Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

An Anonymous Decentralised Messaging Application Utilising the Whisper Protocol

Seán Durban
B. A. (Mod.) Computer Science
Final Year Project May 2018
Supervisor: Dr. Donal O'Mahony

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration of Authorship

I, Seán Durban, hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university .

Signed: _____

Date: _____

Acknowledgements

I'd like to take the opportunity to express my sincere appreciation to the following people for their help and support:

- My project supervisor, Dr. Donal O'Mahony, for his persistent interest, guidance and enthusiastic support throughout the project.
- My fellow classmates in Computer Science, for their continuous help, generosity and humour throughout our four enriching years together.
- My family for their unwavering encouragement and support for all pursuits in life.

An Anonymous Decentralised Messaging Application Utilising the Whisper Protocol

Seán Durban
B. A. (Mod.) Computer Science
Supervisor: Dr. Donal O'Mahony
May 2018

Abstract

With the current extent of surveillance it's difficult to guarantee any form of digital communication will remain private. Furthermore, current messaging applications may offer end-to-end encryption, which provides assurances to the message content being secure, but fail to preserve a user's anonymity. They leak compromising information about the sender, recipients and the message itself from metadata. Centralised applications require a certain level of trust since servers handle the messages; this can present risks for both service and user. Ultimately a potential user should not have to choose a messaging application based on the application or company they trust the most and should have the ability to remain anonymous.

This project will propose a new form of messaging application which supports end-to-end encryption for both direct and group messaging by default. It will be decentralised and transparent, allowing for trust to be distributed in the system. It will also aim to maintain a user's anonymity by being a dark system, where no compromising information is leaked from metadata. A messaging protocol called Whisper, from the Ethereum eco-system, is used to achieve this. Whisper has a unique approach to decentralised messaging and routing which has clear influences from other technologies such as Tor.

The application delivered is functional and provides a unique form of messaging. However, testing of the application was limited due to underlying issues with the Whisper network. As a result, it's still unknown if it is a viable solution for mass usage or if anonymity can be truly guaranteed.

Contents

List of Figures

Chapter 1

Introduction

This introduction chapter will seek to describe the motivations behind the project, issues with current messaging systems and then arrive at a clear problem statement which this project will aim to address.

The overall aim of this project is to design and develop a messaging application which will run on a decentralised peer-to-peer network. It will support end-to-end encryption for both direct and group messaging and importantly it will maintain anonymity.

1.1 Motivation

This section will detail the core motivations behind the project.

Web3 [1] is a movement to build a new decentralised web and an internet where users are in control of their own data and identity. A group called the *Web3 Foundation* was set up in 2017 to guide and build out the Web3 ecosystem. They aim to aid and promote technologies and applications in the decentralised web space and are particularly interested in cryptography and security. In my opinion a major pillar of Web3 will be the ability to communicate securely and anonymously.

The extent of global surveillance has been a concerning development in recent years. The *Snowden-NSA leaks* [2] of 2013 suggested that multiple government agencies were working collectively to monitor people. Major resources are now being spent to monitor our all our activity both online and offline and to break secure systems. One high profile case involved Angela Merkel, the Chancellor of Germany, who's personal phone was reportedly tapped.[3] If the head of a major global power can have their communication compromised without their knowledge then it's safe to assume that they can compromise

any target. The issue here goes beyond attempting to conceal one's activity whether legal or not, it is that the user's trust is broken and there's potential for misuse of very powerful tools. Their communication which they assume to be private could be monitored either passively (bulk collection) or actively.

Current messaging applications tend to focus more on usability and efficiency rather than security. Resources are spent on user features which can differentiate the platform rather than building on the security which the user will never see. This is understandable since many of these are owned by private companies seeking to maximise profits and market share. However this does not make it acceptable. Users in my opinion should have their data protected by default. Unfortunately the opposite is the case and users' data and their lack of awareness is being exploited.

This was the case in the recent *Facebook-Cambridge Analytica data scandal* [4] where over 87 million Facebook user's data was shared to Cambridge Analytica, a private company. The users consented to have their own data collected which also allowed them to collect data on their friends accounts (on Facebook) too. This was unethical and a clear case where the user lost control over their data.

With the improvements in security, encryption and general increase to overall traffic over networks, different attacks are necessary. Traffic analysis is an example of such an attack, this is the method of examining intercepted messages to gather information from patterns in communication. Compromising information can be leaked from the message's size, timing and metadata. Traffic analysis has been used to attack networks which aim to preserve anonymity, such as Tor. Effective methods [5] have been found which greatly reduce the anonymity provided by Tor. Defeating traffic analysis has proved difficult, one solution is *masking* the communication channel, this the act of sending dummy traffic in order to keep the bandwidth constant. Masking is clearly a very inefficient solution and therefore not viable for general communication systems.

1.2 Current Messaging Applications

This section will present a more in-depth analysis on specific issues and limitations with current messaging applications available to users.

Applications exist which do not even offer basic encryption by default. For instance Facebook's messenger platform has over 1 billion active users but their default communication is completely unencrypted. This means that Facebook or anyone with access to their servers could theoretically read all of the messages. With the recent negligence

Facebook has shown, this is concerning and it remains to be seen how committed they really are to user privacy.

The majority of messaging applications work through a centralised server. The messages are sent from the sender's client to the server which relays the message onto the intended recipients. The server is effectively free to do as it wishes with the message once it's there, they can save, ignore or even attempt to decrypt it (in encrypted messaging). It can do this without the senders or recipients knowledge. This means you must be willing to trust the server with all the messages you send.

Centralised applications can also present risks for the company too. They handle and store sensitive data which highlights them as a point of interest for hackers but also government agencies. There has been many cases in the past where messaging application have been ordered by government agencies to release messages and decryption keys. Some applications have even been approached to introduce a backdoor, a method of introducing a process to bypass encryption or authentication in the source code. This could be an issue since large corporations own a large market share of messaging applications and theyre generally concerned with not damaging their relationship with governments.

An example of this risk is what has happened to the messaging application Telegram in Russia.[6] Telegram was banned in Russia due to its refusal to grant the Federal Security Service access to encryption keys needed to view messages required by the anti-terrorism law. In reaction Telegram started IP hopping, which means when one IP address got taken down or blocked they simply moved to another. This allowed users in Russia continued access telegram. At the timing of writing, this is still ongoing and the Russian regulators have now blocked over 19 million IPs causing huge disruption to other services including Spotify, Amazon and Mastercard. [7]

Lack of transparency is also an issue with some applications as source code or documentation of the underlying protocol is not made public. This means we do not know exactly how it works or how secure it is. This again damages trust of the user if its later found the application does not deliver the level of security expected. Open-source applications benefit from the added input of users, researchers and professionals too, where potential problems can be highlighted and quickly addressed. Also independent reviews of the application and it's security can increase overall trust. Signal [8], is an example of a messaging application which has benefited from being open source. It has been praised by experts in the field and at the time of writing it's recommended by the Electronic Frontier Foundation for secure communication. [9]

1.3 Problem Statement

With the current extent of surveillance it's difficult to guarantee any form of digital communication will remain truly private. Furthermore current messaging applications may offer end-to-end encryption, which provides assurances to the message content being secure, but ultimately fail to preserve a user's anonymity. They leak compromising information about the sender, recipients and the message itself from metadata. Centralised applications require a certain level of trust since servers handle the messages, this can present risks for both service and user. Some also lack transparency as source code or documentation is not made public. The lack of peer-review adds another level of trust in the application, that it delivers the level of security and privacy expected. Overall a potential user should not have to choose a messaging application based on the application or company they trust the most and should have the ability to remain anonymous if they wish.

This project will aim to address these problems by proposing a new form of messaging application which supports end-to-end encryption for both direct and group messaging by default. It will be decentralised and transparent, allowing for trust to be distributed in the system. It will also aim to maintain a user's anonymity by being a dark system, where no compromising information is leaked from metadata. A messaging protocol called Whisper will be used to achieve this. Whisper has a unique approach to decentralised messaging and routing which has clear influences from other technologies such as Tor. The priority of the application will be security and anonymity which will come at the cost of efficiency.

Chapter 2

State Of The Art

This chapter will examine the technologies and ideas which underlie the application developed. It will start with some basic principals in Hashing and Cryptography, then consider current Peer-to-Peer and secure communication systems. Finally it will review the Blockchain, Ethereum and provide a detailed analysis of the messaging protocol used in the application, Whisper.

2.1 Prerequisites

This first section will explain some core algorithms and systems which are widely adopted across many different applications and are fundamental to the internet we have today.

2.1.1 Cryptographic Hashing

A cryptographic hashing function is a mathematical algorithm which converts data of any form and arbitrary size to a special data structure, a bit string of a fixed size. They're designed to be deterministic, such that it always produces the same output for an identical input. It's also a one-way function, meaning it is infeasible to reverse the hashing operation.

Many standards for hashing exist and depend upon their usage. SHA is a group of cryptographic functions published by the National Institute of Standards and Technology (NIST). In recent years there has been an effort to move all usages of SHA-1 to SHA-2 due to SHA-1's increased vulnerability to collisions. [10]

2.1.2 Proof of Work

A proof of work (PoW) protocol requires some work to be completed from the service requester. This work must be adequately difficult to complete, generally requiring time and/or computer resources. It must also be easy for others to verify the work has been completed and satisfies certain requirements. PoW has been implemented in many systems, generally to discourage and make spam and spam related attacks (such as a Denial-of-Service attack) more difficult.

The term *proof of work* [11] was formalised in a paper by Jakobsson and Juels in 1999. They described PoW as a *prover* demonstrating to a *verifier* that they have performed a certain amount of computational work in a specified interval of time.

Figure 2.1 shows an example of a general PoW algorithm. Some hashed data and a nonce (an arbitrary number) are hashed together and for the output to be valid it must have a specified number of leading 0's, known as the difficulty. If the output is not valid then the nonce is iteratively changed and the algorithm is repeated until the required difficulty is achieved. The nonce found to produce the valid output can be used to easily verify the difficulty was achieved and therefore can be used as proof that the work must have been completed. The resources (computational and time) needed to find a valid output with the required difficulty will vary for each input of hashed data, however the higher the difficulty the more resources it will consume on average.

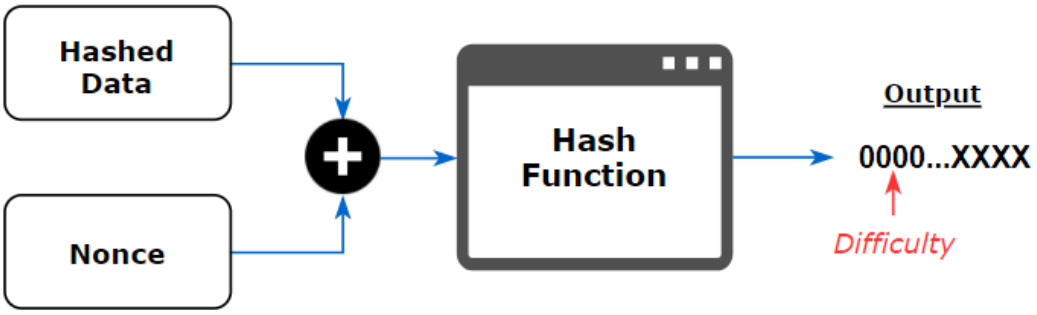

FIGURE 2.1: Proof of Work Diagram

There's no trivial way to figure out a correct nonce value, they can be randomly generated or taken from a sequence such as the ascending natural numbers. The work done to find that nonce is generally useless and the resources are spent purely to provide a form of trust in a system. Figure 2.2 shows an example of a PoW calculation. Suppose we want to preform PoW on the message *"Hello, world!"* and the difficulty required is at least 3 leading zeros. If we start the nonce at 0 and simply increment it every iteration it would take 4,251 iterations to find a nonce which achieves that difficulty. The nonce found is **4250**.

```
"Hello, world!0" => 1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64
"Hello, world!1" => e9afc424b79e4f6ab42d99c81156d3a17228d6e1eef4139be78e948a9332a7d8
"Hello, world!2" => ae37343a357a8297591625e7134cbea22f5928be8ca2a32aa475cf05fd4266b7
...
"Hello, world!4248" => 6e110d98b388e77e9c6f042ac6b497cec46660deef75a55ebc7cfdf65cc0b965
"Hello, world!4249" => c004190b822f1669cac8dc37e761cb73652e7832fb814565702245cf26ebb9e6
"Hello, world!4250" => 0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9
```

FIGURE 2.2: Proof of Work Calculation [12]

2.1.3 Asymmetric Cryptography

Asymmetric cryptography or public-key cryptography is an encryption scheme that utilises a pair of keys. One key in the pair can be shared, called the *public key*. The other key is meant to be remain confidential, called the *private key*. The keys themselves are essentially large numbers stored in byte format. They're mathematically related but not identical meaning either key can be used to encrypt data and only the other key can successfully decrypt that data. Generally the public key is used for encryption and the private key used for decryption. An important feature of these keys is that it is computationally infeasible to compute a private key based on a public key. The RSA algorithm [13] named after Rivest, Shamir and Adleman, is a widely adopted public key cryptosystem and was published in 1978. They presented a system which security relied upon the difficulty of factoring the product of two large numbers, i.e the size of the keys generated. Since its inception we have seen enormous advancements in computational capabilities. For example in their original paper they recommended setting a variable n to be 200 digits long (663 bits), but this was cracked in 2005 by a team as part of the RSA Factoring Challenge.[14] But these advancements along with optimisations to key generation means we can create larger keys in less time too. For instance the National Institute of Standards and Technology (NIST) now recommend a key length of 2048 bits for general usage.[15]

Another fundamental property of asymmetric cryptography is the ability to sign data, which introduced the notion of a *digital signature*. This uses the previously described key-pair to verify the authenticity and integrity of transmitted data. The original sender combines a message with their private key to create the digital signature. The publicly known public key can be used to verify whether the signature is valid. If validated, the recipient is aware that the sender must have access to the private key and that it has not been tampered with.

2.1.4 Elliptical Curve Cryptography

Elliptical Curve Cryptography (ECC), first suggested in 1985, is an optimised approach to asymmetric cryptography. It utilises the structure of elliptical curves over the complexity of factoring large prime numbers. The optimisation being that EEC requires smaller keys compared to non-ECC cryptography while maintaining equivalent security. This in turn reduces storage costs and transmission speeds of providing such an encryption scheme. The same recommended key length of 2048 bits for RSA has equivalent security to an ECC key length of 224 bits.[15] Due to these advantages ECC has been readily adopted in the industry and there exists many different standards for a variety of different use cases. However ECC has been susceptible to issues in the past as a result of it's reliance upon a secure source of random numbers and incorrect algorithms which were later found to have flaws.[16] Also the plausible disruption to asymmetric cryptography due future developments of quantum computing is still unknown.

2.2 Peer-to-Peer

This section will explore different Peer-to-Peer networks and the systems which use them. As the application aims to be decentralised it will have to utilise a Peer-to-Peer network.

A Peer-to-Peer (P2P) network is a distributed computer architecture that allows for direct communication and transfer of data between individual nodes (known as peers). All of these peers are self-organised into a defined network topology with the intention to share resources. The exact implementation depends on the application and size of the network but generally a peer communicates directly with their known neighbours (other connected peers) who in turn, communicate with their neighbours. In this way the information propagates through the network. This differs from a traditional client-server approach since a peer can act as both a client and a server, consuming services from other peers while also providing services simultaneously. A pure P2P network has no centralised authority, but in practice many applications are in fact hybrids as they have P2P sharing with a centralised organiser.

An example of a hybrid P2P is *BitTorrent*, a protocol for P2P file sharing. It is implemented by a number of popular clients such as μTorrent and Xunlei. Older Bittorrent implementations introduced *trackers*, these are centralised, readily available peers with the purpose of introducing peers to one another. When a new node initially joins the network they first connect to one of the trackers, that tracker is then able to introduce them to other peers on the network. Once the new node has established a connection

to a number of peers in the network then they can disconnect from the tracker. There also exists *trackerless* clients such as Vuze which uses Distributed Hash Tables as a replacement to maintain decentralisation.

2.2.1 Distributed Hash Table

A Distributed Hash Table (DHT) is a decentralised distributed system of hash tables. A hash table is a fundamental data structure which allows for the efficient retrieval of data. Data is stored in an entry as a *Key, Value* pair, where the *Key* is used to find the respective *Value*. It utilises an underlying hash function to convert the key to a hash value, which is used as an index into the hash table. A DHT is used to distribute an entire hash table over a network of nodes, this introduces fault tolerance and the ability to scale. The nodes are organised in a defined way which ensures specific nodes are responsible for services based on their unique node ID. A mapping function is used to assign entries to nodes and a lookup function returns the network address of the node that stores the requested data, both are based on a hash function value. A simple DHT example is shown in Figure 2.3 with a ring topology of nodes. Here a hash table with 16 entries (Hashed from 0 - 15) is distributed across the nodes as denoted by the keys they maintain the value for. For example consider node 9, which is looking for some data. Initially the lookup function is used to find the network address of the node which is responsible for that data using the associated data key. Once it has the network address or node ID it can query directly for the data required. This assumes a DHT where the mapping function assigns entries to nodes and all nodes are aware of that mapping which can be used for simple lookup.

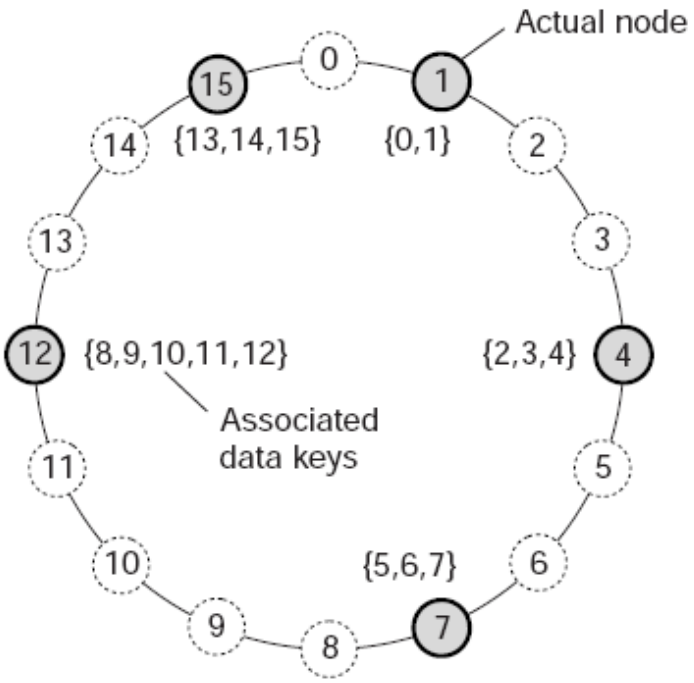

FIGURE 2.3: Simple DHT Example [17]

2.2.2 Kademlia

Kademlia [18] is a popular DHT implementation which uses a unique XOR (Exclusive OR) metric to build the network topology. Each Kademlia node is assigned a unique node ID and the distance between nodes is calculated by the XOR of their node IDs rather than actual physical proximity etc. One major feature is configuration data of the network spreads automatically from queries. Nodes are assigned a unique 160-bit node ID and are considered as leaves of a binary tree, where each node's position is determined by the shortest unique prefix of its ID. For a given node, the binary tree is split into subtrees which do not contain that node. It's ensured that every node is aware of at least one node in every non-empty subtree, this guarantees that any two nodes can locate one another by an ID. Figure 2.4 shows the position of a node (in red) with a unique prefix **0011** in an example Kademlia binary tree. The subtrees for this node are circled, they're all nodes with prefixes 1, 01, 00 and 0010 respectively. In real-world usage a Kademlia binary tree would be significantly larger.

The lookup algorithm iteratively locates *closer* nodes to the target ID, converging to the lookup destination in a logarithmic number of steps. Figure 2.5 shows an example lookup, where the red node with prefix **0011** issues a lookup on the blue node with prefix **1110**. As shown in Figure 2.4: the red node is aware of at least one node in the subtree with a prefix of 1, in this example this is node **101**. The first RPC is therefore to **101**,

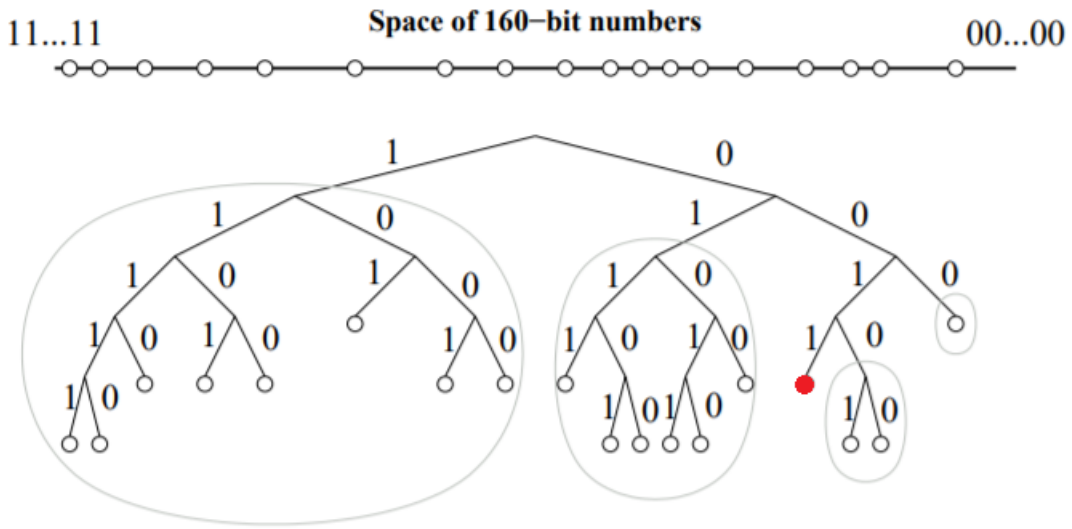

FIGURE 2.4: Example Kademlia Binary Tree [18]

who returns node **1101** details. All following RPCs are made to nodes returned by the previous RPC. Eventually the node will converge to the blue node **1110**, after 3 steps in this example. During this lookup the red node has the ability to save the routing data it has learned, meaning after many lookup operations nodes start to become increasingly more aware of the shortest routes to other IDs.

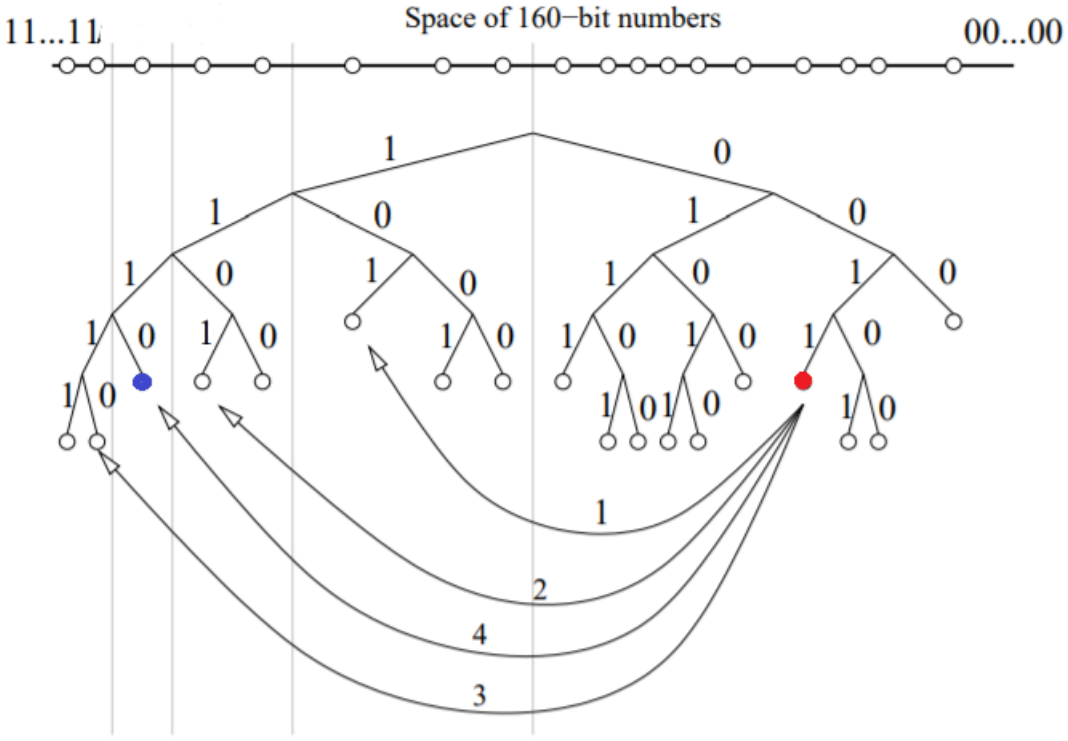

FIGURE 2.5: Example Kademlia lookup [18]

2.2.3 Content-Centric Networking

Content-Centric Networking (CCN) is of interest because unlike DHTs it uses the data itself as a key and the data becomes directly addressable. Also the way the packets are broadcast and propagate through the network will be an important idea for the way Whisper implements routing.

Content-Centric Networking [19] is a communications architecture built on named data. CCN neglects an IP-based routing method to make content directly addressable and routable. *Names* are a series of binary name segments assigned to the content by the publishers, these names individually or collectively can be used as identifiers for the data. For example the data for a segment of a tv show could have the season number, episode number, language, length and name of the tv show as names. In a CCN compliant network endpoints communicate based on named data instead of IP addresses, this makes it more flexible as names are more adaptable and can be location independent. CCN exchanges two main packet types, *Interest* and *Data*. A user requests specific content by broadcasting an *interest packet* with the content name to the network. The interest packet propagates through the network. Once data is found which satisfies the interest, the content is sent back in the form of a data packet on the reverse path of the interest packet. The data is cached locally at each node on the reverse path, this reduces the amount of hops necessary for future interest packets for the same named data. If another node broadcasts an interest packet for the same named data, the nodes on the previous reverse path now have that data cached and can fulfil the interest this time instead of forwarding the interest packet on. This results in more popular data being cached on more nodes, distributing itself across the network which makes it easier and faster to acquire.

See Figure 2.6 for an example CCN in operation. In this instance Client 1 wishes to access data named *TCD Logo* , which is the Trinity College Dublin logo as an image, a sub network of nodes (routers) on the network are shown. Initially (*Step 1*) Client 1 broadcasts out an interest packet, this propagates through the network. Eventually **Router A** receives the interest packet which is satisfied by data it maintains. It returns the content to the router it received and the content is sent backed, as a data packet, on the reverse path. At each router of the reverse path, the content is cached for future requests. This is clearly shown in *Step 2*.

In *Step 3*, Client 2 sends an interest packet for the same content. This time due to **Router D** caching the content in *Step 2*, it can return the content. *Step 4* shows the return path for that data packet.

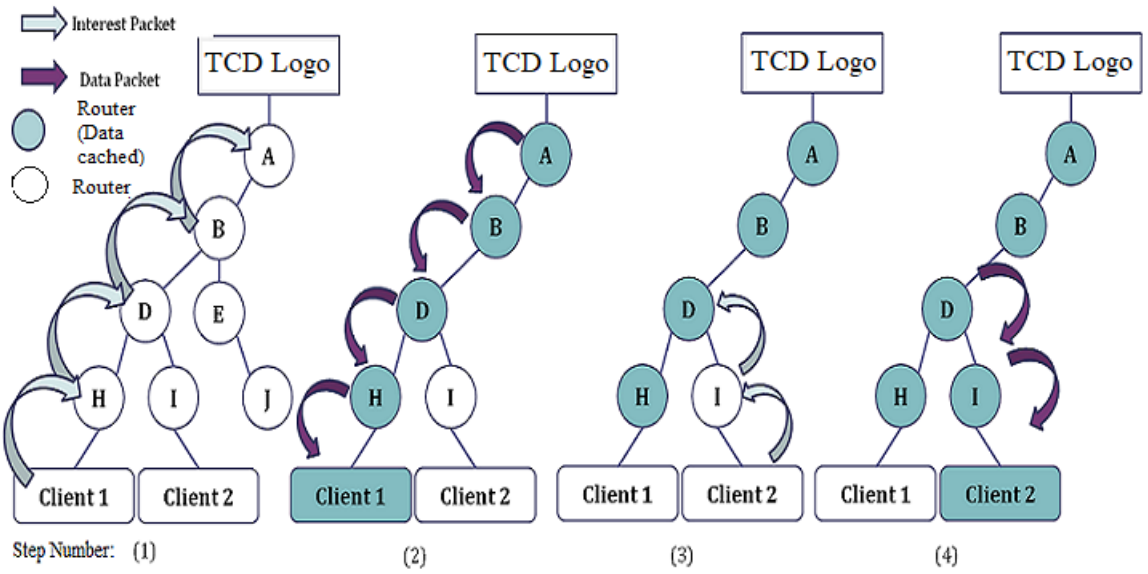

FIGURE 2.6: Example CCN [20]

2.3 Digital Identity

This section will examine identities and systems which utilise digital identities. It will detail both a centralised and decentralised approach and analyse their advantages and disadvantages.

A digital identity is a social identity which a user of the Internet establishes in online communities. It differs from our physical identity in that we can choose to be anonymous if we wish and can hold numerous online identities at any one time. Through their interaction with other users, an online identity can obtain a reputation, this can be used by other users to determine a level of trust.

2.3.1 Public Key Infrastructure

Public Key Infrastructure (PKI) is a widely adopted approach to digital identity and represents a fundamental component of the internet today.

PKI is a system consisting of both hardware and software elements to manage the creation, distribution, revocation and to ensure the integrity of keys and digital certificates. It is essentially an infrastructure for public key cryptography which inherits policies and standards. A *digital certificate* is issued by a trusted party and assert the identity of the certificate's subject and also bind that identity to a specified public key. Generally this trusted party is called a *certification authority* (CA) and they act as the root of trust. The CA provides services to the public which can be used to authenticate the

identity of other entities that they have certified. Their certification can be verified as they sign the certificates with their private key and their corresponding public key is made publicly available in a self-signed certificate. Figure 2.7 shows an example of a digital certificate issued for a user, Mario Ross. PKI has many use cases but can be generalised as a facilitator of secure electronic transfer of information across networks to trusted entities.

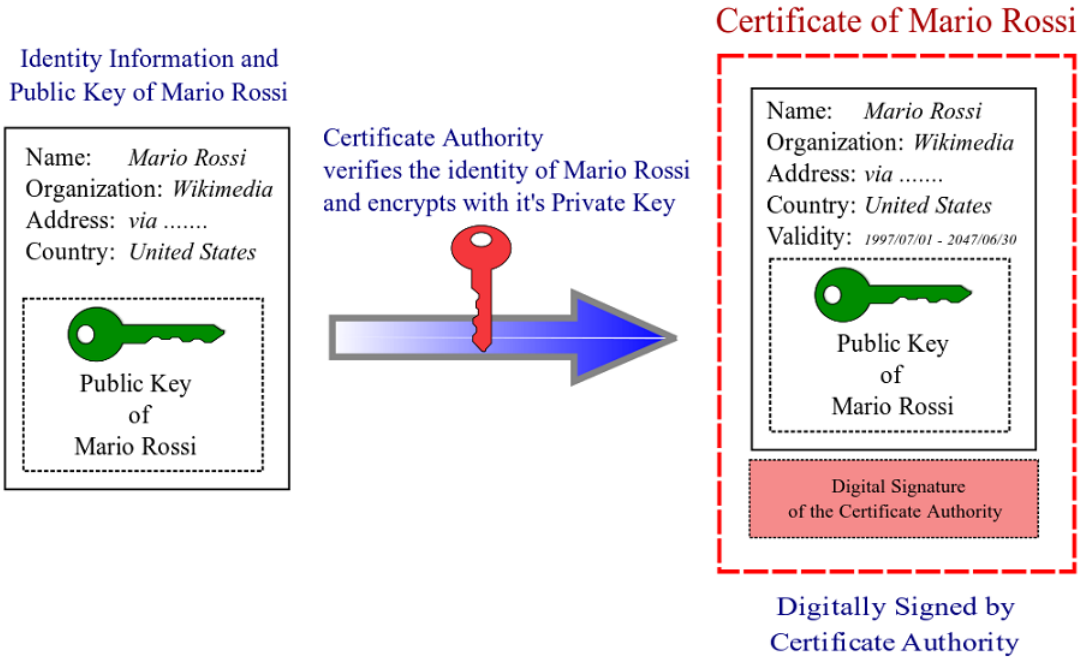

FIGURE 2.7: Example Certificate [21]

X.509 is a standard which specifies the requirements for digital certificates. X.509 is widely adopted in many internet protocols such as TLS, which is a fundamental part of the web today. A X.509 certificate major contents are: a public key, the issuers name, validity period and the identity of the certificate holder. This certificate is either signed by a CA or it can be self-signed. X.509 also specifies a process of certificate revocation, this can occur on a number of grounds such as compromised keys or change in certificate usage. A certificate revocation list is a record of all revoked certificates from a particular CA, these are generally updated on periodic intervals.

Alternative approaches to the centralised CA model include a *Web of Trust* concept which has been used in Pretty Good Privacy (PGP) systems. Certificates are instead self-signed but can also be signed with other users keys who're effectively endorsing the validity of the identity and public key specified in the certificate. When a user looks up the certificate of another user they must trust the users attesting to their identity and ultimately leaves the decision with them.

All of these PKI systems have been used throughout the internet and been foundational in securing electronic transfer of information but they do have many flaws. There exists standards covering aspects of PKI but there's no primary body enforcing these standards globally. The centralised nature of CAs is also a concerning feature, if a CA was to be compromised then all of their issued certificates could be compromised putting numerous users at risk. Revoking certificates is a complex task and compromised certificates can go unnoticed, some certificates may even require browsers to issue software updates. In 2011 a Dutch CA, DigiNotar, was hacked with over 500 fake certificates being created.[22] Web browser vendors reacted by blacklisting all certificates issued by DigiNotar [23], but the main victims were the internet users whose information was breached due to the fraudulent certificates. Another worrying issue with CAs is that the market is currently dominated by a few select companies. For instance a recent study counts two companies (Comodo and IdenTrust) having collective SSL CA market share of 71.7% [24]

PGP removes the centralised power but still has its own issues. The founder of PGP Phil Zimmermann noted his vision in the 2.0 manual:

> ...everyone will gradually accumulate and distribute with their key a collection of certifying signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys. [25]

But I'd argue the emergence he envisioned never occurred. One of PGP's main issues is the lack of a common policy for trust and certificates. By adding their signature to your certificate another user is publicly stating that they have verified the validity of your identity and certificate. Although there's no standard to adhere to for determining this validity. Consider one of the signatures on your certificate could be from your friend, who you just asked to sign it and another could be from a institution like a bank. In general the bank will be much more rigorous in validating your identity is correct before issuing their signature, this gives a higher level of trust. However in PGP both signatures are considered the same way. This rendered many multi-trip chains of trust meaningless, as a single link which you can not trust invalidates the whole chain.

Another issue in PGP is the lack of adoption, this is due to a number of reasons such as dependency on other users for trust and other PKI implementations being considered more secure. PGP has also been critised for its poor usability and the complexity of adoption. It involves many manual and separate steps from generating secure PGP keys to verifying the identity of another user before signing their certificate.

2.4 Secure Communication

This section will examine different approaches to secure communication, types of attacks on communication and how different systems avoid these attacks.

Secure communication involves two parties who wish to communicate with a certain level of assurance that a third party can not intercept it. Different methods of secure communication guarantee different levels of security, with the most secure form being a physical encounter with no possible *interceptor*. An interceptor is a third party which attempts to intercept the communication, potentially with malicious intentions. Many different tools can be utilised to achieve different characteristics of secure communication.

The most basic tool in securing communication is encryption. This allows the communication to remain secure even if intercepted, since the data is encrypted the third party (without means of decryption) can not comprehend it. Provided the encryption method is correctly implemented, the keys used are kept private and are of sufficient size then encrypted communication alone can be considered secure.

Another method is the use of an anonymous network, this a network where communications are routed in a way that it's difficult to detect what the complete message is or the endpoints of the communication (i.e Sender and Recipient). *Tor* is a well known anonymous network.

2.4.1 Attacks

There exists many practical obstacles to carrying out an attack such as resources necessary, legislation and the overall volume of communication. The advancements in secure encryption has led to focus on alternative methods of attack. Instead of attempting to breach the encryption method, by obtaining keys or finding back doors in encryption algorithms, attackers target endpoint devices and metadata leaked from the communication.

A *man-in-the-middle* (MITM) attack is an attack where another malicious party places itself in the communication where two parties believe to be directly and securely communicating with one another. There exists two main forms of MITM attacks, one requires the attack to be in close physical proximity to the target(s). Generally the attacker breaches a poorly secured Wi-Fi router, when a vulnerability is found then the attacker will insert tools in between the target's device and the internet. The other, more recent form is a *Man-in-the-browser* (MITB) attack. A MITB is a type of malicious program

which infects a user's browser and exploits vulnerabilities in the browser's security, generally MITB is quite difficult to detect and can be invisible to a general user.

A successful MITM or MITB attack can allow an attacker to read, insert and modify messages sent between the two parties communicating. An example of a traditional MITM attack is shown in Figure 2.8. In this scenario the attacker intercepts the initial request to setup a secure connection from the Client and intended for the Server. The attacker then sets up a connection with both the Client and Server separately, meaning that every message between the Client and Server is essentially sent through the attacker who can read and modify the messages as they wish. A basic MITM attack such as this can be avoided by implementing an authentication step, since this method is only successful if the attacker can imitate both endpoints.

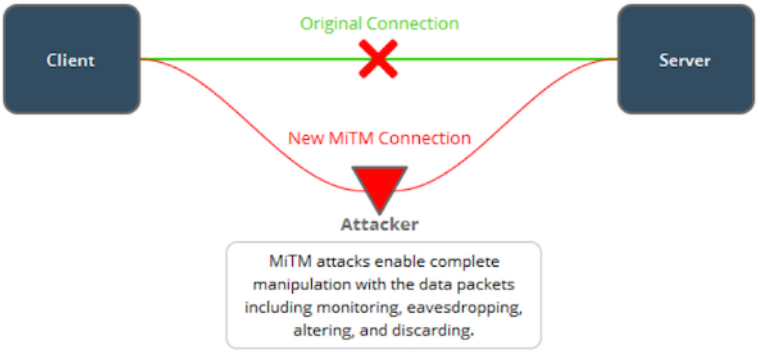

FIGURE 2.8: Example MITM Attack [26]

Traffic analysis is another form of attack, this is the act of examining intercepted messages to gather information from patterns in communication. Packet sniffing is a key method used for traffic analysis, where data is obtained by capturing network traffic using a *sniffer*, a special computer program or piece of hardware that can intercept and log traffic that passes over a network. If the packets sent across the network are not encrypted then they can be simply read and saved by an attacker. Generally packet sniffing is used to analyse and gain valuable information which can be used for subsequent attacks in a network or on a target within that network. Even in networks where communication is encrypted, the metadata can be collected and used for probabilistic attacks. The metadata itself leaks information on the communication such as endpoint IPs, protocols/software used, size of message etc. In a scenario where you wish to be remain anonymous, this type of attack can obtain metadata which could ultimately reveal your identity.

2.4.2 End-to-End Encrypted Messaging

Many messaging applications exist which advertise End-to-End Encryption (E2EE) as a feature. A traditional server based communication system encrypts messages to and from the server and relays them onto the intended recipient, as a result the server has the ability to view, edit and ignore messages. E2EE allows only the endpoints (not the intermediate server) to encrypt all of the data using secret keys stored only at the endpoints. Due to this E2EE systems have the characteristic of the server merely acting as a router for the encrypted messages and therefore even if they wished, they're unable to view the original messages.

The most popular E2EE application is *WhatsApp* which has reached 1.5 billion monthly users.[27] Widespread adoption of E2EE at this scale is promising and Whatsapp have reportedly rejected requests from government agencies in the past for a *backdoor*, a method to access and decrypt messages.[28] This reluctance has even led to the application being banned in the past in Brazil.[29] Although WhatsApp is not without it's issues, a recent paper has highlighted a potentially serious flaw in its protocol.[30] This flaw allows an unauthorised user to add someone to a group chat, even if they're not a group member. This stems from the fact that messages used for group management are not signed or authenticated, meaning the security of the EE2E group chats is dependent on the level of trust on the WhatsApp server, this undermines the purpose of E2EE. The Whatsapp server determines who is a group administrator and is authorised to send group management messages. Therefore theoretically this could be used as a backdoor, since an unauthorised third party has the ability to add themselves to a group meaning they would have access to the previously encrypted messages.

2.4.3 Tor

Tor [31][32] is the name of free software which enables secure and anonymous communication, it's an example of an anonymous network. The Tor Project is the group which empower the network, they provide information and most importantly software to participate in the network which they develop and maintain. The Tor network consists of more than 6000 volunteer-operated servers which act as *relays*.[33] Connections are made through a series of relays making up a *circuit* rather than a direct connection. This allows a user to conceal their location and usage from a party conducting traffic analysis.

The routing method used is named *onion routing*, this involves encapsulating the message along with it's metadata in layers of encryption. Unlike general internet packets,

Tor packets strip away compromising addressing information such as the senders operating system. Also Tor packets encrypt the necessary addressing information so it's only comprehensible to the intended node. The Tor packet is then routed through many relays, making up a circuit, before reaching it's destination. Each relay is only able to decrypt enough information to know which relay the data came from and which relay to forward it onto next. Every hop can be viewed as *peeling* away a layer of encryption, when the final layer is decrypted the message is at it's intended destination. Onion routing has the favourable characteristic that no individual relay knows the complete path of the packet, only the previous and following relays. Once the circuit is established, it can be used to communicate for a set time (generally a few minutes) before having to establish a new circuit.

The Figure 2.9 shows how a message can encapsulated in layers of encryption in onion routing. This shows each level of encryption as an outer cylinder. In this specific scenario a sender sends the whole onion to Router A who decrypts an initial part of the message, interprets Router B's address and forwards it on. Router B receives the packet and also decrypts a part of the message. This continues until the message is finally received at the destination from Router C, in the figure it shows the message in orange which is solely encrypted with the destinations public key. This is a typical 3-hop relay circuit.

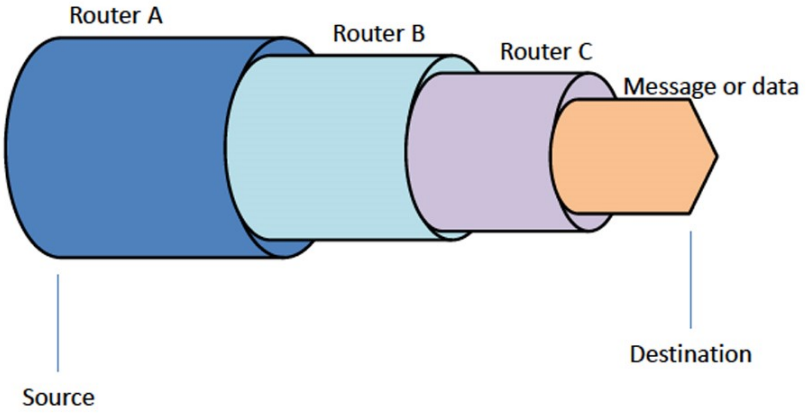

FIGURE 2.9: Onion Routing [34]

To provide anonymity to servers on the Tor network they can be configured to only setup connections within the network itself. These servers are commonly known as *hidden services* and more notoriously make up the *dark web*. Hidden services can maintain their anonymity too as they're accessed by an *onion address* rather than a typical IP address eg) **facebook.onion** and through the Tor browser.

The service randomly chooses relays to act as *introduction points*, it builds circuits to them and provides them with it's public key. This is advertised in a publicly available

DHT where an entry contains a descriptor (of the service), it's public key and the chosen introduction points information. The connection does not leave the network and takes place at a *rendezvous point*, a randomly chosen relay which acts a meeting point for the client and service to communicate using a one-time secret. The client initially sends an *introduce* message to one of the introduction points, requesting it be sent onto the intended service. This message includes the address of the rendezvous point and the one-time secret, this is encrypted using the service's public key. Both the client and the service create a circuit to the rendezvous point and once the connection has been successfully established they can communicate securely and anonymously. The rendezvous point is essentially oblivious to the communication and the endpoint identities, it simply forwards on the messages it receives from either end. A random rendezvous point must be chosen as no single relay should be singled out as being linked to a certain hidden service. The Figure 2.10 shows the basic connection between a user, Alice, and a hidden service at a rendezvous point, RP. It abstracts the Tor circuits for clarity, in reality these would not be a direct connection but an onion routing circuit with many relays. Also there would be many other relay nodes in the Tor cloud/network.

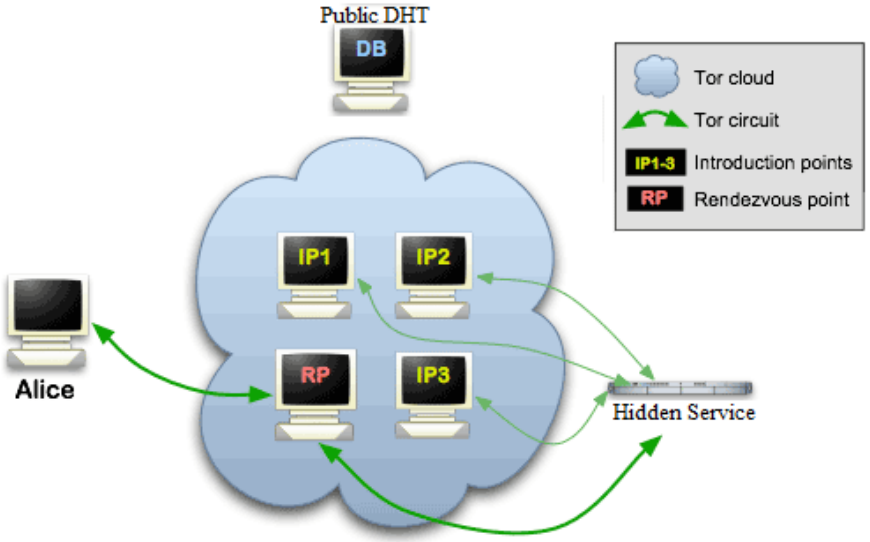

FIGURE 2.10: Hidden Services Example [35]

2.5 Blockchain

This section will examine Blockchain technology, it's origins and specifically look at the implementation of Bitcoin. Blockchain is built on a P2P network and has cryptography and PoW elements.

Blockchain is often referred to as an *'enormously powerful shared global infrastructure that can move value around and represent the ownership of property'*. The blockchain

technology was first described in the Bitcoin white paper [36] by Satoshi Nakamoto in 2008. The paper proposed a new solution to online payments, which allowed peer-to-peer exchange of digital currency in a trust-less environment without the need for a trusted authority such as a financial institution. It uses a distributed ledger data structure to maintain order to transactions and digital signatures to verify transactions.

Bitcoin due to it's underlying blockchain became the first digital currency to solve the double spending problem while continuing to be decentralised. In particular, it uses a peer-to-peer distributed timestamp server which can generate computational proof of the exact order of transactions.

A new transaction is broadcast to all nodes in the Bitcoin network. It is added along with other unconfirmed transactions into a *transaction block*. This block can be identified by its hash, it contains a nonce and a reference to the previous block. In Bitcoin, a *merkle tree* of transactions is created and hash of the root is included in the block header instead of a full list of transactions. Merkle trees optimise the disk usage while maintaining the same level of validation.

A merkle tree is a tree data structure in which every leaf node is the hash of a data block and every non-leaf node is a hash of it's child nodes. This continues up the tree until a single hash remains, this is known as the *merkle root*. Merkle trees allow for efficient and secure verification of large data content. Git is an example of an application outside of cryptography which uses merkle trees.

Figure 2.11 shows an example of a Bitcoin block and the merkle tree for the included root hash. At the leaves of the merkle tree are the hashes, hash 0-3, of the respective transactions (Tx0-3). These are hashed together until it reaches a single output which is included in the block header as the root hash. A real Bitcoin block could contain hundreds of transactions.

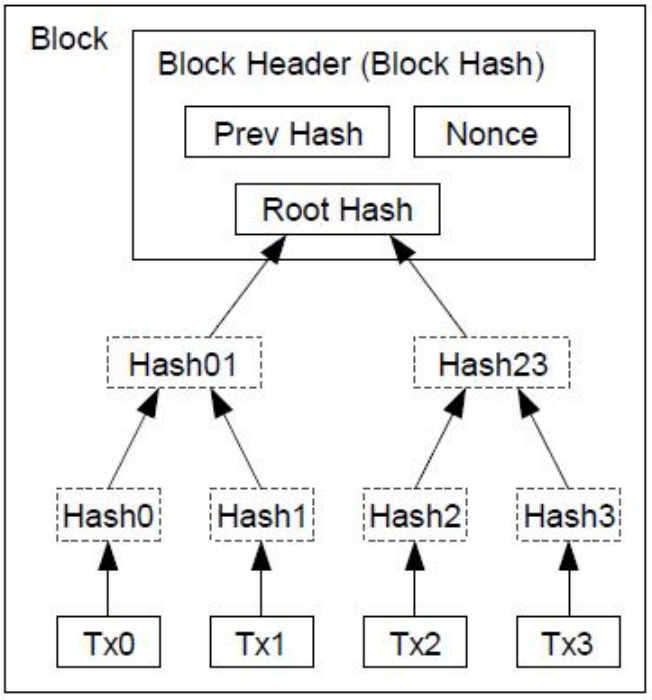

FIGURE 2.11: Bitcoin Block Example [12]

Many blockchains use a PoW system to reach a distributed consensus using the nonce included in the block. Nodes can contribute to establishing consensus by *mining*, this is the act of generating new blocks and then attempting to find a valid nonce for that block. When this is found it broadcasts the new block to all nodes, they will accept the block only if all the transactions are valid. Nodes express their acceptance of a block by creating a new block and using the hash of the accepted block in the previous hash field.

In order to change the data in one block, all successors of that block must be re-written which would be very costly and difficult. This is because the entire block is hashed, changing one transaction would change that hash which in turn would invalidate the next block's previous hash field. Updating this field would invalidate the next block's hash and this continues to the end of the chain. In addition, the longest chain is always accepted by the network and shorter chains are discarded.

Blockchains require full nodes to store a full copy of the entire blockchain including all transactions locally which can be costly. For this reason some blockchain implementations require only the block headers to be stored, if that user is not interested in mining. The required PoW difficulty is determined by observing the average number of blocks per hour, increasing the difficulty if blocks are generating too fast. Generally in return

for computational resources used to verify blocks, *miners* are rewarded, currently Bitcoin miners receive 12.5 BTC per block. Due to the rise of cryptocurrencies, mining has become very lucrative and has had knock on effects in other industries.

2.6 Ethereum

This section extends from the previous section on Blockchain technology to look in-depth at the elements which make up a popular blockchain implementation called Ethereum.

Ethereum is a decentralised blockchain platform with a cryptocurrency called *Ether*. At the time of writing this paper it's currently the second largest cryptocurrency by market cap behind Bitcoin.[37] However Ethereum aims to be much more than a cryptocurrency by providing the functionality to build and use decentralised applications (Dapps). Dapps work over P2P networks rather than centralised client-server networks. Ethereum achieves this by building out a foundation level which includes specialised programming languages allowing developers to create and manipulate *smart contracts*. These programming languages get executed in an Ethereum Virtual Machine (EVM). Ethereum borrows many ideas from past solutions such as Bitcoin but intends to be a more generalised platform by being featureless, i.e not supporting any common high-level use cases as intrinsic parts of the protocol.

It must be noted that although the Ethereum platform has a vast amount of resources invested into it, it's certainly not yet complete and in a constant evolving state. One instance of this is the future *Casper* update [38] which will fundamentally change the protocol by introducing Proof of Stake (PoS). PoS is an alternative to PoW for achieving distributed consensus. The creator of a new block is chosen in a deterministic way, depending on their wealth, known as a *stake*. This wealth in the form of Ether is locked up and can not be used for a predetermined period of time. There's no block reward and therefore *stakers* receive the transaction fees instead. PoS aims to improve energy efficiency and be cheaper compared to current PoW implementations.

2.6.1 Origin

Ethereum [39] was initially described in a white paper written by Vitalik Buterin in 2013 with the goal of developing '*A Next Generation Smart Contract and Decentralised Application Platform*'. In the paper he singled out Bitcoin in particular due to it's introduction of blockchain technology and distributed consensus. Although he clearly stated the limitations of Bitcoin and was particularly passionate about the introduction of a

scripting language to enable application development. Later in January 2014, Buterin along with a core team of supporters went public with Ethereum. [40]

2.6.2 EVM and Ether

The EVM is a foundational component of Ethereum, it runs code written in a range of languages such as Solidity, of arbitrary algorithmic complexity in a deterministic manner. The network must verify all state changes on the blockchain to maintain consensus. Code that is executed within smart contracts is also considered a state change. Every node on the Ethereum network runs the EVM and executes identical instructions within it, this is to maintain consensus through code execution across the blockchain. For this to occur all nodes must also reach the same state following execution ensuring deterministic execution of smart contracts. Since the smart contract code is run on all nodes across the network it guarantees fault tolerance, immutable state changes and censorship resistance.

Ether is the main currency within Ethereum, its main purpose is to act as *fuel* for the Ethereum blockchain. It is used to pay transaction fees in terms of *gas*, the fee paid for computation of a transaction within the EVM. The base unit of Ether is called a *Wei* and other denominations of Ether are expressed in terms of Wei. The gas is intended to stay at a constant cost in terms of the network resources required. Therefore the *Gas Price* is calculated in terms of the current value of Ether, ensuring that transaction costs do not become too high or low to be feasible. Gas also brings the benefit of spam prevention since transactions have a cost associated with them, spam would become very expensive.

Ether is generated to reward miners as part of the PoW system but it can also be obtained on major cryptocurrency marketplaces such as Coinbase. Since it can be traded for other currencies it can be given an American Dollar (USD) valuation, A single Ether (1^{18} wei) is valued at $663 at the time of submitting this paper (May 2018). Although as with many cryptocurrencies Ether has been extremely volatile.

2.6.3 Accounts

Accounts are a significant component of Ethereum as they are the source of state. Each account has a 20-byte address and an ether balance. Transitions in state are due to direct transfer of value or information between accounts. There exists two types of accounts: Externally Owned Accounts (EOA) and Contract Accounts, the main differentiation between the two is that EOAs have no code. Contract accounts (generally referred to as contracts) do have associated code which can be triggered by transactions. Furthermore

all actions on Ethereum stem from transactions fired by EOAs. Contract accounts are what makes Ethereum unique since EOAs alone are essentially a coin system to transfer ether.

2.6.4 Transactions

A transaction [41] in Ethereum refers to the signed data package to be sent from an EOA to another account on the blockchain. If the destination is an EOA it could be used to transfer Ether. Otherwise if the destination is a contract, then the transaction can activate the contract and the contract's code will be executed automatically. A transaction contains:

- Eth Address of recipient
- Signature from EOA private key
- VALUE - Amount of Wei to transfer from sender to recipient
- (Optional) Data - Contains message
- STARTGAS value - max number of computational steps the transaction execution can take
- GASPRICE value - fee the sender is willing to pay for each unit of gas used

2.6.5 Solidity

Solidity is a high-level language specifically for developing smart contracts. It was initially proposed with the intent to design a scripting language to target the EVM and at present is still the primary language. However alternative programming languages exist such as Serpent and Viper. Solidity is a statically-typed language which is intended to abstract complex implementation elements, allowing the majority of use cases to be a small amount of code. It was influenced by major languages such as C++ and JavaScript in order to be familiar to many developers.

2.6.6 Smart Contracts

Smart contracts in Ethereum are effectively high-level programming objects which are developed in a scripting language targeted at the EVM. The code is compiled to EVM byte code and deployed to the blockchain for execution. They consist of code (functionality) and data (state), Once deployed to the blockchain these smart contracts are

public and immutable and reside at a specific address. However Smart Contracts can be interacted with and functions can be invoked when a transaction is sent to that contract. Contracts cannot execute native operations such as API calls and must be prompted by an EOA. As all nodes will execute the same contract code they must reach the same outcome, this guarantees deterministic execution of smart contracts deployed to the blockchain. Ethereum claims that Smart Contracts are Turing complete which makes verification difficult.[42] Once the smart contracts are public it limits the possibility of preventing exploitation of bugs and security holes. One example of this occurring was the attack on The DAO (Distributed Autonomous Organisation) in June 2016. [43] Although it was clear to the community the source of the bug it could not be quickly fixed. This resulted in a loss of $50 million for many Ethereum account holders and later a hard fork occurred to restore the lost funds. There have also been many other attacks [44], which given the considerable value of finding an exploit is to be expected.

2.6.7 DEVp2p

This section so far has analysed key elements of the Ethereum implementation of blockchain technology. But it has not yet been explained how the nodes in the Ethereum network communicate.

Ethereum required a unique network stack and as a result DEVp2p [45][46] was developed. It's a general protocol for the discovery and connection of Ethereum nodes. Sub-protocols are defined on top of it, these currently include Ethereum (eth), Swarm (bzz) and Whisper (shh). The eth protocol, which confusingly has the same name as the platform itself, is essentially the Ethereum blockchain. bzz is a distributed storage platform and shh is an unique decentralised messaging protocol. Devp2p also provides the network infrastructure for P2P communication between nodes running these sub-protocols. They communicate in terms of packets which includes a message-ID. Each sub-protocol is assigned a certain message-ID space in that format so a node can infer which protocol should handle the packet. These packets are used for any communication between the nodes in the network. For example this could be a node broadcasting a newly verified block through the eth protocol or a message being sent through the shh protocol.

2.6.7.1 Node Discovery

The node discovery protocol [47] is an augmented implementation of the Kademlia DHT that stores data about Ethereum nodes. Every node is assigned a key which acts as its public key but also as a *node ID*, this is the node's unique identifier. The XOR metric,

used to calculate the *distance* between nodes, is a bitwise XOR on the hashes of the two node IDs. This results in a network where peers considered in the *neighborhood* of a particular node must have similar IDs. The Kademlia structure is used as it yields a topology of low diameter.

2.7 Whisper

The following section will explore a sub-protocol of Ethereum, namely the Whisper protocol and how it operates. It's particularly important because it's the underlying messaging protocol used by the messaging application.

Whisper [48][49] is a sub-protocol (**shh**) in the Ethereum ecosystem which utilises the DEVp2p transport layer. The protocol provides a low-level API and similarly to Ethereum itself, is implementation agnostic. It is a decentralised peer-to-peer and pure identity-based messaging system. Meaning the sending and routing of messages is not based on user attributes such as IP addresses, therefore there does not exist any singular endpoints in the system. The Whisper network is made up of nodes which run an implementation of Ethereum such as Geth [50] (implemented in the Go language) and have the Whisper protocol turned on.

At a basic level messages are not sent directly to the recipient(s) but broadcast to the entire Whisper network. Where all nodes are expected to continuously receive and forward on packets to their peers, even if they're the intended recipient. For example, If I wish to send a message to my friend, I send the message into the network via my peers rather than directly to their node. The message will eventually propagate to their node due to the continuous forwarding and they will successfully receive it. But in this scenario I might never know my friend's node details, he won't be able to know from the message alone who and where it is from nor will any of his peers know he was the intended recipient. Resulting in our identities remaining private.

Whisper is not a typical communication platform, nor is it designed to replace traditional protocols such as UDP or HTTP. It was developed to provide the next component of DApps which require efficient broadcasting and support for low-level asynchronous communication. All communication systems require a trade-off between efficiency and both resilience and privacy. The main aim of this protocol is to provide a user-configurable amount of *darkness* at the expense of both bandwidth and latency. Theoretically Whisper can deliver 100% darkness.

2.7.1 Messages

Messages sent between Whisper nodes consist of two key elements: *Envelopes* and *Messages*.

Envelopes are essentially the data packets sent between Whisper nodes. An envelope contains an encrypted payload and the envelope metadata in plain text. This metadata is necessary for routing and decryption. The envelope is sent via DEVp2p, which also uses its own layer of encryption. Envelopes contain the following fields:

- Expiry: Unix time stamp of envelope expiry
- TTL: Time To Live (in seconds)
- Topic: 4 bytes of arbitrary data
- Data: Byte array representing encrypted payload
- Nonce: 8 bytes of arbitrary data (used for PoW)

PoW is introduced to the envelopes for spam prevention and to control network traffic. The PoW value is also used in routing and envelopes with higher values are given priority. The PoW value in Whisper is more than just the difficulty, it also includes the TTL and message size. This means if you require a larger message or for the envelope to live longer in the network then the resources required to achieve a certain PoW value will increase. The default PoW target for Whisper is 0.2, this is what all envelopes must achieve to be received. This takes on average a few milliseconds on a general PC.

Messages refer to the envelope's payload in plain text. All messages must be encrypted either symmetrically or asymmetrically, therefore messages can only be decrypted by the owner of the corresponding key. A message has the following format:

- Flags (1 byte) - Indicates message metadata
- Signature (Optional) - Signature of original sender
- Payload - Data of arbitrary size

The signature, if provided, is in the form of an ECDSA signature of the hash of the unencrypted data using the secret key of the original sender. As the signature is only included in the message and not the envelope, the signature is only revealed to those who can decrypt the payload.

2.7.2 Topics

Topics are simply four bytes of arbitrary data, this means there's over 2 billion potential topics. **0xffddaa11** is an example of a topic, this could be generated randomly or from the hash of a keyword. A topic is included in every envelope and acts as an identifier. It's used to provide a probabilistic hint about the encryption key used. For example, If I was subscribed to a newsletter which is sent to the network with the example topic above and I also possess the symmetric key used for encryption. If an envelope is received with that exact topic then the probability is high that I can successfully decrypt the message.

A topic collision occurs when an envelope with a known topic can not be decrypted with the corresponding key. It would require a large and busy network for topic collisions to occur regularly but they're desired for darkness, since being the only node(s) on the network looking for a particular topic could be compromising. Partial topics can be introduced to raise the probability of a topic collision occurring, this is where only the first number of bytes are set and the rest is randomly generated. **0xffdd**____ is an example of a partial topic where only the first two bytes are set.

2.7.3 Filters

Filters indicate if a node should attempt to decrypt an incoming envelope. It contains a key and a number of conditions. If the conditions are not satisfied then the node does not attempt to decrypt the envelope, since attempting to decrypt all envelopes would be infeasible. These conditions include:

- Array of topics/partial topics which must match envelope's topic
- PoW requirement (Minimum PoW value for incoming envelopes)

Filters are created by Dapps using the API and a single node can have many filters. The node handles the incoming envelopes and if it successfully fulfils the filter it's saved in an envelope pool. The Dapp is expected to regularly poll the node for new envelopes and if present they're returned.

Receiving and forwarding all messages could obviously become very resourceful and inefficient. This is how the topics come into the routing, nodes are able inform their peers of the topics they're interested in. Then those peers will only forward on envelopes with those topics. In turn if you pick any node in the network, they consolidate their topics of interest and topics their peers are interested in and this will be advertised to their own peers. In this way a single node advertising their interest in a topic will propagate

through the network, creating a path from peer to peer ensuring those envelopes will delivered.

It's clear that informing the network what topics you're interested in could be compromising, for this reason a bloom filter can be used. A bloom filter allows a node to consolidate their topics and advertise them as a bit mask of 32-bits. This allows topics to be described in an incomplete manner. However this will introduce inefficiency since it raises the probability of undesired envelopes being delivered.

2.7.4 Operation & Routing

Whisper nodes are expected to receive and send envelopes continuously. They maintain a map of envelopes, indexed by expiry time which is managed accordingly. It should deliver decrypted messages to the front-end API while maintaining the mapping between DApps, their filters and envelopes. When a node's memory is exhausted, they may drop envelopes it considers unimportant for both itself and it's peers.

To send a message, the envelope is created on the DApp level and posted through the Whisper API. The envelope is placed in the node's envelope pool and then forwarded onto other peers in due course. Essentially an authored envelope is treated identically to a received envelope.

Whisper is designed to be able to **route probabilistically** with two major characteristics: Leak minimal routing information and be resilient to statistical attacks (from metadata collection).

The underlying DEVp2p layer provides the functionality for *leaking minimal routing information*. The Whisper protocol rates a node's peers with the aim to *steer* the set of peers to those who deliver *useful* envelopes. An envelope may be considered useful for numerous reasons such as matching a local filter's topic or having a high PoW value. Nodes can blacklist peers which deliver expired or invalid envelopes. The act of peer-steering, changing peers in search of ones which will have a higher rating, provides an incentive for nodes to provide useful information to their peers. If they're identified as a poorly rated peer they run the risk of being rotated out in favour of another node.

The second characteristic is achieved by nodes advertising their topics of interest to one another. Bloom filters are key to reducing the amount of information leaked and render statistical attacks more difficult. The level of metadata leaked to peers from topic advertising should be established by explicit user settings and traffic modelling. The distribution of useful information to total traffic must be determined in order to narrow this setting down. For maximum efficiency a node can inform a peer of a complete topic,

depending on the sensitivity of the messaging this may be desired. Also apart from their own interests, a node must develop filters for their peer's interests in order to ensure constant delivery of useful information.

Figure 2.12 shows part of a Whisper network and will be used for an example. The node AB is sending a message with a topic, $topic_A$, encrypted with a symmetric key. Other nodes AC and BC have a filter created for that topic and the symmetric key used for encryption. The dotted lines indicate that the connected nodes are each others peers. All nodes with a red circle around them are nodes who're actively listening for the topic, $topic_A$. This shows how the interest in this topic has propagated through the network, these nodes are not listening for this topic for their own use but looking to forward on envelopes to their peers who have expressed an interest.

Initially the envelope will be authored at Dapp level and posted using the Whisper API. Authored envelopes are treated identically to all other envelopes and therefore it is placed in the envelope pool. Eventually the envelope will be forwarded through the network and reach both AC and BC. As previously mentioned, they have a filter which this envelope satisfies and they're able to decrypt the message. Even though they were successfully decrypted the message, they still forward that envelope on. When their Dapps poll for new envelopes for that specific filter, it will be returned. The envelope will continue to propagate through the network until it's TTL expires then it will be dropped. None of the other nodes successfully decrypted the message. They were either only forwarding the envelope onto their peers or a topic collision occurred.

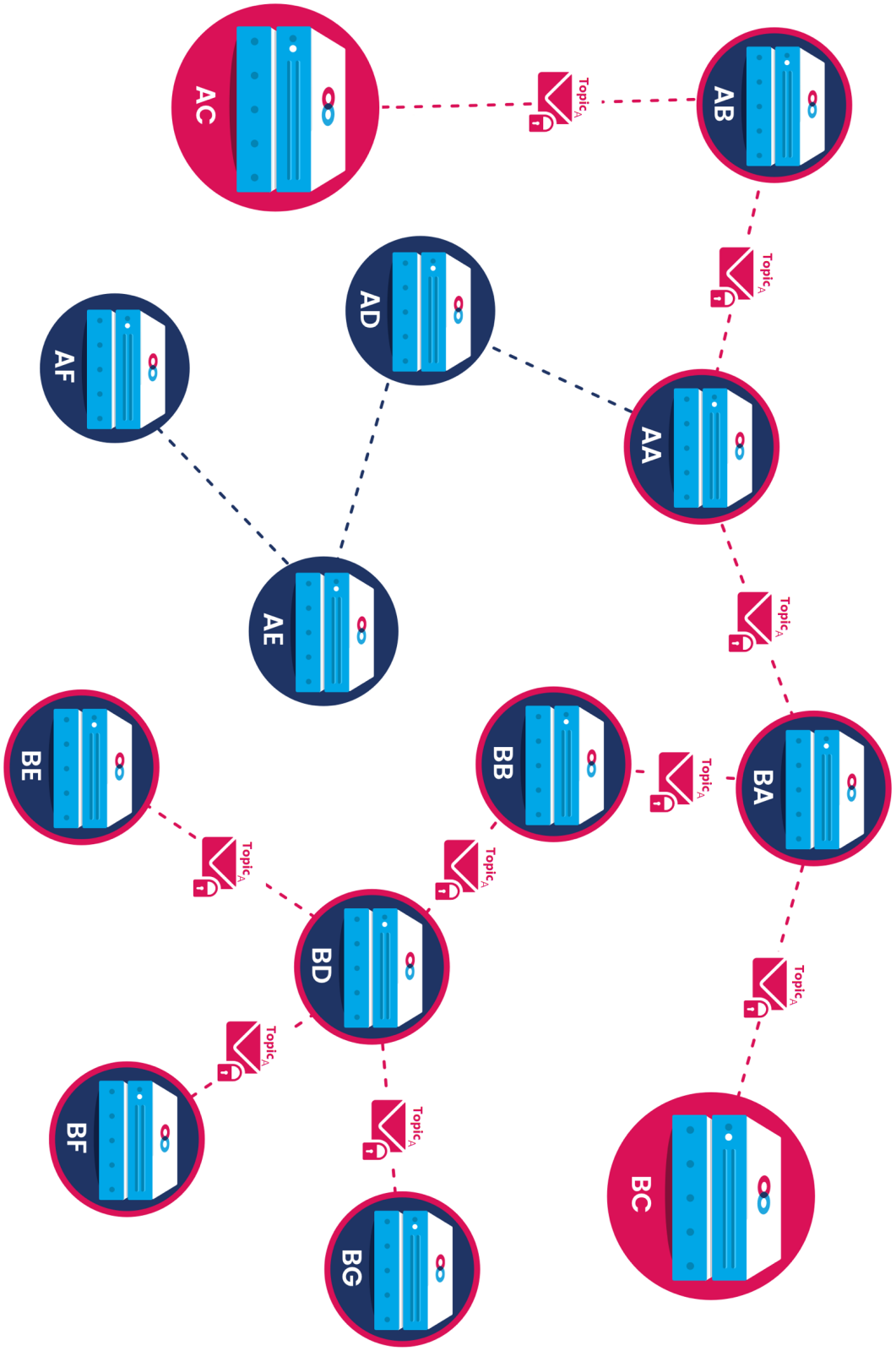

FIGURE 2.12: Whisper Example [51]

2.7.5 Darkness

Darkness is the characteristic which makes Whisper unique compared to other secure communication protocols, the following is a description from the Whisper specification:

> A truly dark system is one that is utterly uncompromising in information leakage from metadata. [48]

It's best to differentiate between encryption and darkness by considering different forms of attacks which can be carried out on a network of Whisper nodes and detailing the strategies that aid in achieving complete darkness. Decentralised and encrypted communication can still be exploited through bulk metadata collection and packet-sniffing. For the following scenarios consider a resourceful adversary which has the capacity to run many nodes. They can also exploit the network topology by targeting nodes and becoming their peer, this can be done by generating node IDs similar to the target node ID.

Through simple traffic analysis it is still impossible for the adversary to identify the recipient of a specific envelope, but the sender can be identified through the topology fixing mentioned above. The sender could maintain a constant level of *noise*, i.e send an average amount of useless envelopes encrypted with random data over a set period. This will make it make it more difficult for the adversary to recognise the sensitive messages, also increasing the amount of resources necessary to handle more envelopes. However if two nodes constantly communicate with a common trait such as the message size or topic used, both the sender and recipient could be recognised. Since the size of a message (payload in envelope) is a leak of metadata, padding is introduced. Padding is used to align all messages to 256-byte boundaries. It's also highly recommended to regularly regenerate topics for a session and to use separate topics for sending and receiving messages.

It's not feasible to have filters without a topic since a node would be required to attempt to decrypt every incoming envelope. Topic collisions introduce plausible deniability, since messages from different nodes could be sent with the same topic but encrypted with an unknown key. Therefore topic collisions are desired to ensure it's more difficult to tie a certain topic or set of messages to a particular node. It's recommended to set the topics dynamically with respect to the amount of traffic in the network, making use of partial topics if necessary.

Plausible deniability also ensures leaked messages can not be linked to an identity with 100% confidence, if using a symmetrically encrypted communication channel on Whisper

and they do not sign their messages. One user may be sure that they're communicating with a specific user but their communication can not be used as evidence, since the message(s) could have been sent by anyone with knowledge of the symmetric key and topics. If the messages are signed using a known private key then they would be verifying their identity and lose plausible deniability.

Steganography can also be introduced for added security, this is the act of concealing data within some other data, for instance you could conceal a password in the pixel values of an image. In the Whisper protocol the padding could be used to conceal data, since padding added by default it would be impossible to distinguish between randomly generated or encrypted data bytes. This however would rely upon an encryption method which does not reveal metainformation.

2.8 Web 3.0

This final section will examine Web 3.0, a core motivation behind this project.

Web 3.0 [52] is a term which originates from a blog post by Gavin York in 2014, interestingly he was a core founding developer of both Ethereum and Whisper. He envisioned a completely new and re-imagined way of interacting with the *web*. He clearly states the need for current systems to be redesigned fundamentally according to a new zero-trust interaction system. Currently we use Web 2.0 which is dominated by centralised trust authorities and client-server models. We relinquish control of our identity and data to organisations to exploit and must trust our information to many arbitrary entities to use the internet. Web 3.0 looks to eradicate these issues by providing a new model for interaction between parties over the web which is decentralised and enforces our assumptions of our information usage. It is declared with four major components: static content publication, dynamic and private messages, trusted transactions and an integrated user-interface.

A decentralised, encrypted publication system would be utilised to publish publicly any static portion of information that we're happy to share. This exists currently in Web 2.0 in the form of BitTorrent, but combined with other elements of the new web this could see much higher adoption and be more efficient. At the time of writing this paper, a lot of progress has been made in this area. One such example is IPFS [53] (InterPlanetary File System), IPFS is a distributed file system where the underlying distribution protocol is addressed by content and identity. IPFS is completely decentralised and authenticated automatically. Since files are addressed due to content, authenticity is guaranteed.

IPFS incentivises it's users to maintain and share the data using Filecoin. Filecoin is a cryptocurrency which was developed to provide an incentive layer on top of IPFS.

The second component is an *identity-based* secure messaging system, which allows numerous different entities to communicate on the network. This system would have to adopt strong cryptography in order to provide guarantees over the messages. Any form of communication should take place in encrypted channels. As a result of being identity-based, communication can become untraceable due exclusion of previous Web 2.0 addresses (IP addresses etc). Ethereum developed Whisper as a solution to this component. It provides resilience and secure communication at much higher expense vs current Web 2.0 solutions such as WhatsApp. As previously stated, Whisper can theoretically deliver 100% darkness, guaranteeing anonymity.

The third component involves a system based on decentralised consensus, which we have seen implemented in blockchain technology. The major usage is to agree upon set rules which will automatically result in their enforcement in the future. Since a failure of a previous transaction could have knock-on effects for proceeding transaction there's motivation to reduce this risk by making all users into stakeholders. Information we assume to be agreed can be placed on a consensus-ledger giving us immutable records and allowing for trusted transactions. Consensus is not used for transactions and verification in Web 2.0, instead there exists a reliance on centralised authorities such as Comodo, Google etc. Implementations of blockchain such as Ethereum and Litecoin provide such transactions.

The final component is the user-interface which allow users to interact with Web 3.0. This is arguably one of the most important components as all these other services rely upon the assumption of a large integrated user base. The success of the Web 3.0 will ultimately be reliant upon it's user adoption from the familiar Web 2.0. For instance IPFS will have very limited functionality if there are only a few hundred users or Whisper is redundant if none of the entities I want to securely communicate with use it. Many user-interfaces have adopted the similar look and feel of a Web 2.0 browser such as Google Chrome. One example of this is Status [54], although their mobile app is still in early development it is promising to be a user friendly, easily accessible interface for Ethereum DApps. Considering smartphone usage has now exceeded desktop usage, the trends suggest targeting the mobile platform will be the best route for wider adoption. The app is branded as transforming your mobile device into a light client Ethereum node, allowing access to the full Ethereum ecosystem from anywhere. If the Ethereum ecosystem matures and it's development continues, Status could become one of the first fully functional Web 3.0 clients. Since Swarm, Whisper and the Ethereum blockchain (all sub-protocols of Ethereum) provide the first three components.

Chapter 3

Solution Design

3.1 Goals

We start with an overview of the goals we strive to achieve with the solution. The primary goal is to develop a messaging application with the following features:

- Ability to engage in direct or group messaging
- Supports End-to-End encryption
- An intuitive user interface
- Preserves backward and forward secrecy
- Spam prevention
- Maintains anonymity and leaks no compromising information from metadata - Achieve a high level of darkness

As with most decentralised applications, the strength of the Whisper network is in the number of users it has. Therefore it will be important for the UI to be straightforward and easy, abstracting the complex operations since this will help to attract and retain users to the application.

Backward and forward secrecy are important features of communication in messaging applications. Backward secrecy refers to the addition of a group member where that new member is unable to view the messages sent before they joined. Forward secrecy refers to the removal of a member where that member is unable to view messages sent after their removal. Both of these features guarantee that messages sent will only be viewable by the intended recipients.

The solution will aim to prevent spam by using Whisper's envelope design which includes PoW. A user should be allowed to specify a PoW value which assures them the sender has spent a certain amount of both computational resources and time in order for the message to be received. For example, if the PoW value is set by all Whisper nodes to a value which takes on average one second per message, this would make spam very expensive.

Current messaging implementations fall short in concealing a users identity but this solution will aim to achieve this. As with the Whisper protocol itself the solution will have multiple options which will increase or decrease darkness. Therefore the user has control over how much efficiency they're willing to sacrifice in return for increased security.

A minor goal is to analyse the Whisper protocol. In particular, to determine if it's feasible to implement and if it delivers on the characteristics it promises in its specification.

3.2 Design

The purpose of this section will be to explain how we have taken the technologies and ideas from the research to develop a solution to the issues outlined in the introduction and that achieves the goals detailed in the previous section.

3.2.1 System Overview

The system developed is made up of many key components that will be explained in this section.

The *base identity* of a user includes an asymmetric key pair, a topic and a minimum PoW value, this is where the user can be contacted and how other users can initialise a group with them. The application creates a constant filter with these details. The minimum PoW value is the minimum value must have in order to be accepted by a user's node, otherwise they're ignored. These details could be publicly shared on some other platform so it would be easily obtainable for other users. To add another user to your list of contacts, you simply provide their public key, topic and minimum PoW value. To send a message to another user it is required to have them as a contact.

A *group channel* is simply a group of users communicating. It can have one or more members so it encapsulates both direct and groups of users. Each group channel has a randomly generated session key, which is a shared symmetric key, also a set of randomly generated topics where each member of the group is assigned their own topic.

Messages sent in a group channel are encrypted using the same session key but are sent to every topic in the group's set of topics. Each member's system is aware of their topic in this set and creates a filter with that information. Essentially each group member is actively listening or subscribed for their own assigned topic. Separate topics are used since a single topic significantly increases the chances of the messages being probabilistically identified. Especially if you consider a large group channel communicating regularly, it would become clear some communication was using that exact topic.

The exact Whisper API function for sending a message is known as *post*. This encapsulates encrypting the message with a provided key, creating the envelope with the provided topic and TTL and sealing the envelope by computing a nonce for the PoW target provided. Finally it sends the envelope through the local Whisper node.

A group channel is created by a member and they become the *group controller*. They act as the group administrator and have the ability to add or remove members. They're also responsible for generating new keys and topics, and handling session timeouts and subsequent rekeys. The group controller results in the application having a centralised group key management system. However messages are not relayed through the group controller and therefore they not control the messages, meaning they can not intentionally block a message from being delivered.

A centralised group key management scheme has a number of advantages and disadvantage. The following points were taken into consideration before ultimately choosing to implement a centralised group controller. Advantages include:

- Reduces load on other group members
- Less setup overhead in contrast to decentralised or distributed schemes
- Control over the group and it's members rests with original creator
- Mitigates compatibility and security issues with different users computing keys
- Maintains fact that initiator is only member which is aware of true identity of all group members. Since other member will only be aware of the amount of group members unless otherwise disclosed

The disadvantages of a centralised group controller include:

- Single point of failure - The group is disbanded once the controller is no longer active
- Group members must trust the controller and have no control over group actions

- Potential delays and unintentional member loss due to an inconsistent network. Packets can be lost or delayed meaning a member could miss crucial group information such as a change of encryption key

When a *rekey* occurs a new session key and set of topics are generated for a group channel. The group controller is responsible for the rekey and informing all members of the newly generated details. Those new details are then used for the group communication. A rekey can occur on a session timeout or due to membership changes in the group.

A *session* refers to a period of time where the group channel uses a certain key and set of topics. This is essentially the time in between rekeys and when a session times-out a rekey occurs. The amount of time a session can exist is predefined in the application's settings. The more regular a rekey occurs the more difficulty traffic analysis will become.

3.2.2 Message Structure

All messages follow a similar structure, this allows the system to be modular for potential additions in the future. The current message types are:

- INIT - Informs all members of the newly created group
- REKEY - Informs all members of a rekey
- END - Informs another member or all the members that the group has ended.
- EXIT - Informs group controller that a particular member wishes to leave the group
- General Message - To send a message to all group members

Figure 3.1 shows the messages and their fields. This message is encrypted and sent as the envelope's encrypted payload.

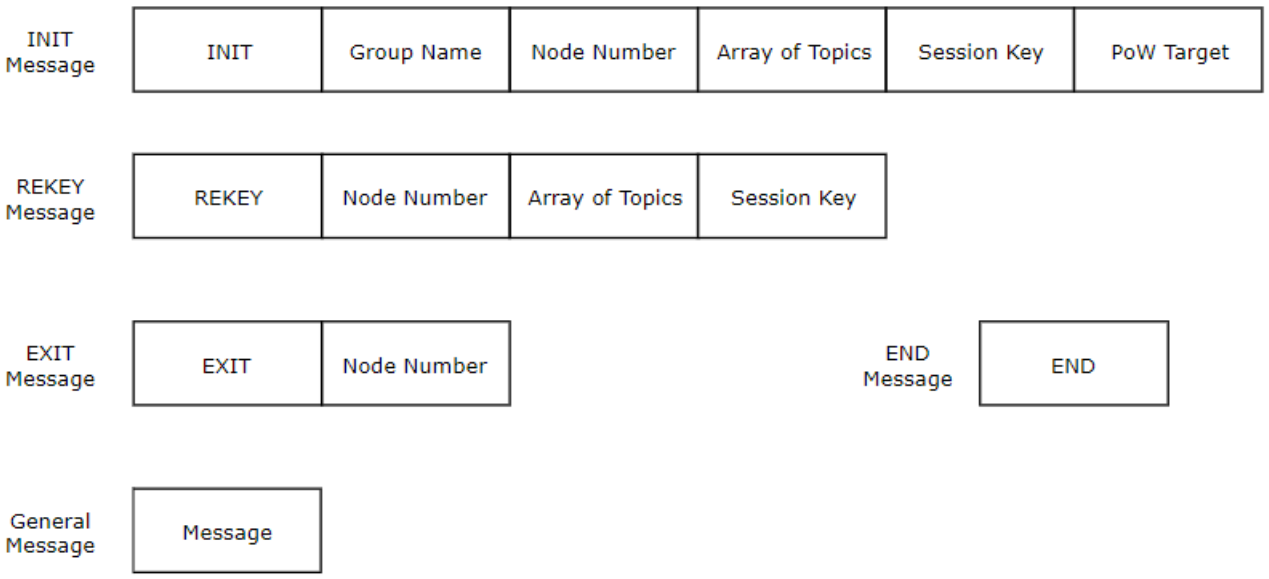

FIGURE 3.1: Message Structure

The node number field is used to inform user's application of their place or their index in the array of topics, this is their assigned topic. The PoW target is the PoW value all envelopes in the group must achieve and it will be used in creating the filter. A group name must be provided when creating a group to act as a human readable identifier for the group. Other messaging applications default their group names to the names of their members but this is obviously not possible if the aim is to remain anonymous. The sequence number in the general messages was intended to be used for general messaging sequencing so that the system can ensure it has received the correct messages and in order.

The INIT, REKEY and END messages are only sent by the group controller. An END message can be sent individually to a removed member or to the group if the group channel has ended. The EXIT message can only be sent by a member of the group and signifies their intention to leave the group. The node number is included so the group controller is aware which contact has left and which topic to ignore when sending the subsequent REKEY message. Finally general messages can be sent by any member of the group.

Message sequencing was implemented but abandoned at a later stage of development. Originally the sequence number would allow a member of the group to request certain messages from another member. However it became clear that this would violate backward secrecy since a user could request messages with sequence numbers before they were added and the group members are not aware of the membership time line. Even

if this issue is avoided, there still exists an issue with the information leaked from the sequence number itself. A newly added member could become aware that a certain number of messages were sent before they were added.

The envelope object returned by a node after polling for a particular filter has the following fields:

- Hash of enveloped message
- Payload (decrypted)
- PoW value
- Timestamp of message generation
- Topic
- TTL
- Signature - public key of sender who signed the message
- Recipients public key

Neither the signature or recipients public key fields are used in the system as the group communication used symmetric session keys. But the signature could be used as a future extension of the system in order to verify you're communicating with a particular user.

3.2.3 Message Sequence

The sequence in which the different types of messages are sent and the subsequent actions taken by the application will now be explained through a series of message sequence diagrams.

Figure 3.2 shows the sequence of an INIT message, which is sent to all members of the newly created group. Before creating the group User A has added User B and User C as contacts using their base identities. The diagram initially shows User B and User C creating a filter for their base identities. User A creates a group named "ChatDemo" with the default PoW value (0.2), these were provided as parameters. Initially User A will generate the session data, the session key and the topics. An INIT message is composed as detailed in the previous section, which is sent through the post Whisper API call. In this single function, the message is encrypted with the provided public key and added to an envelope with the provided topic. This envelope is sealed by finding a nonce which achieves the minimum PoW value provided. Once this is complete the

envelope is sent into the network. User B and User C receive an envelope with their base identity's topic which is successfully decrypted with their base identity's private key. The application is able to recognise the message as an INIT message, it parses the group details and stores it in the applications memory as a mapping between the group name and the group details. They use the *nodeNo* field to find their assigned topic and create a new filter with that topic, the group session key and the minimum group PoW value. Now the group channel has been established and members will use those details to communicate securely.

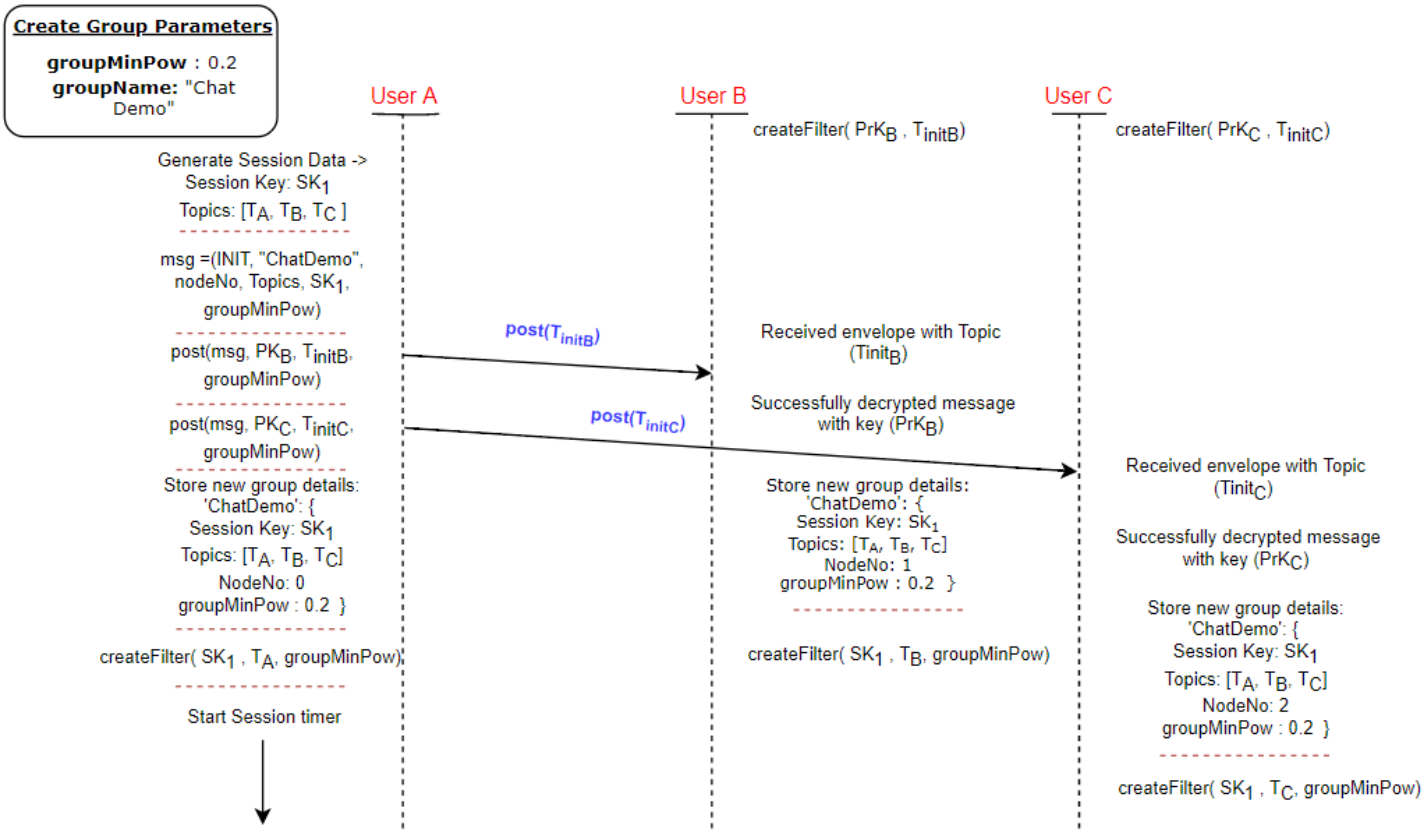

FIGURE 3.2: INIT Sequence Diagram

Figure 3.3 shows the sequence of a REKEY message when a session times out. This can also be sent after membership changes in the group. The sequence follows a very similar pattern to an INIT message, where new session data is generated and sent to the group members. However the message is encrypted with the group session key and sent to all of the group's topics (excluding the group controller). When a REKEY message is received the mapping between the group name and the group details is updated, an example of this is shown in Figure 3.4.

The application always uses the details found in storage when sending a new message and therefore for efficiency and darkness the previous filter and app details should be deleted. If the session time is short then an application could end up with many filters, this would increase the amount of envelopes the node will receive and attempt to decrypt. Scenarios need to be considered where a group member has not received the REKEY message yet but other members have and then that member sends a message using the expired details. If the session is cleared immediately, this message could be missed by other group members. To solve this issue there's a period after the session has expired before it is cleared, this is also shown in Figure 3.4. This ensures that all group members have had a reasonable amount of time to update their group details too.

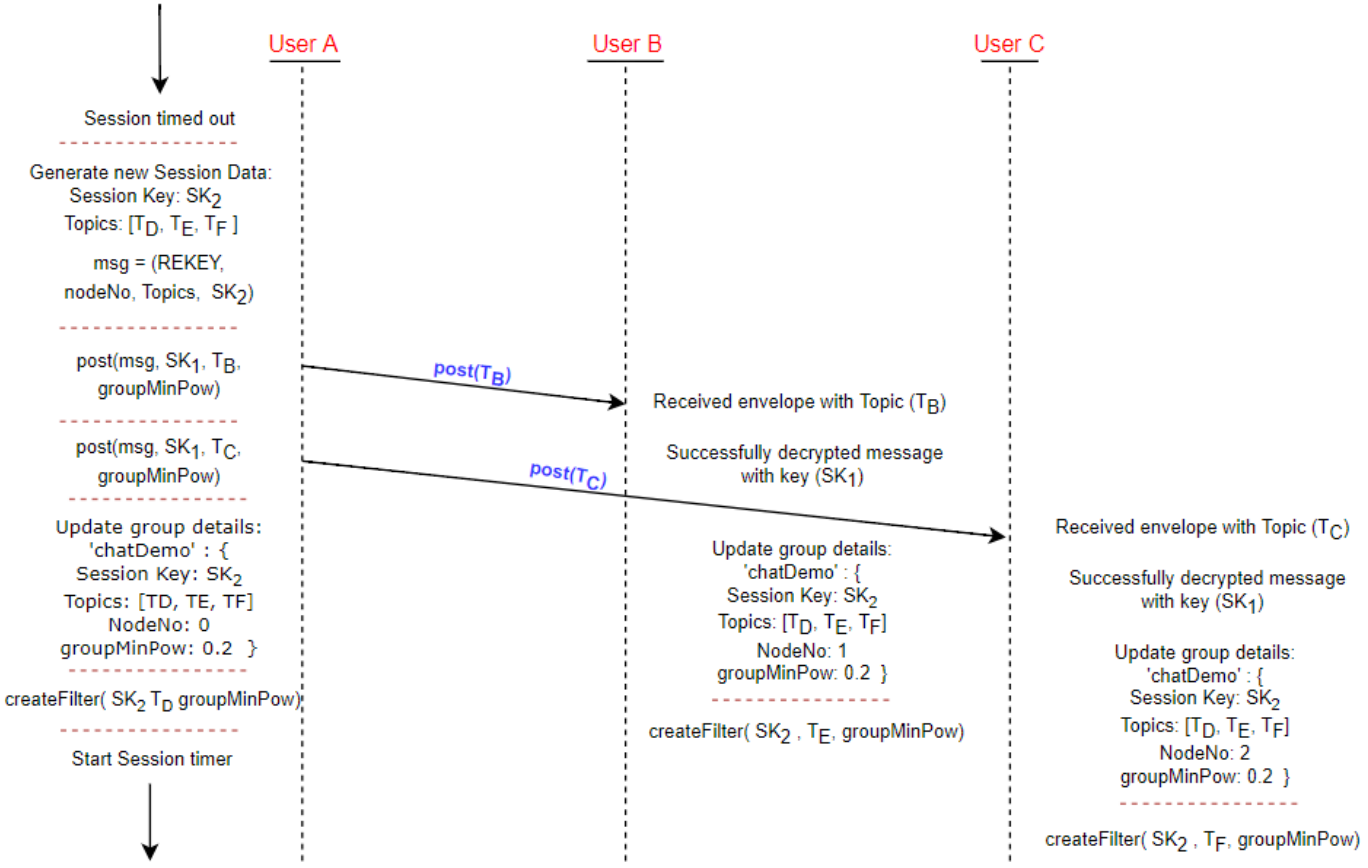

FIGURE 3.3: REKEY Sequence Diagram

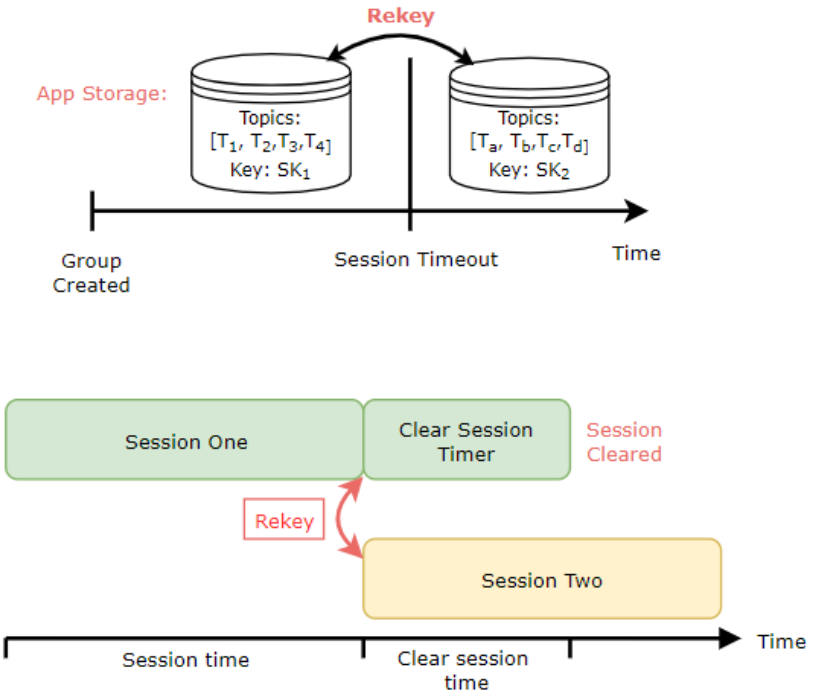

FIGURE 3.4: REKEY Timeline Diagram

Figure 3.5 shows the sequence of a general message sent in a group channel. The current group details are shown in the application storage. The message simply consists of the message composed by the user. It is encrypted using the session key and sent to every topic in the group details. When the message is received and decrypted it is added to an array of messages for that group channel.

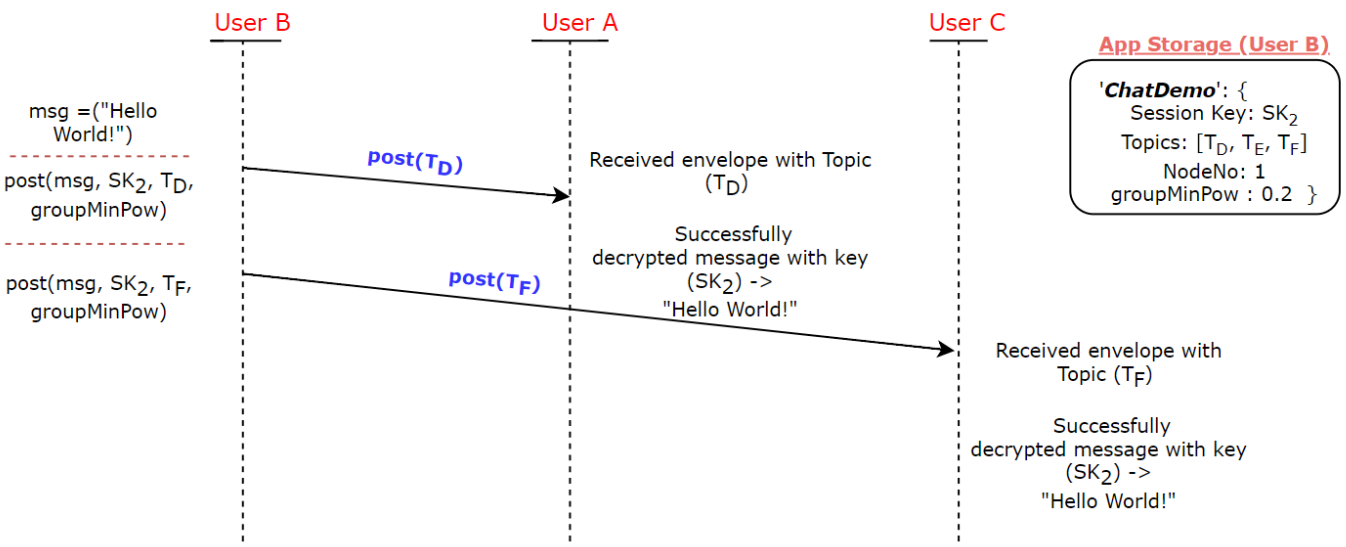

FIGURE 3.5: General Message Sequence Diagram

3.2.4 Spam Prevention

As shown previously a minimum PoW target value is specified in the group details, this value is chosen when the group is initially created. This means that all group communication must reach that PoW target before sending the envelopes in order to ensure their messages are received. But this can only be used to prevent spam within groups. Since a user's base identity may be shared publicly, there's a higher probability that this will become a target for spam. To prevent this a user can specify a PoW value for their base identity higher than the Whisper default of 0.2, in the application this is available as a slider so entirely user configurable. That user could share this value along with their other details so if another user wanted to communicate with them they're aware of this PoW target. By setting their base identity PoW to a higher value such as 50, they can be assured that the original sender has spent significant resources in order for their message to be delivered. This means the user has the ability to choose the types of messages they receive, since more general messages such as news, adverts etc would have lower PoW values. In contrast, more important, personal messages such as direct or INIT messages can be expected to have higher PoW values.

3.3 Implementation

The purpose of this section is to explain how I went about implementing the design previously stated and the technologies described in the *State of the Art* chapter, particularly the Whisper protocol.

3.3.1 Application Architecture

Figure 3.6 shows the application architecture and how the individual components interact. The user interacts with the application using a user interface in the browser. This interfaces uses a locally deployed Node.js application which handles all of the interface's requests. The Node.js application communicates with the Whisper Node using Web3, a JavaScript framework. A locally running Whisper node is expected to expose an IPC connection for communication. This Whisper node is essentially an Ethereum client with the Whisper protocol turned on. The Ethereum client used is *Geth* or Go Ethereum. The Whisper node interacts with it's peers via the DEVp2p protocol and in turn the whole Whisper Network.

Also shown in the architecture diagram the main components of the application run locally meaning the only data leaving the device is the envelopes from the Whisper Node. Running these components locally is more secure and it eliminates the need for trust with a website or server on the user's part, which may be hacked or leak their stored details without their knowledge.

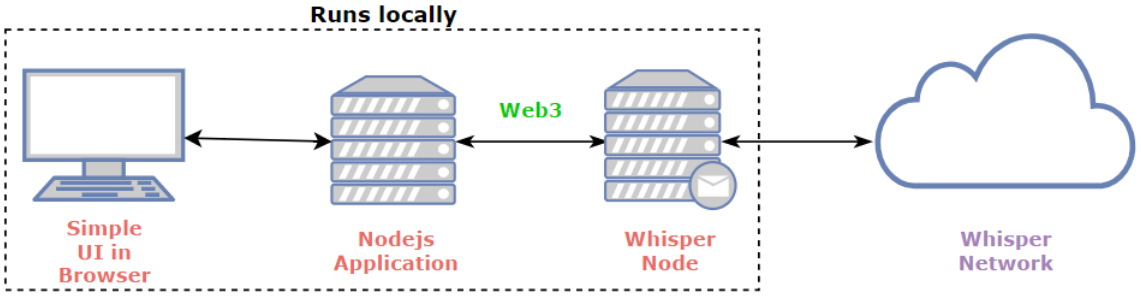

FIGURE 3.6: Application Architecture

3.3.1.1 Node.js

Node.js [55] is a cross-platform Javascript run-time environment which executes server-side. It is lightweight and efficient to run. It also has extensive documentation and a large package ecosystem, named npm, which makes development much simpler. It was

used for the application because it could run on various platforms and I required an efficient application since it is expected to be always on.

By design the application's data is volatile, meaning all the data is cleared once the application is stopped. This design choice was made because users should not have to trust third party applications such as a database with their private messages and key/topic data. If this was stored in a database then that database could become the target of an attack since it could be considered an easier way to expose an identity and retrieve the messages as opposed to traffic analysis.

The Node.js application acts as a simple web server where the user interface (accessed in the browser) sends the requests. This is intended to be run locally too, all the data is stored locally and is for that specific user therefore there's no reason to deploy or host it.

Numerous of the application's action are time based meaning that many timers were needed in the implementation. Node.js has a Timers package which was used for the application, it allows a specified function to be executed after a certain amount of time. The Crypto package was also used to randomly generate the topics.

3.3.1.2 Web3 Framework

Web3 [56] is a JavaScript framework which implements the JSON Remote Procedure Call (RPC) [57] standard used by Ethereum clients. It enables a JavaScript application to interact with an Ethereum node using Websocket or IPC connections. An IPC connection was chosen as it's considered more secure, Websockets expose a port which could be used by another application to gain access or to change a node's details. The framework provides JavaScript functions which act as wrappers to the equivalent Whisper API functions. The framework was used as it's makes development of Ethereum-based applications much simpler.

3.3.1.3 Geth

Geth [50] is an Ethereum client written in the Go programming language which can be run in the command line. This application requires an Ethereum client with the Whisper protocol turned on. Geth was chosen since it is the official and most widely adopted implementation of Ethereum. The client and it's developers are part of the Ethereum Foundation [58], a council of key influences in Ethereum with the mission to promote, support and develop the Ethereum platform. The current development of the Whisper protocol is for Geth and therefore it is also the first client to receive updates.

3.3.2 Application Interface

A simple user interface (UI) was developed using HTML and the Bootstrap framework. The application has three main screens and is modular to allow for future changes. First users are greeted with the login screen, users can choose between providing their base identity details (private key and topic) or having it randomly generated for them.

The homepage shown in Figure 3.7 is the centre of all actions for the application, a user can create a group or add a contact, they can view their node information and direct messages or open up a particular group channel. They can also change the minimum PoW value using the slider.

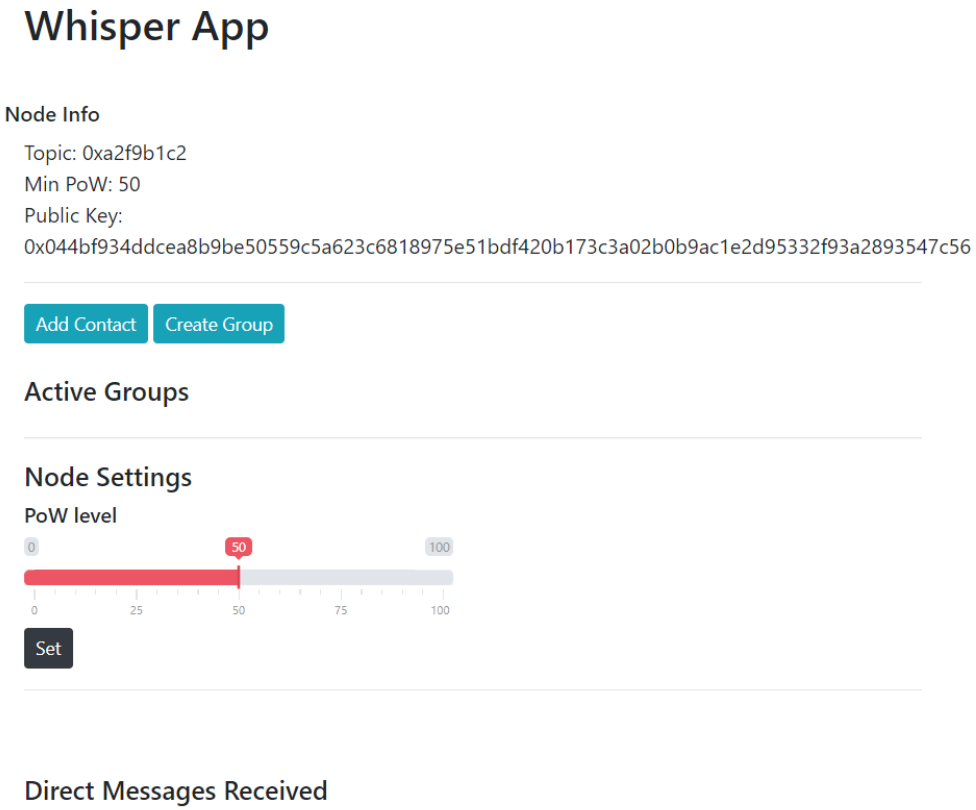

FIGURE 3.7: Homepage UI

Finally there's a group channel interface which is different for the group controller, they have unique actions to add or remove members and to end the group channel. All group members can view the group's information and the received messages. They also have the option to write a message or send a file.

Chapter 4

Conclusions

This chapter will include my reflections on what the project achieved. Starting with the application that was delivered and highlighting potential use cases. The difficulties faced and limitations of the application are documented next. A section on future work is included to summarise how the application could be improved and possible future developments. Finally the report closes with some final thoughts on the project.

Reviewing the original goals stated at the start of the Solution Design chapter, the project can be deemed a success. The delivered application implements all of the features required. Also the minor goal of analysing the Whisper protocol was partially completed, the Whisper protocol was successfully implemented and it was fully functional. However as will be explained in more detail in the following sections, issues were encountered that severely restricted the amount of testing done on the operation of the protocol.

4.1 Use Cases

This section will detail specific use cases where the application could provide a solution. It's not intended as a direct replacement for current messaging applications but it could potentially be used as one. The application has a familiar UI and its operation is similar. Ideally the application should be used for scenarios requiring anonymity and strong security. The inefficiency introduced to maintain anonymity is significant and therefore I would only recommend the application in these scenarios.

Anonymity has always been a key element in journalism. Sources of information can wish to keep their identity private since the information they're providing could be sensitive, confidential or even illegal. The severe consequences of being uncovered acts as a major deterrent to possible sources. If a journalist used the application and publicly

advertised their base identity, sources could make initial contact and remain completely anonymous. They can also continue their conversation in a group channel over a longer period of time. This use case could also be augmented to include whistle blowers or witness protection instead. These are situations where communication is necessary but could put a person(s) at risk, the application will negate this risk by ensuring they remain anonymous while communicating through the system.

The application could also be used to overcome censorship. Group channels could be set up to share news and ideas freely. Since the communication is relayed over the DEVp2p network, the application could also be more resilient to suppression. Generally, internet services are simply blocked by their IP addresses or at the source but this would not be viable. Importantly this could regain freedom of speech in areas where not previously possible.

One negative use case is for crime whether this be illegal material or to facilitate communication leading to criminal activity. It could be used to evade surveillance which potentially would have prevented the crime from occurring. This is a valid global security concern and technology like this is generally strongly opposed by Governments. In my opinion this is a trade-off between the positive and negative effects of the application and ultimately it falls to the user and how they choose to utilise this new-found power over their identity.

4.2 Difficulties

There were several difficulties faced throughout this project, the majority stemmed from the fact that the Whisper protocol is still relatively new and in constant development.

Firstly, we struggled with the lack of documentation, Github and YouTube (for live demos and presentations) were the sole sources of information on the protocol. This caused issues with developing the application but mainly with writing this report since there was limited views and interpretations. Also, much of the available documentation was written on earlier versions of the protocol and therefore there was a constant issue of verifying it was still correct. On numerous occasions we had to resort to the latest source code, which was difficult to comprehend as it was written in an unfamiliar language (Go) and quite complex. However, we were able to raise queries to the Whisper Gitter channel and got helpful replies from other developers using the protocol.

The lack of documentations also meant a lack of examples too. Examples and code samples that exist were either very simple (send a single message and receive it) or they just did not work due to inconsistencies with newer API versions. Again we had

to analyse the source code regularly to grasp the protocol's operation. There was also issues setting up and connecting to my Geth node but this was due to inexperience. Both of these meant that the initial setup of the application took significantly more time than anticipated.

The application has a critical dependency on the Web3 framework as this provides the JavaScript methods which can communicate with the Whisper API on the local node. At the time of development there were certain features of Whisper that could not be used as Web3 did not support them such as bloom filters. This will be the case in the future too since Web3 updates will always take some time to implement following a Whisper update. But currently there's no alternative framework to Web3 for Whisper therefore it would require implementing the wrapper functions to communicate over JSON-RPC which was not feasible in the time frame.

A major difficulty was simply using and testing my application because currently Whisper is not functional on any of the Ethereum networks in use such as Mainnet and Rinkeby. This stems from Ethereum nodes being required explicitly turn the protocol on start up. Currently there's no incentive for Ethereum nodes and their users to turn Whisper on and therefore there's a very small number of nodes in these networks that support it. Due to this the messages do not propagate through the network at all, most of the time the message will be simply ignored by your peers as they do not support it. Almost all the nodes in these networks are running specifically for blockchain capabilities and to mine. This is already requires a high amount of resources and turning on Whisper would only increase this load.

4.3 Limitations

This section will detail limitations with the developed application.

Arguably the biggest limitation is the application must be on and connected to a local node for a user to receive messages. If they turn it off or disconnect they will not receive any messages while offline. Node failure could occur for a number of reasons such as loss of internet connection and the application has no system to recover from this. Meaning if a user's node fails they run the risk of not receiving envelopes meant for them. Considering an envelope's TTL is generally in the range of 5-30 seconds, the probability of envelope loss is high. This could result in a user missing crucial group channel information such as an INIT or REKEY message.

The centralised group controller brings some limitations too. If the group controller's node fails, then the group could potentially fail and if they decide to leave then the

group will end immediately. This is could be an issue for other users because they might be having a conversation which is cut short and they can not recreate the group channel as by default they do not know the identities of the other members.

The testing of the application has been very limited due to the issue with the networks, this an issue the protocol itself has encountered. To test the application a private Ethereum network was set up where multiple Whisper nodes were connected and run on a single machine. In this network the application worked as expected and communication was successful. However, this network was unable to recreate scenarios you would expect to see in an actual Whisper network such as a heavy traffic and envelope loss. As a result, it's not known how much resources the application will require or how inefficient it truly is. Ultimately, it's still unknown how the application and protocol itself will handle an actual network and if it will be able to scale to the extent of current applications with user bases of millions and even billions.

4.4 Future Work

This section will detail the improvements and additions we would make to the application given more time. Also, to discuss how future developments could impact the application.

As mentioned previously the Whisper protocol is still in constant development and a significant version 6 update is upcoming. One of the useful additions is a mail server feature which will allow a node to designate a trusted peer to store envelopes on their behalf. The node can then poll the peer for these envelopes and they will be resent. This means even when offline a user could still receive messages, this would introduce email-like functionality. This could potentially solve the limitations of the application when offline too.

At the time of writing, there's currently a solution being tested to resolve the Whisper adoption issue on Ethereum networks. This solution involves migrating the protocol to a different P2P networking stack (from DEVp2p). The current replacement being tested is libp2p [59], this is the network stack used by IPFS. libp2p's implementation is modular and far more flexible than DEVp2p and therefore could be a viable solution. DEVp2p was developed specifically for Ethereum before libp2p existed and their sole interest at that time was to build a working system for the blockchain fast. Therefore, later additions to Ethereum such as Whisper have had issues with the stack.

If the migration is a success and Whisper can be run on a larger network, more complete testing would become possible. As previously mentioned we were unable to test how the application would scale, but this could remove those difficulties. We would plan to carry

out qualitative testing on the traffic levels in the network, latency of messages and the efficiency of the application itself. It would be worth investigating how consistent the network is too, in terms of envelope delivery. Also, this would allow more substantial and realistic mock attacks on the application. Increased numbers of nodes and envelopes would provide the data for an attempt at traffic analysis.

The amount of user configuration in the application could be improved too. Users could be allowed to specify settings such as the session timeout time. All these additional options would give the user more control over the trade-off between darkness and efficiency.

Status [54] is a Web 3.0 company which is also using the Whisper protocol. They're currently building a mobile application which runs an Ethereum node implementation, giving access to the full Ethereum ecosystem from anywhere. Interestingly they believe that future applications will be built around messaging rather than a traditional browser. By adding more functionality to the application by integrating other Dapps it could potentially make it more attractive to potential users.

4.5 Final Thoughts

Overall, we're happy with the outcome of the project. It delivers all the features we set out to achieve and we truly believe this application or a similar application could provide a completely new form of messaging. Although we do not feel the anonymity aspect was adequately tested and therefore can not be guaranteed.

With any project of this scale you have to reflect on whether it was worth the time invested. In our opinion it was worth the investment, we were able to test this new protocol and developed a unique solution using it. The application itself could be used in the future as an example or as a starting point for another application. We also believe the report has established the importance of secrecy and why we should look to develop applications which give users back control over their data and identity. Of course, secrecy is and always will be a security concern and whether this application will have an overall negative effect on global security is not known.

Another unanswered question is whether it will actually attract any users and if the loss in efficiency is worth it. In our opinion the attributes of the application alone will attract users. There exists a niche in the market which until now has not been filled, the use cases are specific but they're large in scale. Fields like journalism and law enforcement are open to new technologies which could aid them and if they choose to adopt this platform they would bring a large user base. Since this is one of the only existing

solutions to anonymous communication, the inefficiency will be worth it for users who genuinely require it. Efficiency can be gained from a decrease in darkness and users will have control over that trade-off. However as previously mentioned, we do not envision this application as a direct replacement to current messaging applications rather filling the niche of anonymous communication.

Appendix A

CD

The entire source code for the application and it's dependencies can be found on the CD included with this report. This source code can also be found on Github at github.com/SeanDurban/FYP_App with the Tag **'V1.0'**.

Bibliography

[1] *Web3 Foundation*. Accessed on 22-04-2018. URL: https://web3.foundation/.

[2] Ewen MacAskill Glenn Greenwald. "Boundless Informant: the NSA's secret tool to track global surveillance data". In: (June 2013). Accessed on 22-04-2018. URL: https://www.theguardian.com/world/2013/jun/08/nsa-boundless-informant-global-datamining.

[3] Holger Stark Laura Poitras Marcel Rosenbach. "GCHQ and NSA Targeted Private German Companies and Merkel". In: (Mar. 2014). Accessed on 22-04-2018. URL: http://www.spiegel.de/international/germany/gchq-and-nsa-targeted-private-german-companies-a-961444.html.

[4] Olivia Solon. "WhatsApp rejected Government request to access encrypted messages". In: (Apr. 2018). Accessed on 22-04-2018. URL: https://www.theguardian.com/technology/2018/apr/04/facebook-cambridge-analytica-user-data-latest-more-than-thought.

[5] George Danezis Steven J. Murdoch. "Low-Cost Traffic Analysis of Tor". In: (2005), pp. 183–195.

[6] Neil MacFarquhar. "Russian Court Bans Telegram App After 18-Minute Hearing". In: (Apr. 2018). Accessed on 23-04-2018. URL: https://www.nytimes.com/2018/04/13/world/europe/russia-telegram-encryption.html.

[7] Ingrid Lunden. "Russiafffdfffdfffds game of Telegram whack-a-mole grows to 19M blocked IPs, hitting Twitch, Spotify and more". In: (Apr. 2018). Accessed on 23-04-2018. URL: https://techcrunch.com/2018/04/19/russias-game-of-telegram-whack-a-mole-grows-to-19m-blocked-ips-hitting-twitch-spotify-and-more.

[8] *Signal*. Accessed on 23-04-2018. URL: https://signal.org/.

[9] "Electronic Frontier Foundation - Surveillance Self-Defence". In: (). Accessed on 23-04-2018. URL: https://ssd.eff.org/en/module/communicating-others.

[10] B. Morton and C. Smith. "Why We Need to Move to SHA-2". In: (Jan. 2014). URL: https://casecurity.org/2014/01/30/why-we-need-to-move-to-sha-2/.

[11] M.Jakobsson and A.Juels. "Proofs of Work and Bread Pudding Protocols". In: *CMS '99 Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security* (Sept. 1999).

[12] Chris Pacia. "Bitcoin Mining Explained Like Youfffdfffdfffdre Five: Part 2 fffdfffdfffd Mechanics". In: (Sept. 2013). Accessed on 24-04-2018. URL: https://chrispacia.wordpress.com/2013/09/02/bitcoin-mining-explained-like-youre-five-part-2-mechanics/.

[13] R[onald] L. Rivest, A[di] Shamir, and L[eonard M.] Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *CACM* 21.2 (Feb. 1978), pp. 120–126.

[14] F. Bahr et al. "We have factored RS200 by GNFS". In: (May 2005). Accessed on 02-03-2018. URL: http://www.crypto-world.com/announcements/rsa200.txt.

[15] Quynh H. Dang Elaine B. Barker. "Recommendation for Key Management Part 3: Application-Specific Key Management Guidance". In: *Special Publication (NIST SP) - 800-57 Pt3 Rev 1* (Jan. 2015). URL: https://www.nist.gov/publications/recommendation-key-management-part-3-application-specific-key-management-guidance.

[16] Bitcoin. "Android Security Vulnerability". In: (Aug. 2013). URL: https://bitcoin.org/en/alert/2013-08-11-android.

[17] *DHT Diagram*. Accessed on 30-03-2018. URL: http://cse.csusb.edu/tongyu/courses/cs660/notes/distarch.php.

[18] David Mazieres Petar Maymounkov. "Kademlia: A Peer-to-peer Information System Based on the XOR Metric". In: (Oct. 2002).

[19] Van Jacobson et al. "Networking named content". In: (2009), pp. 1–12.

[20] *CNN Diagram*. Accessed on 30-03-2018. URL: http://networking.khu.ac.kr/layouts/net/research/res11.htm.

[21] *Public Certificate Diagram*. Accessed on 31-03-2018. URL: https://commons.wikimedia.org/wiki/File:PublicKeyCertificateDiagram_It.svg.

[22] Fox IT. *DigiNotar Certificate Authority breach fffdfffdfffdOperation Black Tulipfffdfffdfffd*. Accessed on 12-03-2018. Sept. 2011. URL: https://www.rijksoverheid.nl/ministeries/ministerie-van-binnenlandse-zaken-en-koninkrijksrelaties/documenten/rapporten/2011/09/05/diginotar-public-report-version-1.

[23] Johnathan Nightingale. *DigiNotar Removal Follow Up*. Accessed on 12-03-2018. Sept. 2011. URL: https://blog.mozilla.org/security/2011/09/02/diginotar-removal-follow-up.

[24] W3Techs. "Usage of SSL certificate authorities for websites". In: (Nov. 2017). URL: https://w3techs.com/technologies/overview/ssl_certificate/all.

[25] Phil Zimmermann. "PGP version 2.0 manual". In: (1992).

[26] *MITM Attack Diagram*. Accessed on 31-03-2018. URL: https://www.teskalabs.com/blog/protect-mobile-app-and-prevent-man-in-the-middle-attack.

[27] Ali Salman. "WhatsApp Hits Another Milestone fffdfffdfffd 1.5 Billion Monthly Active Users In Q4 Of 2017". In: (Feb. 2018). Accessed on 31-03-2018. URL: https://wccftech.com/whatsapp-hits-another-milestone-1-5-billion-monthly-active-users-in-q4-of-2017/.

[28] Tom Cheshire. "WhatsApp rejected Government request to access encrypted messages". In: (Sept. 2017). Accessed on 31-03-2018. URL: https://news.sky.com/story/whatsapp-denies-government-access-to-encrypted-messages-11043069.

[29] Kate Conger. "WhatsApp blocked in Brazil again". In: (July 2016). Accessed on 31-03-2018. URL: https://techcrunch.com/2016/07/19/whatsapp-blocked-in-brazil-again.

[30] Jorg Schwenk Paul Rosler Christian Mainka. "More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema". In: *3rd IEEE European Symposium on Security and Privacy (EuroSP 2018)* (Jan. 2018). Accessed on 27-03-2018.

[31] Nick Mathewson Roger Dingledine. *Tor Protocol Specification*. Accessed on 30-03-2018. URL: https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt.

[32] *Tor Project: About Tor*. Accessed on 30-03-2018. URL: https://www.torproject.org/about.

[33] *Tor Network Status*. Accessed on 30-03-2018. URL: http://torstatus.blutmagie.de/.

[34] *Onion Routing Diagram*. Accessed on 31-03-2018. URL: https://1technation.com/tech-savvy-dark-side-onion-router.

[35] *Hidden Services Diagram*. Accessed on 31-03-2018. URL: https://www.torproject.org/docs/onion-services.html.en.

[36] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: (Nov. 2008). URL: https://bitcoin.org/bitcoin.pdf.

[37] *Coin Market Cap*. Apr. 2018. URL: https://coinmarketcap.com/.

[38] Virgil Griffith Vitalik Buterin. "Casper the Friendly Finality Gadget". In: (Nov. 2017). URL: https://arxiv.org/abs/1710.09437v2.

[39] Vitalik Buterin. *A Next Generation Smart Contract and Decentralized Application Platform*. 2013. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[40] Vitalik Buterin. "Ethereum: Now Going Public". In: (Jan. 2014). URL: https://blog.ethereum.org/2014/01/23/ethereum-now-going-public/.

[41] *Ethereum Homestead Documentation - Contracts and Transactions*. URL: http://www.ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html#what-is-a-transaction.

[42] Karthikeyan Bhargavan et al. "Formal verification of smart contracts: Short paper". In: (2016), pp. 91–96.

[43] Michael del Castillo. "Ethereum Executes Blockchain Hard Fork to Return DAO Funds". In: (July 2016). URL: https://www.coindesk.com/ethereum-executes-blockchain-hard-fork-return-dao-investor-funds/.

[44] Tiziana Cimoli Nicola Atzei Massimo Bartoletti. "A Survey of Attacks on Ethereum Smart Contracts". In: (Apr. 2017). URL: https://eprint.iacr.org/2016/1007.pdf.

[45] *DfffdfffdVp2p Wire Protocol*. Accessed on 02-04-2018. URL: https://github.com/ethereum/wiki/wiki/%C3%90%CE%9EVp2p-Wire-Protocol.

[46] *RLPx Specification*. Accessed on 02-04-2018. URL: https://github.com/ethereum/devp2p/blob/master/rlpx.md.

[47] *Node Discovery Protocol v4*. Accessed on 02-04-2018. URL: https://github.com/ethereum/devp2p/blob/master/discv4.md.

[48] *Whisper PoC 2 Protocol Spec*. Accessed on 16-03-2018. URL: https://github.com/ethereum/wiki/wiki/Whisper-PoC-2-Protocol-Spec.

[49] *Whisper Overview*. Accessed on 16-03-2018. URL: https://github.com/ethereum/go-ethereum/wiki/Whisper-Overview.

[50] *Go Ethereum*. Accessed on 22-04-2018. URL: https://github.com/ethereum/go-ethereum/.

[51] *Whisper Network Routing Diagram*. Accessed on 05-04-2018. URL: https://mainframe.com/protocol/.

[52] Dr Gavin Youk. "DApps: What Web 3.0 looks like". In: (Apr. 2014). URL: http://gavwood.com/dappsweb3.html.

[53] Juan Benet. "IPFS - Content Addressed, Versioned, P2P File System". In: (July 2014). URL: https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf.

[54] *The Status Network- A strategy towards mass adoption of Ethereum*. June 2017. URL: https://status.im/whitepaper.pdf.

[55] *Node.js*. Accessed on 22-04-2018. URL: https://nodejs.org.

[56] *Web3 Framework*. Accessed on 22-04-2018. URL: http://web3js.readthedocs.io/en/1.0/.

[57] *JSON Remote Procedure Call*. Accessed on 22-04-2018. URL: https://github.com/ethereum/wiki/wiki/JSON-RPC.

[58] *Ethereum Foundation*. Accessed on 22-04-2018. URL: https://ethereum.org/foundation.

[59] *libp2p*. Accessed on 25-04-2018. URL: https://libp2p.io/.