read from server
writer.write "JOIN " + channel
is channel joined?
print "cannot join channel, retrying"
Yes
No
print "channel joined"
writer.write "PRIVMSG " + channel + " :" + message
Has message sent?
print "message cannot send"
No
Yes
print "message sent"
Does message contain 'bye?'
No
Yes
end

Figure 3.2: Client Flowchart

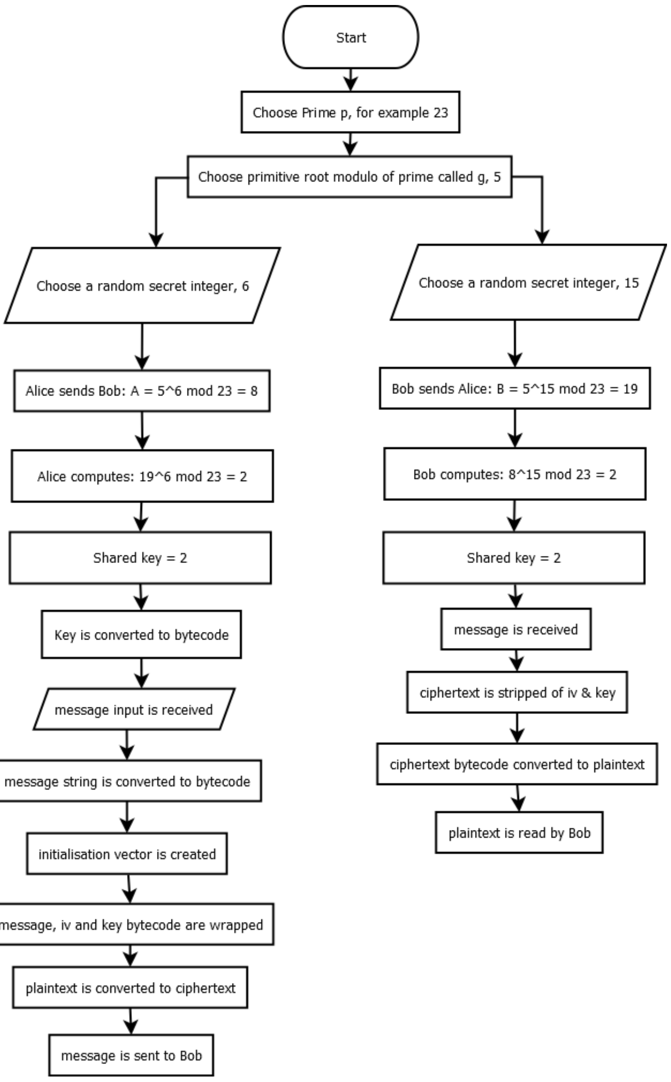

Start
Choose Prime p, for example 23
Choose primitive root modulo of prime called g, 5
Choose a random secret integer, 6
Choose a random secret integer, 15
Alice sends Bob: A = 5^6 mod 23 = 8
Bob sends Alice: B = 5^15 mod 23 = 19
Alice computes: 19^6 mod 23 = 2
Bob computes: 8^15 mod 23 = 2
Shared key = 2
Shared key = 2
Key is converted to bytecode
message is received
message input is received
ciphertext is stripped of iv & key
message string is converted to bytecode
ciphertext bytecode converted to plaintext
initialisation vector is created
plaintext is read by Bob
message, iv and key bytecode are wrapped
plaintext is converted to ciphertext
message is sent to Bob

Figure 3.3: Diffie-Hellman Flowchart

2. Major decisions can be documented and the reasoning behind them is understood

3. Priorities can be established in order to understand the costs and benefits of change

4. Can be used to help others understand the program when collaborating with others.

5. it can help explain the style the code was made in and can help avoid bugs.

4 Implementation

4.1 Introduction

Chapter 3 explained the criteria for design in order to explore the requirements needed when developing an open source and secure messenger. This chapter will explain the process by which this system was implemented.

4.2 Creating the Server

4.2.1 SSH and IP Addresses

It is important to select the best suited Operating System for this server. After several experiments with operating systems including Ubuntu, Jessie, Raspbian and Noobs, Noobs due to a number of factors:

- Quick to re-install server when broken
- Simplicity of use
- Ease of use

It was important to make the server accessible through SSH. This allows access without a monitor and keyboard and enables access to the server remotely. If a security breach occurred, it would allow ad-hoc repairs. SSH was turned on in the raspberry pi's configuration menu, openSSH was downloaded and updated.

The raspberry pi started with a dynamic private ip address that would repeatedly change. The ifconfig command gave the current ip address. The broadcast range, mask, gateway and destination were then required. When all this information was available the configuration file was edited through the command: 'sudo nano /etc/network/interfaces'. From this the line 'iface eth0 inet dhcp' was changed to 'iface eth0 inet static'. The broadcast range, mask, gateway and destination were then added, as well as the chosen ip address. Another ifconfig command finished off the process, initialising a static local ip address.

It was necessary to create a URL so the raspberry pi could be accessed globally, not just locally. It was necessary to forward the ports on the router to allow public access.

After some research into the area, noip.com was chosen. This gave a free dynamic DNS that points to a dynamic address. The only disadvantage of this is that it needs to be manually updated every 30 days to check that it is still in use. The URL chosen was called 'secmess.ddns.net' which is short for secure messenger.

The router was entered and using port forwarding in order to allow access from outside the local network. Ports 6665 to 6669 for the IRC server and port 5222 for the XMPP server as well as port 22 were forwarded in order to allow access to the server.

4.2.2 Server Security

Immediately after the raspberry pi was made publicly available a firewall needed to be configured. The default host-based firewall on the raspberry pi left all the ports open, giving the raspberry pi a large attack surface. Therefore the firewall had to be expanded to block nearly all connections. The ports that were kept open were 6665 to 6669 and ports 5222 for the IRC/XMPP server.

Below can be seen the original firewall rules:

```
Chain INPUT (policy ACCEPT)
target prot opt source destination
Chain FORWARD (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

Now below can be seen the new output from the firewall.

Allow all loopback (lo0) traffic and drop all traffic to 127/8 that doesn't use lo0

```
-A INPUT -i lo -j ACCEPT
-A INPUT -d 127.0.0.0/8 -j REJECT
```

Accept all established inbound connections

```
-A INPUT -m state –state ESTABLISHED,RELATED -j ACCEPT
```

Allow all outbound traffic - you can modify this to only allow certain traffic

```
-A OUTPUT -j ACCEPT
```

Allow SSH connections

```
-A INPUT -p tcp -m state –state NEW –dport 22 -j ACCEPT
```

Allow ping

-A INPUT -p icmp –icmp-type echo-request -j ACCEPT

Log iptables denied calls

-A INPUT -m limit –limit 5/min -j LOG –log-prefix "iptables denied: " –log-level 7

Drop all other inbound - default deny unless explicitly allowed policy

-A INPUT -j DROP

-A FORWARD -j DROP

The firewall is now more secure and blocks more ports. Once the firewall was created. The next step was to install fail2ban. This is a third party service that would lock the raspberry pi after a number of failed attempts to login. This would help mitigate dictionary attacks against the server.

Then ClamAv was installed. ClamAv is an open source IDS, a multi-threaded daemon which runs an anti-virus engine from a shared library. This IDS contains a number of utilities, including a command line scanner and advanced tools for database updating, in addition to built in support for nearly all popular file formats. After installation a scan was performed in order to check that the IDS worked correctly.

Finally, the server was encrypted, this was achieved using software called eCryptfs. It provides disk encryption for Linux and is a stacked filesystem level encryption layer. It has been used in the Ubuntu operating system to implement Ubuntu's encrypted home directory feature and is used in Google's ChromeOS [**Moog**, **2017**].

It was necessary to ensure the server allowed SSH and other necessary connections. These connections still functioned correctly and the process of securing the server was a success.

IRCd-Hybrid was then installed and powered the server. After the server was complete, it was necessary to make the client.

4.3 Creating the Client

4.3.1 AES

AES is an update of the old DES system of encryption. A block of plaintext is taken and encrypted using a key. However, a higher level description of the algorithm reveals a more complex process. AES encryption works through key expansions, described by Rijndael in his paper from September 1999 [**Daemen**, **2003**]. AES takes a 128 bit block of plaintext and creates a matrix of bytes, called a state using 16 bytes from that block. These states are combined with a equivalent part of the key using a bitwise Xor in the initial round. Following this there are several rounds that substitute, transposition and mix the bytes from both the key and the original plaintext. Substitution is done in a non-linear manner which replaces each byte with another, in transposition rows are cyclically shifted according to a number of rounds. Mixing involves combining four bytes from each column, this is until the final round where column mixing is no longer implemented.

To make an AES cipher an input string is required, then a key. There are several ways to develop a key, including making one from a string, the key generation chosen can be seen in the next section.

Padding is also a vital aspect of AES. Symmetric cryptographic functions process messages in blocks of a fixed length. Therefore nearly all hashing functions include a padding scheme. This helps prevent length extension attacks, which appends more information to the end of a message in order to reconstruct the internal state of the hash of a message [**ISO**, **2011**].

When the string to encrypt is ready and the key has been made the process of encryption can begin. The string and key both need to be converted to bytecode and an initialisation vector of 16 bytes must be created, preferably with a random number. The initialisation vector, string and key are then wrapped in a cipher which creates the ciphertext. The importance of the initialisation vector is for randomness, otherwise encrypting the same string with the same cipher would result in the same message. The two strings of the same message could then be analysed and the key could be worked out from those messages. The initialisation vector stops this analysis by randomising the ciphertext for every message.

Decryption works through a reversal of the above process. Firstly the AES cipher key needs to be stripped from the message, then the initialisation vector must be taken off. Finally the message needs to be converted from a byte array into a string, which can then be read by a user.

4.3.2 Diffie-Hellman

Implementing encryption and decryption cannot be done without a key. Generating a key through the Diffie-Hellman protocol is a secure way to share a symmetric key between two parties who have no prior knowledge of each other over an unsecured channel.

The process is explained in pseudocode in the Appendix, but more depth can be seen here. The effectiveness of Diffie-Hellman is the modulus operator which provides the common secret key.

$$a^b \bmod p = g^a b \bmod p = g^b a \bmod p = B^a \bmod p$$

In this example, a and b are the public keys for the two users, p is the prime and g is the primitive modulo of the prime. It is also important to note that all values aside from a, b and p are kept public, while those three vital values are kept secret. Once the secret values a and b are computed, it is possible to encrypt a message using the secret shared value, though in practice the shared secret key would be much larger [**Diffie**, **1976**].

Furthermore, a secure example would need a prime of at least 600 digits, such a large example would create the discrete logarithm problem. This problem has been described as a mathematical problem for which no efficient method for its computation is available. This problem is often used with cryptography because of the assumption that the discrete logarithm problem has no efficient solution. [**Shor**, **1995**].

The same can be said of:

$$g^a \bmod p$$

This is because that formulae is known as modular exponentiation, something that can be done efficiently with large numbers. However, despite the size of the numbers involved, g only needs to be a small number because of the modulo function.

Once the users have a shared secret key they can use it for encryption and decryption key. Although a prime of about 600 digits or above will be required in order to secure the symmetric keys in practice according to the discrete logarithm problem [**Daemen**, **2003**]. By contrast, the primitive modulo only needs to be a small integer because of the nature of the modulo function.

In practice Diffie-Hellman was made from scratch in order to understand the equation and processes required. By creating a prime and primitive modulo it was possible to make a SecretKey value from a Big Integer value. But afterwards it was decided that it would be safer to create the keys through a key generation function where it could not be manipulated by the user, making the program simpler to use and more secure.

4.3.3 Creating the Client

The next step was the creation of the client. First, a connection was made to the server using sockets, this required the server address and the port number. A channel was chosen and created on the server through a command and a string variable.

It was necessary to read messages from the server. A while loop cycled through the output stream attached to the server through the socket. A boolean was used to check whether the server had finished printing the message of the day before the scanner begun. A write method that contained a message string and command string in order to send messages to the server from an input scanner.

The write function contained two strings, command and message. An IRC protocol requires commands coupled with strings in order to function. The string is what the user is writing to the server. The full message is printed and the printer is flushed to make sure all messages are sent correctly.

4.3.4 Creating a Secure Client

A key generation class was first created. This class stores the public, private and agreed secret keys for both users. This class also contains the initialisation vector for the encryption, in addition to the key generator. In a commercial package, this class would not be seen by a user for the purpose of security. Although for an open source project this would be accessible for the purpose of transparency.

The second class is the Encryptor class. This class calls the first key and initialisation vector from the keyGeneration class. Encryptor also contains a number of methods, including connector, to connect to the server and join the channel. The client method allows a user to write to the server channel and read from the server, encryption is also called in this class, so messages from the client and encrypted.

The write method mentioned above allows messages to be written to the server and messages are decrypted locally so the user can see what they send, but these decrypted messages are not written to the server.

In the code itself, bytecode had to be sent over the server to be read by the decryption class, as sending strings resulted in data corruption and loss when they were converted back into bytecode for decryption. This was due to two factors, the first was 2 bytes equal 4 chars, which doubled the data being recited. The second factor was encrypted data being minus values, which are not allowed in UTF-8 byte format. These were automatically changed to the highest UTF-8 value which resulted in immediate data loss.

The third and final class is Decryptor. Once again this class calls the key and initialisation vector from keyGeneration. It contains the same connector and client that reads from the server. It also decrypts whenever a piece of ciphertext is found. This class contains the write method so that information can be written to the server. Decryption takes the place of encryption, the difference is that bytecode is read in as well as strings. If ciphertext is read excess bytes, in the form of the channel name and username of the sender, are stripped off before being decrypted as usual.

The program above hopes to answer what considerations should be made when developing a secure open source chat messenger. By the creation of a secure system there has been an attempt to create a prototype proof of this. The hardware that this program has been developed of remains lightweight and simple, in order to fulfill the requirements stated in Chapter 1.

5 Testing

5.1 Introduction

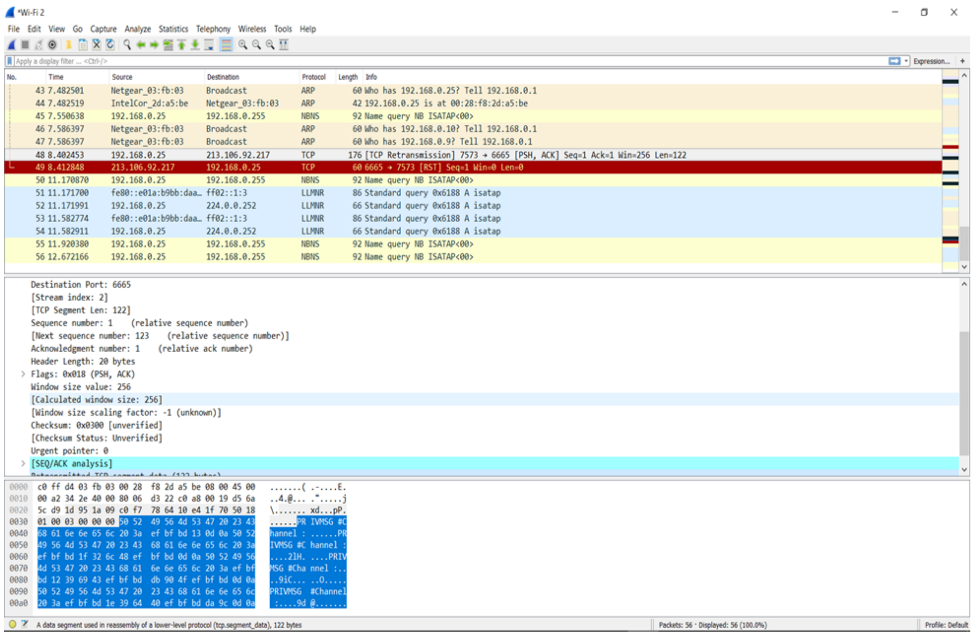
Chapter four defined the implementation of the project, this chapter follows on from the implementation with testing. This will expose how the program functions under various scenarios. Followed by a detailed user evaluation that asks about how the questions set out in Chapter one could be answered.

5.2 Project Testing

5.2.1 Server

The speed of the server was tested by sending unencrypted messages from a number of locations, these included the University building, my residence and a coffee shop with WiFi. In all cases the server functioned well and proved itself to be both speedy and reliable.

The server was investigated using the program Wireshark. An open source packet analyser that works in a similar way to a TCP dump. Packets of data are captured from a network and can be viewed through a GUI. The data that was


*Wi-Fi 2
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.  Time       Source                   Destination      Protocol  Length  Info
  43 7.482501   Netgear_03:fb:03         Broadcast        ARP         60 Who has 192.168.0.25? Tell 192.168.0.1
  44 7.482519   IntelCor_2d:a5:be        Netgear_03:fb:03 ARP         42 192.168.0.25 is at 00:28:f8:2d:a5:be
  45 7.550638   192.168.0.25             192.168.0.255    NBNS        92 Name query NB ISATAP<00>
  46 7.586397   Netgear_03:fb:03         Broadcast        ARP         60 Who has 192.168.0.10? Tell 192.168.0.1
  47 7.586397   Netgear_03:fb:03         Broadcast        ARP         60 Who has 192.168.0.9? Tell 192.168.0.1
  48 8.402453   192.168.0.25             213.106.92.217   TCP        176 [TCP Retransmission] 7573 → 6665 [PSH, ACK] Seq=1 Ack=1 Win=256 Len=122
  49 8.412848   213.106.92.217           192.168.0.25     TCP         60 6665 → 7573 [RST] Seq=1 Win=0 Len=0
  50 11.170870  192.168.0.25             192.168.0.255    NBNS        92 Name query NB ISATAP<00>
  51 11.171700  fe80::e01a:b9bb:daa…     ff02::1:3        LLMNR       86 Standard query 0x6188 A isatap
  52 11.171991  192.168.0.25             224.0.0.252      LLMNR       66 Standard query 0x6188 A isatap
  53 11.582774  fe80::e01a:b9bb:daa…     ff02::1:3        LLMNR       86 Standard query 0x6188 A isatap
  54 11.582911  192.168.0.25             224.0.0.252      LLMNR       66 Standard query 0x6188 A isatap
  55 11.920380  192.168.0.25             192.168.0.255    NBNS        92 Name query NB ISATAP<00>
  56 12.672166  192.168.0.25             192.168.0.255    NBNS        92 Name query NB ISATAP<00>

    Destination Port: 6665
    [Stream index: 2]
    [TCP Segment Len: 122]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 123    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header Length: 20 bytes
  > Flags: 0x018 (PSH, ACK)
    Window size value: 256
    [Calculated window size: 256]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x0300 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > [SEQ/ACK analysis]

0000  c0 ff d4 03 fb 03 00 28  f8 2d a5 be 08 00 45 00   .......( .-....E.
0010  00 a2 34 2e 40 00 80 06  d3 22 c0 a8 00 19 d5 6a   ..4.@... ."....j
0020  5c d9 1d 95 1a 09 c0 f7  78 64 10 e4 1f 70 50 18   \....... xd...pP.
0030  01 00 03 00 00 00 50 52  49 56 4d 53 47 20 23 43   ......PR IVMSG #C
0040  68 61 6e 6e 65 6c 20 3a  ef bf bd 13 0d 0a 50 52   hannel : ......PR
0050  49 56 4d 53 47 20 23 43  68 61 6e 6e 65 6c 20 3a   IVMSG #C hannel :
0060  ef bf bd 1f 32 6c 48 ef  bf bd 0d 0a 50 52 49 56   ....2lH. ....PRIV
0070  4d 53 47 20 23 43 68 61  6e 6e 65 6c 20 3a ef bf   MSG #Cha nnel :..
0080  bd 12 39 69 43 ef bf bd  db 90 4f ef bf bd 0d 0a   ..9iC... ..O.....
0090  50 52 49 56 4d 53 47 20  23 43 68 61 6e 6e 65 6c   PRIVMSG  #Channel
00a0  20 3a ef bf bd 1e 39 64  40 ef bf bd da 9c 0d 0a    :....9d @.......

A data segment used in reassembly of a lower-level protocol (tcp.segment_data), 122 bytes    Packets: 56 · Displayed: 56 (100.0%)    Profile: Default
```

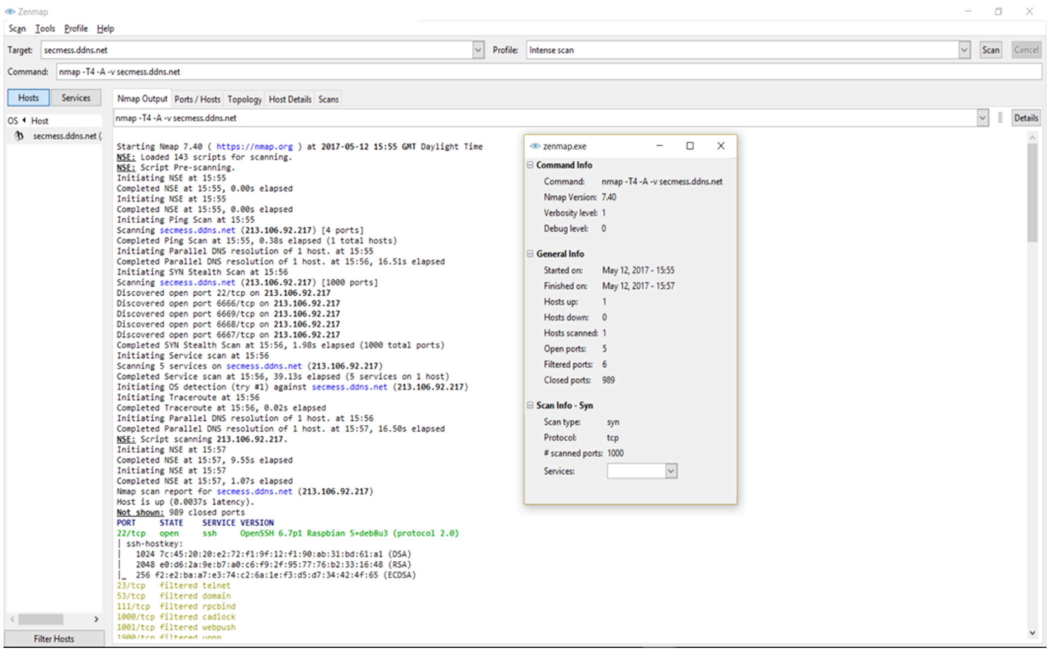
Figure 5.1: Captured message packet

captured on my home network using Wireshark can be seen in figure 5.1:

This shows us that despite a large amount of traffic, the messages are easy to find. It also interesting to note that although we can see that it is a message to the server and that there is a channel involved. We cannot see what the messages are.

After analysis of the packets, the Port scanning service Nmap, also known as network mapper was used to test the firewall. Nmap is used to discover open ports on a computer, as well as hosts and services on a computer network. Nmap sends special packets to the target and analyses the responses. This is used to audit the security of the server by showing what network connections could be made through the firewall.

The check against the server in Figure 5.2 showed me that the ports were closed. Aside from SSH, however, that has been mitigated by fail2ban, which will stop dictionary attacks against that port.



Target: secmess.ddns.net
Profile: Intense scan
Command: nmap -T4 -A -v secmess.ddns.net

```
nmap -T4 -A -v secmess.ddns.net

Starting Nmap 7.40 ( https://nmap.org ) at 2017-05-12 15:55 GMT Daylight Time
NSE: Loaded 143 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 15:55
Completed NSE at 15:55, 0.00s elapsed
Initiating NSE at 15:55
Completed NSE at 15:55, 0.00s elapsed
Initiating Ping Scan at 15:55
Scanning secmess.ddns.net (213.106.92.217) [4 ports]
Completed Ping Scan at 15:55, 0.38s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 15:55
Completed Parallel DNS resolution of 1 host. at 15:56, 16.51s elapsed
Initiating SYN Stealth Scan at 15:56
Scanning secmess.ddns.net (213.106.92.217) [1000 ports]
Discovered open port 22/tcp on 213.106.92.217
Discovered open port 6666/tcp on 213.106.92.217
Discovered open port 6669/tcp on 213.106.92.217
Discovered open port 6668/tcp on 213.106.92.217
Discovered open port 6667/tcp on 213.106.92.217
Completed SYN Stealth Scan at 15:56, 1.98s elapsed (1000 total ports)
Initiating Service scan at 15:56
Scanning 5 services on secmess.ddns.net (213.106.92.217)
Completed Service scan at 15:56, 39.13s elapsed (5 services on 1 host)
Initiating OS detection (try #1) against secmess.ddns.net (213.106.92.217)
Initiating Traceroute at 15:56
Completed Traceroute at 15:56, 0.02s elapsed
Initiating Parallel DNS resolution of 1 host. at 15:56
Completed Parallel DNS resolution of 1 host. at 15:57, 16.50s elapsed
NSE: Script scanning 213.106.92.217.
Initiating NSE at 15:57
Completed NSE at 15:57, 9.55s elapsed
Initiating NSE at 15:57
Completed NSE at 15:57, 1.07s elapsed
Nmap scan report for secmess.ddns.net (213.106.92.217)
Host is up (0.0037s latency).
Not shown: 989 closed ports
PORT     STATE    SERVICE VERSION
22/tcp   open     ssh     OpenSSH 6.7p1 Raspbian 5+deb8u3 (protocol 2.0)
| ssh-hostkey:
|   1024 7c:45:20:20:e2:72:f1:9f:12:f1:90:ab:31:bd:61:a1 (DSA)
|   2048 e0:d6:2a:9e:b7:a0:c6:f9:2f:95:77:76:b2:33:16:48 (RSA)
|_  256 f2:e2:ba:a7:e3:74:c2:6a:1e:f3:d5:d7:34:42:4f:65 (ECDSA)
23/tcp   filtered telnet
53/tcp   filtered domain
111/tcp  filtered rpcbind
1000/tcp filtered cadlock
1001/tcp filtered webpush
1900/tcp filtered upnp
```

zenmap.exe

Command Info

Command: nmap -T4 -A -v secmess.ddns.net
Nmap Version: 7.40
Verbosity level: 1
Debug level: 0

General Info

Started on: May 12, 2017 - 15:55
Finished on: May 12, 2017 - 15:57
Hosts up: 1
Hosts down: 0
Hosts scanned: 1
Open ports: 5
Filtered ports: 6
Closed ports: 989

Scan Info - Syn

Scan type: syn
Protocol: tcp
# scanned ports: 1000
Services:

Figure 5.2: Closed Ports

5.2.2 Client

Debugging Tests

The first task involved checking that the keys and Initialisation Vector were correctly generated, the keys themselves were generated through Diffie-Hellman key exchange, simulated in the key generation class. The Initialisation Vector was created manually in one case, for the purpose of testing, and made using SecureRandom in another case, to make sure that each encrypted message was different, even if they contained the same words. In order to test them the keys were used in a variety of experiments. These experiments included local encryption and decryption using asymmetric keys to verify that the program would not work, followed by comparing the generated keys to keys made using big integers. It was this way that verified that the keys were functional and worked correctly.

Then message encryption was verified. At first messages did not decrypt, but this was because the bytecode address was being encrypted rather than the actual string, this of course did not function correctly. To fix this problem the ciphertext was converted to new String in order to correctly update the message to ciphertext.

The messages were then checked for encryption using a different client, in this case the popular client Hexchat was used to connect and send messages. Plaintext messages could be sent from Hexchat to my client and displayed to the user. Encrypted messages were sent to the Hexchat client where they remained encrypted, proving that encryption worked without the client receiving messages.

Decryption. The first error encountered was Null pointer exception as the ciphertext was not correctly pointed at the server message output. This led to an input padding exception, an error normally found where the keys used to encrypt and inconsistant with the key used to decrypt. But previous tests had proved that the keys did match, negating the possibility of that error

There could have been three more causes for this error. Either the Initialisation Vector was being called incorrectly, or the keys were being called incorrectly. The third option was that data was being added in transit and the message length did not match the correct padding for decryption. Since the Initialisation Vector and keys were called correctly tests needed to be implemented to find any message corruption.

In light of this, tests were added to make sure decryption worked locally, then debug output was added to report the size of the data and the size of the packet being sent. Both of these showed that the data being sent was correct. An output of the packet being received was also made that checks could be made on the size of the data on exit and entry through the client. It was this last test that showed that excess data was being relieved through the server, and a string replacement could be made to cut this excess data off.

It was after this last test that the program appeared to function correctly in encrypting and decrypting messages. To ensure continued accurate function more tests were needed.

White-box Testing

Unit tests are a method where individual units of source code are tested together with their appropriate data types, to conclude whether or not they are fit to use. These whitebox tests were done on the three classes of the program and each tested the main units of functionality.

The Key Generation class was tested via the shared secret keys, key1 and key2. The Initialisation Vector was also tested for functionality. These put the expected values against actual results from the client and showed that the program worked effectively and that the keys and IV expected were correctly generated.

In the Encryptor class, it was the cipher-text and message that were tested for functionality, comparing actual results against expected results. Tests showed that the both values came out as a string and bytes.

The final test for the Decryptor class was for the serverMessage, ciphertext and messageChar variables. These put expected results against actual results and these tests showed that serverMessage was the expected String, ciphertext was in bytes as expected and messageChar was also in bytecode.

Black-box Testing

Key Generation Functionality:

Key generation functions with all expected input, the key was made securely and stored correctly in memory. The key's location was printed when the program was run and the test was a success.

The Initialisation Vector was also made securely and randomly and made each message individual, even if messages were the same. This helped secure the program against differential analysis.

Client Functionality:

The client functioned well with expected inputs, channel strings were made and connected to without failure, the server was also joined without difficulty. Messages could be sent to the server through the client with integers, floats and characters as well as strings which could all be sent to the server.

Encryption and Decryption:

Encryption functioned well, both locally and through the server with all expected input, including strings, integers and floats being encrypted and sent to the server without failure. These encrypted strings could then be sent to the server without any problems and remain encrypted correctly. Local decryption also worked well, any message input as mentioned previously could be decrypted without causing errors.

Tests were made using all expected inputs, including long and short strings, in addition to numbers and sentences. These were black-box tests that proved the functionality of the program.

Stress Tests

Two stress tests were performed on the server, to see how well it could cope with an excessive number of connections and an excessive number of messages.

The first test was to see how the server could cope with a large amount of input from a single connected client. The test showed that despite sending a 252 messages a minute, the client was able to read all messages. However, this did come at a predictable cost to speed as messages came through more slowly the more input the server and client had to deal with.

The second test showed how well the server reacted to a large number of connections from one computer. Over twenty connections were made from one IP address before that IP address was banned for a short time, stating that too many connections had been made over too short a period. The original aim was to have more than one hundred connections, but this proved impossible unless made from several computers at once.

5.3 User Testing

5.3.1 Questionnaire

A survey of 16 people from different backgrounds such as barrister and quantity surveryer was undertaken and they were shown the program, evaluated and given free reign over the program. First I gave them the program with no guidance, they were then given guidance if they did not understand how to make the program work. They were then asked to rate different aspects of the program out of 10.

5.3.2 User Feedback

The users described how they felt about each section of the program using number from one to ten, one being low and 10 being high. These sections included the accessibility of the code and the accuracy of the program in terms of the overall goal. Users came from a variety of backgrounds and age groups, although mostly from a demographic that would be more likely to use the messenger. The feedback itself shows us that the original considerations that were assumed in the development of a secure open source chat messenger. Some users were from low tech backgrounds and were not as comfortable using technology at all.

5.3.3 Data Description

First an average was taken to understand the overall confidence that was felt by the users. In addition to this, the standard deviation of the data was measured in order to understand how confident the users felt in their results. The low deviation between values suggests that the values are all close to the mean. This gives the impression of a high level of confidence in the confidentiality of the program.

The data also suggests that the there is a high level of confidence in the encryption and simplicity of the program, coupled with a high average for the accuracy to goal this illustrates a positive consensus of the program. This implies that the barrier to entry for cryptography for developers is still present, but it has been successfully lowered as developers are more confident in taking it up.

The data also indicates how the 'ease of use' was successfully balanced against the strength and reliability of the encryption. The high averages of the Secure and Encrypted field, together with the ease of use and visible to use fields, coupled with the low standard deviation points to a successfully integrated system.

In terms of the considerations that should be made when developing a secure and open source chart messenger, the data shows us that usability and confidentiality are very important. But it also shows us how speed is also valued as part of the system. Something that was only partially considered in design. The data insinuates that users were satisfied with the programs output. The question of the audience was answered by the comfort level of some of the users. The younger users found the program easier to use and more accessible than those who had less experience with computing. The developments that could be made to expand the project were also answered by the users, who suggested improvements such as encrypting metadata and adding a GUI.

6 Discussion and Future Work

6.1 Introduction

Chapter 5 explained some points of the testing and user evaluation that took place in order to satisfy the research questions. This chapter seeks to bring that forward in a discussion of how the program could be improved and used in future, followed by a conclusion of the work done.

6.2 Evaluation of Research Methods

6.2.1 Server, Client and Encryption

The creation and implementation of the server went well. The IRC server software was taken from Github and installed without failure. The server was made globally accessible and security features were added. The features added included the configuration of a host-based firewall, installation and scanning using an open source anti-virus, or IDS. SSH was configured correctly to allow ac-hoc repairs to be made, this coincided with a service called fail2ban, which stopped dictionary attacks by banning a user after multiple failed logins. Using eCryptfs parts of the servers filesystem was encrypted in order to help preserve the confidentiality of users. However, the demographic will not always be able to afford the time and financial costs to buy and create a server.

The client is functional, and the messages are relayed in good time. The user interface, although based in the console, works well to direct the user to the right action. However, the client could be improved with the addition of a cleaner and minimalist GUI interface. This would create an easier system for the user and would make the program more attractive to others, in addition to making it more accessible.

The encryption and decryption are strong. The key generation is kept hidden from the user and successfully generates keys in a secure way. Messages can be encrypted and decrypted locally, they can also be encrypted, sent over the server and decrypted by the client successfully. The encryption and decryption could be improved by using some of the concepts used in Signal, including the double ratchet and man-in-the-middle attack checker. But for a proof of concept, the program works well.

In relation to the original questions asked, the considerations when developing a secure and open source messenger have been examined and found that for a basic system, only a working server protocol, encryption standard and key generation are needed. Although there are improvements and customisations that can be made in order to improve the program, such as those mentioned in future work, later in this chapter.

The audience of the program is most certainly programmers and those worried about their security. While programs like WhatsApp and Signal still function to protect the privacy of users. But for those who do not have access to this kind of technology, the accessibility of cryptography and being able to build their own is invaluable.

The program itself has been built to run on light hardware, such as a raspberry pi or a phone. It is hoped that the ability to run the program on lighter hardware without straining the system will increase the usability and flexibility of the program.

6.2.2 User Evaluation

The user evaluation shows positive results because it documents how people interacted with the program.

However, the sample size is small, only 16 people from various age group were chosen. To make a truly efficient sample size there would need to be 100 or more people from a mix of backgrounds and age groups making data more representative.

Furthermore, the sample only allowed numerical values, which limited the scope of the feedback. This was problematic because it stopped the sample giving their full breadth of the experience. A potentially on instructed method may provide better insight.

There is also observation bias because the developer was present when the questionnaire was filled in. Some of those who filled in the questionnaire knew the author, leading to greater bias. In a real scenario the tester would use unsupported over the program and they would have no affiliation with the author. If this were the case then the sample would become a more accurate representation.

6.3 Overall Findings and Results

6.3.1 Server, Client and Encryption

The server, client and encryption have been tested both separately and as one system. It has been found that they function well in nearly all tests. However, more testing could be done, specifically attacks could be made against the encryption.

Since the project will be open source, bugs could be planted by other users and the integrity of the program could be compromised in order to try and reveal the keys used.

In relation to the original research question, the program has shown that considerations into security, speed and user-friendliness need to be made when creating a secure chat. However, considerations also need to be made of reliability and simplicity when it comes to the creation of secure messengers to help those who want to help customise and develop the program.

In terms of the hardware, the program could be made on light hardware, but it could also be used on more complex machines and could be customised to create stronger encryption or to encrypt more objects, such as images or documents.

6.3.2 User Evaluations

The requirements which have been developed based on user evaluation and investigation of the academic literature are as follows:

- Simple and minimalist
- Intuitive controls for users
- Graphical user interface for ease of use
- Instruction for new users

These requirements show the type of considerations which should be made for the user experience elements. This partially contributes to the sub-research question one, as the demographic and audience have been taken into account for the project design.

6.4 Future work and Development

There are many ways that the project can be expanded, improved and customised. This includes the addition of more features and the expansion of the program. This also includes future development and how it can be developed to allow others to use cryptography securely and safely.

If the program were made open source then there could a larger number of people developing the security, testing and contributing. This would make the program more secure.

Integrity checks could be added to the system. By taking the signatures of keys, or by creating a centralised Public Key Infrastructure that uses certificates. This way the users of an IRC can verify their identity and also authenticate their privileges. This would help maintain the integrity of the server and stop man in the middle attacks.

A lack of a user interface was a recurring issue in the user evaluations and despite the functionality of the console application, a more user friendly solution could be sought. The use of a GUI would allow those less experienced users to be able to easily navigate the program, thus improving the user experience though a simple and easy interface that should be intuitive to use.

Encryption could be extended to encompass the whole message as it leaves to go to the server. Including encrypting the messages metadata as it leaves the client through the server. This would involve the client decrypting two levels of encryption, which was done during one of the experiments created. However, this would be more suited to an asymmetric as opposed to a symmetric key pair for future work.

Thus far the availability of the server has not been guaranteed, except by banning a user that makes too many connections. In order to fully secure the availability of the server, it may be possible to create a backbone of servers that allow the transmission of messages from clients to other clients. However, there would be cost and time constraints in creating this kind of network backbone. Another implementation is allowing the user to send messages offline through an ad-hoc server system. Again cost constraints and securing ad-hoc communication is not focused in the security community.

The project could also be extended for mobile. IOS and Android are two of the most commonly used operating systems in the world and prevalence of secure messaging on mobile means it makes more sense to create end-to-end encryption on smart phones. However this would also create greater exposure and routing through mobile could be made to work through the Tor network to hide paths.

6.5 Conclusion

In addition a Gantt chart demonstrating the project management throughout the project duration has been added, it can be found in the Appendix. The process of researching, testing, coding, and writing changed as the project progressed. Deviations due to un-planned bugs and unexpected technical issues altered the original time frame.

Basic functionality is also important. A strong and stable encryption will be required in order to allow the program to properly function, and although the example created will just use a basic version of AES end-to-end encryption. It is hoped that later development and future work could add stronger encryption, integrity checks and a greater level of security. Another aspect is that in future encryption and decryption could be expanded into protecting images, files and other objects that appear on chats.

In terms of the original question, "What kind of considerations should be made when developing a secure open source chat messenger?"

To answer this question, different approaches have been considered throughout the research. Ultimately it has been found that the balance between the ease of use for the user, the speed of the program and the security of the encryption, and infrastructure and client were the main considerations. Together these have helped create a secure system. In order to truly reach this aim, integrity checks and stronger encryption will need to be added. Coupling these points with the creation of an open source project, should allow the goal of accessible cryptography to be reached.

In answer to the first sub-question, the audience and demographic of this program will certainly be the developers and specialist users. From the small research sample, this group demonstrated desire for the program to be easily customisable. This reinforces the value of good documentation and emphasis on easy collaboration.
The second sub-question: "What kind of hardware system would this program be developed for?" This question has been explored across a small range with the recognition that development could expand to mobile platforms. This software will function on a simpler hardware system but should be adapted for different operating systems.

The third and final sub-question is "What kind of developments can be made to expand this project?" There are a number of useful developments which can be made in order to expand the reach and functionality of the software. In order to reach this aim, the creation of an open source project will certainly be of benefit, with the help of others, this program can be extended and developed in order to help different users learn about cryptography for implementation.

6.6 Reflection

The research questions have been explored, however there is more that can be done to create a fully comprehensive set of requirements before being implemented. In reaching the aim to create a secure messenger which will make cryptography more accessible, the research has advanced closer to reaching this goal. It is hoped that the project can be expanded further, adding more features and making the program and server more secure. What has been done has worked well, resulting in an increased level of confidence and enthusiasm in furthering knowledge in this field.

6.6.1 New skills learnt

The most valuable skills which has been developed in the duration of this project, aside from the hardware knowledge and software development, are the debugging skills. Creating a clear understanding of what the software should do to compare to the actual behaviour of the code shows how to identifying bugs. Finding out how to make code modular, with various styles of testing is essential in finding and resolving issues. This can include checking the data types, the use of memory, and connections between modules to understand to how improve code. The value of documentation has also been evident throughout this work in order to ensure that code is clear and understandable for later development. It is also understood how important this is for good code practice, especially in open source work.

Another useful skill which has been developed is from applying theory to practical work. Different types of approaches and techniques have been necessary to place abstract theory in context. Secondarily to this is the importance of time management in project planning, as this has proved instrumental to ensuring the project has developed as planned.

Appendices

A
Review of Security Algorithms

A.1 Final Project Code: encryption, decryption and key generation

A.1.1 Key Generation Class:

A.1.2 Encryptor Class:

A.1.3 Decryptor Class:

//see also attached zip files for project code

A.2 Appendix Code and Experiments

//see attached zip files for experiments and appendix code //No reliance on libraries for any code

IDE used:

IntelliJ IDEA 2016.3.5 Build IU-163.13906.18, built on March 6, 2017 Licensed to matthew Sellman Subscription is active until October 1, 2017 For educational use only. JRE: $1.8.0_112-release-408-b6x86JVM : OpenJDKServerVMbyJetBrainss.r.o$

A.3 Images and Documentation

A.3.1 Pseudo-code

Client:

```
* set server name, port number, channel
* initialise scanner and reader streams
* connect to server through socket
* if login complete then
* print 'logged in'
* choose nickname
* choose username
* write nickname to server
* write username to server
* join channel
* print 'channel joined'
* while in.hasNext()
* read from server
* write messages to server
* read messages from server
* if message sent
* if no print 'message cannot send'
* if yes print 'message sent'
* end
```

AES Cryptography and Diffie-Hellman:

Diffie-Hellman Key Generation: -

```
* prime chosen (or p) = 23
* primitive root modulo of prime (or g) = 5
```

* Alice chooses a secret integer a = 6, then sends bob A = ga mod p which equals 8

* bob chooses a secret integer a = 15, then sends bob A = ga mod p which equals 15

* Alice computes s = ab mod p which = 2

* bob computes s = ab mod p which = 2

* both parties now share a secret: 2

* key1 is sent to Alice

* key2 is sent to Bob

AES Encryption and Decryption: -

* Alice decides to send a message to Bob

* Alice's key is converted to bytecode

* message string input is received

* message string is converted into bytecode

* initialisation vector is created

* message, iv and key bytecode are wrapped

* message is encrypted to ciphertext

* ciphertext is sent

* ciphertext is received

* ciphertext is stripped of iv and key to create plaintext

* plaintext is converted from bytecode

* plaintext is read by Bob

A.4 Further Reading

raspberry pi schmatic: `<https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/RPI-3B-V1_2-SCHEMATIC-REDUCED.pdf>`

`<https://www.eff.org/node/82654>`

`<http://stackoverflow.com/questions/19640703/what-is-the-technology-behind-we`

<http://repository.bilkent.edu.tr/bitstream/handle/11693/29073/10100587.
pdf?sequence=1>

/bibliostyle(plain) /