Fall 2022

Enabling Use of Signal in a Disconnected Village Environment

Evan Chopra

Enabling Use of Signal in a Disconnected Village Environment

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Evan Chopra

December 2022

The Designated Project Committee Approves the Project Titled

Enabling Use of Signal in a Disconnected Village Environment

by

Evan Chopra

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

December 2022

Dr. Ben Reed	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Sid Anand	Masters of Computer Science, Distributed Systems

ABSTRACT

Enabling Use of Signal in a Disconnected Village Environment

by Evan Chopra

A significant portion of the world still does not have a stable internet connection. Those people should have the ability to communicate with their loved ones who may not live near by or to share ideas with friends. To power this achievable reality, our lab has set out on making infrastructure for enabling delay tolerant applications. This network will communicate using existing smartphones that will relay the information to a connected environment. The proof of concept application our lab is using is Signal as it offers end to end encryption messaging and an open source platform our lab can develop.

ACKNOWLEDGMENTS

I would like to express my thanks to Dr. Ben Reed for his support and knowledge of this topic throughout this project. I would also like to thank him for coming up with this powerful idea for our lab. He has showed me that software has a strong reach and can accomplish things that are innovative and seemingly impossible! I would also like to thank Dr. Robert Chun for supporting my ideas and giving my topic the chance to blossom into something greater. He has been a lot of help writing and perfecting this paper. Lastly, I would like to thank Sid Anand for always being there to support my learning process and always being available for questions!

TABLE OF CONTENTS

CHAPTER

1 Introduction 1

1.1 Our Labs introduction 2

1.2 Significance of Signal 3

2 Background 5

2.1 Signal Application 5

2.2 The Signal Protocol 5

2.2.1 KDF Chains 6

2.2.2 The Double Ratchet Protocol 8

3 Related Works 11

3.1 Mobile Delay Tolerant Networks 11

3.2 Remote DTNs with bundling abilities 12

3.3 DTNs over the Metropolitan Public Transportation 12

3.4 WaterChat: A Group Chat Application Based on Opportunistic Mobile Social Networks 14

3.5 Signals Handshake X3DH 15

3.6 PKI in a DTN environment 15

4 System Design 17

4.1 Overarching Design 17

4.2 Scalability 18

5 Design of Signal Plugin 19

5.1 Formalized Design Flow 20
5.2 Challenges of libsignal-java 21
5.3 Registration 22
5.4 Sending the First Message 25
5.5 Sending Messages 26
5.5.1 Exploring Alternate Methods to Sending and Receiving Messages 28
5.5.2 Signal Envelopes 29
5.5.3 Spoofing Prevention 30
5.5.4 Abuse Protection 31
5.5.5 Encrypting & Sealing the Envelope 31
5.6 Client Requirements for Double Ratchet Protocol 31
6 Experiment 33
6.1 Registration 33
6.2 Message Sending and Receiving 34
7 Future Works 36
7.1 Enabling Group Chats 36
7.2 Working Directly with Signal 37
7.3 Expanding Application Compatibility 37
8 Conclusion 39

CHAPTER 1

Introduction

As the world seemingly becomes more and more connected, a large portion of the world's population still lacks a stable internet connection. A report by the United Nations International Children's Emergency Fund concluded that two-thirds of the world's children do not have access to the internet[1]. This lack of available connections can restrict a child's ability to learn about various topics that are not readily known in their community. This lack of internet can also inhibit economic viability with everything moving to e-commerce. The ability to have some internet connection is extremely important in the modern day.

In an area where the internet is not reliable or available, there are solutions that can enable users to have the ability to use the internet in a delayed fashion. This ability is built off the backbone of a delay-tolerant network paradigm. Delay tolerant networks(DTNs) typically follow an opportunistic network paradigm. This paradigm means that forwarding messages usually take advantage of devices proximity to one another and the devices willingness to accept a message from another. The end goal of these networks is that eventually a packet will make its way to its destination through device jumping. In this case, an opportunistic network paradigm requires that each user's device uses a store and forward model. As users request more information from various websites or attempt to send messages on different platforms, their requests are bundled up and sent as a client becomes available to take it. In general, these requests can consist of packets from any application such as Whatsapp or YouTube. An opportunistic network does not care what packets are being sent, just its ability to forward them in close proximity to other devices. To enable a remote village or an internet-less area to connect to the outside world, an efficient DTN needs to be created that can utilize reliable transportation of packets to connected areas. Doing

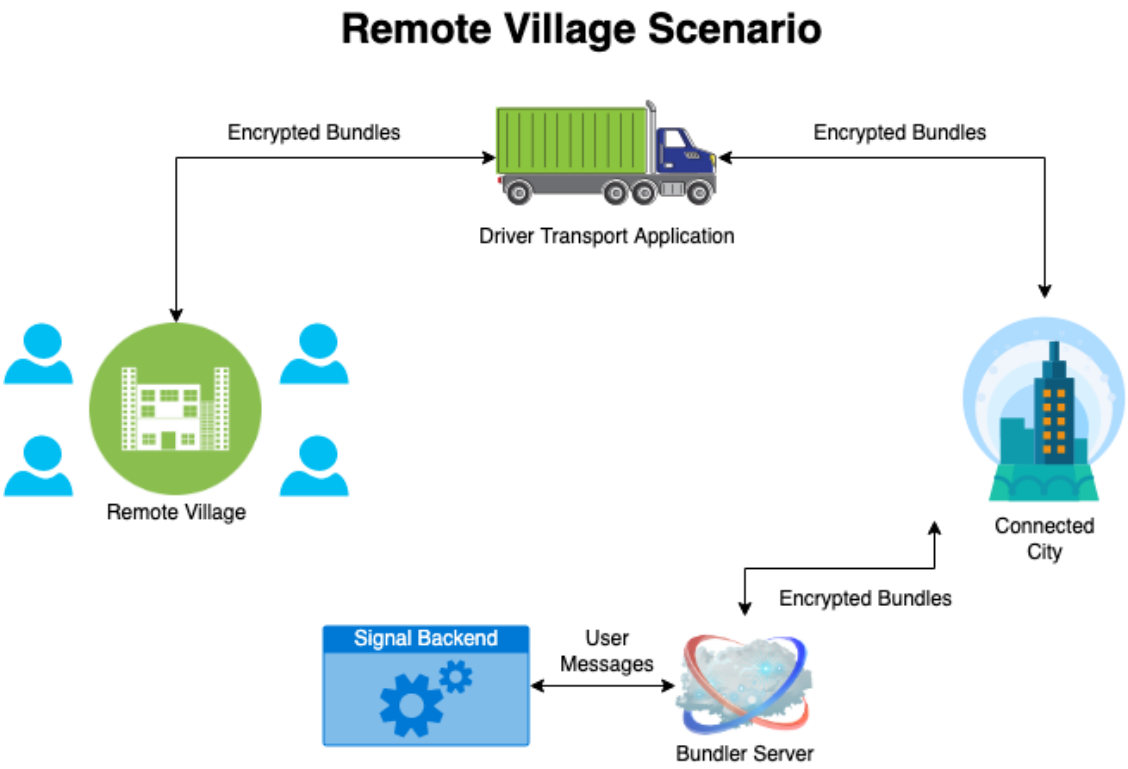


Figure 1

this successfully and making it accessible to the masses allows those who are normally disconnected to be plugged into the rest of the world.

1.1 Our Labs introduction

Typically DTN remote village research usually focuses on implementing custom hardware that acts as the infrastructure that carries information from the disconnected village to the connected world. In our study, we are simplifying that typical custom infrastructure by moving everything to personal smartphone devices. Normally, this custom hardware needs to be purchased or maintained with software updates in order to remain functional. By using the operating system built into an individuals phones, our lab can create infrastructure out of everyday devices that need minimal development maintenance and no upfront additional cost. Our lab can leverage the flexibility and built in technology in Android to reduce the development needed to

create such infrastructure. Individuals who frequent the village and a connected environment will be downloading a transporter app that will take all of the bundled information and transfer it to a bundler server that is running and ready to send out individual's requests and accept responses for them. Those in the villages will have a simple client app that will intercept requests going to the various apps that use this store and forward model and zip their requests into a file that can be read by our server that is connected to the internet. The DTN should operate on a zero trust schema, meaning that no one in the network should be able to read zip bundles that are not their own.

1.2 Significance of Signal

In the present digital world many companies seek to increase their revenue through new models of selling consumer data to third party services[2]. This spreading of information puts users privacy at stake. Signal empowers users to stay private and message people they care about without fear of company or government overreach. Signal is an independent nonprofit operated company that purely operates off of donations made by users and larger organizations that believe in their mission[3]. They do not track user information or sell user data, and they prefer to let you handle your own data. They are proudly an open source and have a very active community of developers and Signal protocol driven projects[4]. As of September 2022, they have 100 million downloads[5] and 40 million monthly active users[6]. The cofounder of Whatsapp now runs the operation of Signal after leaving Whatsapp and he stated that he shared, "differences surrounding the user of customer data and targeted advertising"[6]. Signal is a company that stands for privacy and empowers the average user to take back their privacy and data. This is a rare occurrence in a world driven by profit. This privacy comes free of cost allowing this app to also be

extremely accessible for those in underdeveloped areas. Those underdeveloped areas are our target, so that we can enable them to be connected to the rest of the global community.

CHAPTER 2

Background

2.1 Signal Application

The Signal application is the proof of concept application we have chosen for our study as it is widely used and is a secure open source messaging platform. Being open source, it is able to be modified to fit into the store and forward model. When a user has the modified Signal application on their phone, it will act as if it is sending messages normally, however these messages will be bundled into encrypted zip files and sent as Wi-Fi direct connections to other available clients. These encrypted signal zip files are passed to a client that is accepting bundles. With an end to end encryption model, it does not matter who gets these zip bundles. The zip bundles are designed and encrypted in such a way that we are not worried about malicious actors acquiring them. The hope is that one of these clients will eventually get to a connected environment and pass these bundles off to our bundler server.

2.2 The Signal Protocol

The Signal Protocol is what powers the open source encryption that the Signal app executes in order to keep messages unreadable. It is important to have a high level of understanding of how this algorithm works so that certain steps of our bundling and message transmission can be tweaked to keep messages private. The Signal Protocol is an end to end encryption protocol that uses a variation of a public-private cryptography to keep things private between two clients.

The Signal Protocols main algorithms that are used to do the encryption are the Extended Triple Diffie-Hellman (X3DH)[8] and the Double Ratchet algorithm[9].

The X3DH algorithm is designed for asynchronous settings where a user is offline but has published information to a server. A different user who is online can use this published information on the server to send encrypted data to the offline user. In

order to make a client available to receive messages, a client must generate a pair of long term identity key pairs which do not change, a signed pre-key pair, and multiple expendable pre-key pairs[8]. The signed pre-key pair is used in the Diffie-Hellman calculations for verification and decryption. It is also used to verify the opposing client's signature. The pre-key pairs are one time use keys that are used to initiate and encrypt the handshake that is X3DH. Batches of pre-keys are sent up to the Signal servers since these are deleted after one use[8]. Next, the client app bundles all of these keys and a registration ID into a key bundle and sends it up to Signals servers to be added to a key distribution center. This distribution center contains receipts public keys, so that the requesting client can obtain the appropriate information to start a conversation. This key bundle contains not only the long term public key but the rest of the generated keys. The local client then generates their own keys and creates a shared master key. The new master key is sent back to the original client and is validated. Messages can now be sent as they will be encrypted using this master key to start[8].

2.2.1 KDF Chains

The Double Ratchet algorithm allows two parties to send encrypted messages based on a shared secret key. X3DH is used to come to agreement on what secret key will be used. Following this initial secret key, the clients will use the Double Ratchet protocol to send more encrypted messages. The clients use this algorithm so that earlier messages cannot be calculated from later ones.

Inside the protocol, a key derivation function (KDF) chain is a core mechanism that allows for encryption. A KDF is a cryptographic function that takes in a secret and random KDF key and some data and then outputs some data. If the key is not known, the output look random to an outside viewer. If the key is still known, KDF

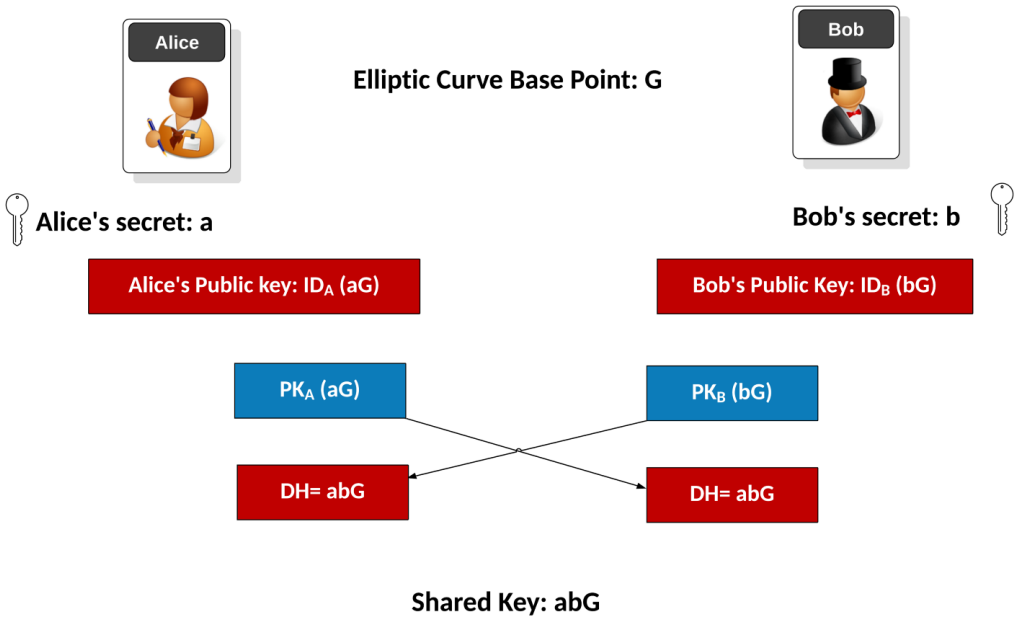


Figure 2: X3DH handshake shown with Elliptic Curve Diffie Hellman[1]

should still provide a secure cryptographic hash of its input data. The reason this concept is described as a chain is because KDF blocks are chained together, using their output as another KDF blocks input. The benefit of using output as input is it guarantees forward security. This mean that output from earlier KDF blocks looks random even if they have obtained the current key and input[9]. This concept also improved resilience as an adversary will see the outputs as random without any knowledge of the KDF keys. Lastly, the KDF chain has the break-in recovery attribute that makes future output keys appear random even if a KDF key was learned at some point in time. This trait relies on the fact that there has been some sort of randomness injected into the chains. Now the double ratchet protocol involves three separate KDF chains all working in unison[3]. The three chains working are named the root chain, sending chain and receiving chain. Lets say Alice and Bob send messages to each other, their Diffe-Hellman public keys and outputs become inputs to the root

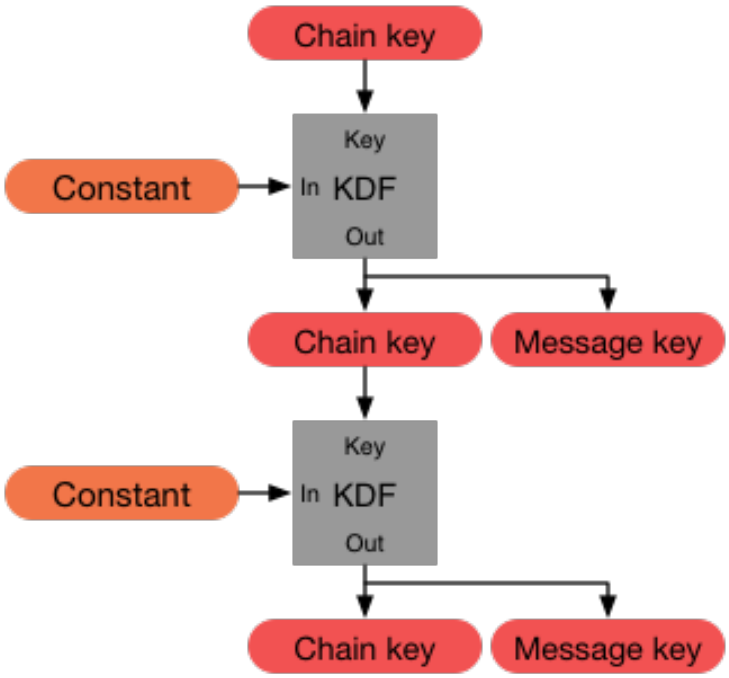


Figure 3: A KDF chain feeding output into the input of the next KDF block [9]

chain. The output keys from this root chain feed into the sending and receiving chains as new input. This is what is called a Diffe-Hellman ratchet.

2.2.2 The Double Ratchet Protocol

To send or receive messages a unique message key must be used. These message keys are generated from the sending and receiving KDF chains that we defined above. Since the external inputs for these chains is constant, they do not benefit from break-in recovery. To generate new message keys, the KDF chains use the output from the previous message key. This is what is called a single ratchet step in a symmetric-key ratchet. The problem with this simpler way of encrypting messages is that if an attacker steals one of the users sending and receiving chain keys, the attacker can

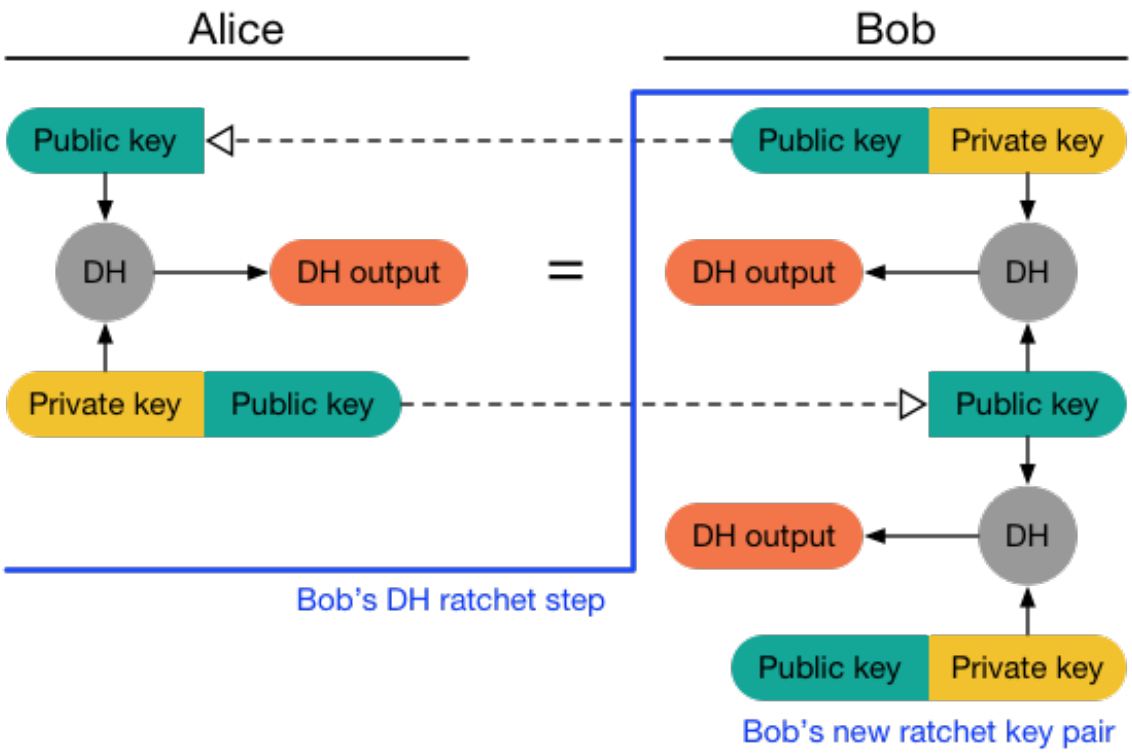


Figure 4: Ratcheting Step [8]

compute all future message keys and decrypt all future messages.[9]

To prevent this very doable attack, the Signal protocol has a counter play to this. In order to make future messages uniquely encrypted, the Double Ratchet protocol implements another step that integrates with the symmetric-key ratchet. This step is known as the Diffie-Hellman ratchet. In this combined algorithm, each party creates a DH (Diffie-Hellman) key pair which consists of a private and public key. For every message that is exchanged the public key is attached in the header. When the recipient receives the senders public key, a DH ratchet step is executed which then replaces the local key with a new key pair. If an adversary gets and compromises the private

key, we can say that this is okay because it will eventually be replaced. There can exist a situation when messages using the double ratchet program can be delivered out of order. To counter this potential problem, each message includes metadata that has the message number and the length of the previous message keys. While waiting for the messages it's missing to be delivered, it can skip to the current message key based on what was given. When a message is delivered, a ratchet step is started and the protocol can figure out how many messages it's missing. To figure this out the protocol can calculate the previous chain's length minus the current receiving chain's length. If the receiving message does not trigger a ratchet step, then the received message number minus the length of the receiving chain is the number of skipped messages[9]. If Alice sends three messages to Bob and only the third message arrives, Bob's ratchet step will trigger on message three instead of message two. Message three will have the length of the previous messages keys as two and a message number three. Bob knows that this is the third message and that the sending chain of Alice has two previous entries to the chain. This will let Bob know that he needs to store message keys for messages one and two; they can be read and decrypyted when they arrive[9].

CHAPTER 3

Related Works

This section discusses previously done work that our own work has been inspired by and what we want to build upon. We will start with an examination of Delay Tolerant Networks or DTNs. Then we will shift our focus to similar bundling concepts that others have been able to use in disconnected villages and public transport as a median to push data to a internet connected server. Finally, we will look at X3DH and public key infrastructures.

3.1 Mobile Delay Tolerant Networks

In a rural community where having an internet connection can lead to a huge influx of development and economic prosperity, enabling connections in a cheap and efficient way can be the difference between a rural community developing or staying disadvantaged. This paper focuses on one use case where an ordinary person can become a micro-entrepreneur based on this enabling of a delay tolerant network. Disney Labs Zurich coined the idea that if people can stream movies using what they called a "mobile cinema" it can bring economic wealth to many individuals who would normally not have the opportunity[10]. These networks would build upon infrastructure that this lab would stand up to. They would equip mobile information stations on frequent public transportation vehicles such as taxis [10] while keeping the required infrastructure at a low cost in order to make it successful and accessible[10]. This paper was important in our consideration of how to build a project on top of already existing work with DTNs and disconnected or remote villages. Disney Labs Zurich was able to enable this DTN environment using the custom-built infrastructure. Our project wants to enable the same connectivity without propping up new and expensive hardware. Using a typical mobile phone will help cut costs, increase accessibility, and lower the overhead to maintain the application. Having app based

infrastructure also allows us to leverage any device running Android or iOS. Using these OS's empower our team to use only standard permissions that users would give us access to on their phone. This allows us to create an app that works well with user consent and will not overstep the boundaries of what is allowed.

3.2 Remote DTNs with bundling abilities

A huge portion of real DTN environments is examined in an urban environment with already existing, though underpowered, infrastructure[11]. These types of networks can benefit hugely from DTNs as they can spread out the loads on certain regions of a network. A study conducted by Grasid and Lindgren was able to layout stationary and mobile infrastructure in order to provide internet and connectivity to a remote village in Sweden [11]. In this paper a proof of concept of what our group is going to attempt to recreate and improve on is outlined. They created a server that makes network requests and bundles them together in order to send them out as delayed requests[11]. The study had specific compatibility with a range of apps including an open-source email client, a server that had pre-selected podcasts that it would scrape and cache for users, a web caching server similar to the podcasting service and a SMS server to send messages. This scenario laid out a trusted network model where everything in this network is good and behaves in a correct manner[11]. Our lab wants to improve upon this model by not only bridging these servers up to date with modern applications like Signal, but also building a network off of a trust-less model. Our group does not care who gets what and where data goes, no one will be trusted and thus our infrastructure won't know what it's transporting.

3.3 DTNs over the Metropolitan Public Transportation

Human mobility is extremely unpredictable and not constant, so the solution that allows DTNs to be consistent in multiple scenarios is public transportation. This

infrastructure can be installed on multiple bus lines and allow labeled data movement throughout each line depending on a return route. When a user gets aboard or near a bus with a module installed on it, their request is forwarded to the buses server with an identifier that indicates which bus line they will be taking home. This predictive network routing allows the DTN to plan out which returned requests will be stationed where. This planning can and will relieve strain on a network that could normally be flooded with requests from anyone and everyone[12]. In our study we will begin with this random request distribution as our network is labeled as untrusted. This allows our bus driver to accept every request it finds as long as there's space for them. When the driver returns they can distribute the requests to everyone and only the original sender can decrypt their zip bundle. As our project expands and needs to scale, we will adopt the methodology in this paper. By only accepting messages in which their response will be going in the same place as the request was sent. The bus driver's phone can use more space, and our bundling server can be made more aware of which buses often contain the most data. This paper was extremely helpful when exploring delivery options for messages since they used the same basic architecture that our lab is considering. This allows us to take what they learned in these congested areas and adjust our distribution model for a completely opposite scenario, servicing villages.

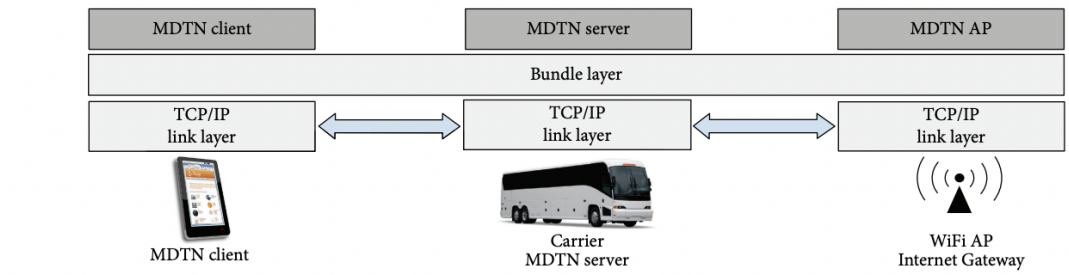


Figure 5: Transport Network [12]

3.4 WaterChat: A Group Chat Application Based on Opportunistic Mobile Social Networks

An opportunistic mobile social network can be defined as a delay tolerant network that plays off of social interaction between the nodes. In order for a network like this to function, it's important to analyze patterns of each of the nodes. This way, there can be a predicable delivery window for packets to get to their destination. Waterchat is a store and forward based chat application that supports chat rooms and individual messages[13]. The user of this app can also prioritize the delivery of certain messages based on their importance. Similar to our own, their initial approach before networking optimizations is a spray and pray approach. The idea is that in order to lower the delay of total message delivery time, they want to send out messages to every node that is seen in this opportunistic network by a client[13]. By bouncing the message around the ad hoc network, there is a high probability that a message will be delivered to a desired node. The groundbreaking part of this strategy is the optimizations they make when sending fragments of messages or updates regarding messages if they are already in the network. By controlling the updates to the network in an efficient way, Tsai, Liu and Han prove that their protocol is effective and superior to a pure broadcast and networking flooding delay tolerant network[13]. Their protocol has a higher delivery ratio and lower message delivery delay. This paper relates to the work our lab is doing as these types of studies showcase the different delivery and distribution methods that can increase the chances of delivery of Signal messages. It also showcases store and forward and its effectiveness in a near identical use case, validating the strategy the Signal plug-in and Signal app will be using to hold and prepare information for transit.

3.5 Signals Handshake X3DH

The Signal protocol consists of 2 major security protocols, X3DH and the double ratchet protocol. X3DH is used in order to do key exchanges for initial session keys required in Signal. Hashimoto and Katsumata define a new type of X3DH coined as a Signal conforming authenticated key exchange(AKE) protocol[14]. This is done in order to define a proper security model based on other authenticated key exchange protocols. In order to build this new version of X3DH, they used a plethora of cryptographic primitives like key encapsulation mechanisms(KEM), a signature schema and a pseudo-random function[14]. They believe with this design the security in X3DH is comparable to the security achieved using the double ratchet protocol. This X3DH improvement was put to the test using NIST post-quantum standardized processes which compare the computation power and bandwidth needed to break these encryptions. While our lab is not making a new version of the X3DH protocol to use in place of Signals, this paper shows us the various ways that X3DH is being iterated on due to it lack of deniability. This deniability is seen in Signals X3DH key exchange because when Alice and Bob want to start a conversation, one of them must be seen as the initiator. This initiator can deny that they played any role in messaging a person because the information that they send leaves no trace. However, the responder can be tied to the conversation which can create a vulnerability in who is doing what. This modified X3DH algorithm strengthens the current durability paradigm, making it safer.

3.6 PKI in a DTN environment

Public key infrastructure is a highly used method for accessing and verifying the legitimacy of public keys and users certificates within a system[15]. These systems typically do not work well in a delay tolerant network because of the nature of

intermittent connection and unknown delays. PKI usually consists of an online certificate status protocol (OCSP)[15] and a certificate revocation list (CRL)[15]. A OCSP is a framework for checking to see if there has been a change in a specific certificate. This systems is always querying an online setting, which makes it not runnable in an environment that is not typically connected. A CRL is a similar mechanism but it's a blacklist for certificates that have been revoked before they expire. Due to this offline nature, this kind of list of revoked certificates can be distributed among many of the nodes on the network as connections become available. A problem that can arise is that as the network grows its difficult to maintain an accurate record at scale and verfiy that all CRLs are signed with a real certifcate[15]. Bhutta and Cruickshank propose a solution that uses a much simpler and lighter CRL mapping system that allows nearby nodes to also validate the certificate authority offline[15]. This verification is done on startup by computing the hash of the original CA certificate that it's granted and its closest neighbor according to a defined formula[15]. If it's a verified node it takes on its persistent storage. Dealing with Signal in this DTN environment will create storing and verifying keys difficult. As the Signal plug-in we are designing needs to retain the least private information as possible, our lab will need to design a similar PKI that can verify keys in a similar manner as this proposed solution. This PKI system will also need to store names and their keys that are needed for sending and receiving messages in Signal.

CHAPTER 4

System Design

4.1 Overarching Design

When creating a design for this project, it was obvious that this was going to be a multi-device system that needs to work in an extremely untrustworthy and delayed environment. There needs to be a client side application that runs on a users phone, that monitors a modified Signal app to see when messages are queued to send. Then, there needs to be an application that the "bus driver" installs that will transport messages between the disconnected and connected environment. In this scenario a "bus driver" can be anyone who is frequently moving between a connected and disconnected environment. They will be a message mule for our infrastructure. This application needs to automatically connect to in-range devices that are requesting to send their encrypted zip files. These zip files are formatted to contain pending message requests from the Signal application. The way our system will be designed allows our application to not read anything inside of these encrypted files in order to protect the privacy of our users. Once these packages are transported to a connected environment, they will be sent to our bundler server. This service will send messages and receive messages for each user who has an account with our service.

The focus of this project will be on the Signal plugin inside of the bundler server. The design of this server needs to be done in a way that it can be leveraged for other services besides Signal. The hope for our infrastructure is that it can be deployed cheaply and have an open API to make requests per application. Proving that the Signal protocol and app can work in a DTN will lead other developers to create support through this open API. This will lead to more users in these DTNs to receive more service compatibility.

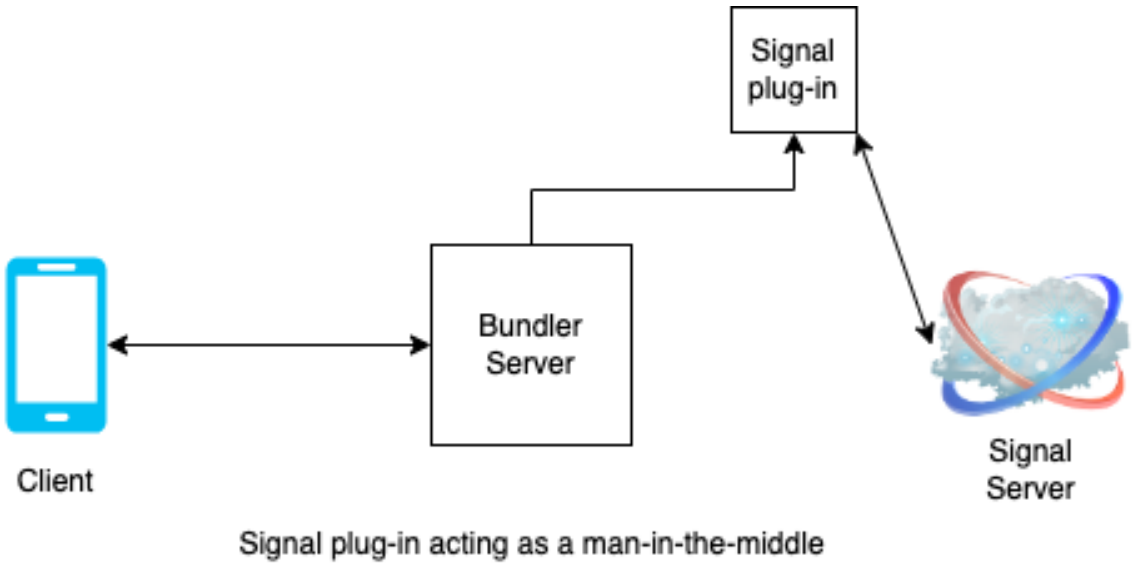
4.2 Scalability

When designing a server that will eventually be dealing with larger requests for numerous amounts of apps and servers, scalability needs to be at the forefront. Scalability is a term that can be used to describe a specific systems ability to handle increased amounts of load as a service grows overtime. How much load and what type of load depends on the system expectations and data that a system intends to process. A system can be great at scaling quickly for peak load but suffer when the average load creeps at a quick rate. This predicament happens in all developing and aspiring scalable systems [16]. The important thing to take away from these strategies is to build your system to be flexible as your priorities change with the growth a system receives. Another aspect of scalability is the performance one can expect at these high load times. Things to take into account are how a system may perform if parameters, power of the system, are left unchanged. Will requests get dropped, or will the system just take an immense time to respond to them? These are important design choices that may have an impact on the stability of a system. Moreover, if the parameters are increased in the system, how much do they need to increase for the performance to be unchanged with the increase of requests? Oftentimes systems aim to please a certain percentile of requests that are outliers, this can guarantee that some of the worst requests will only take a certain allowable amount of time. All of these above factors are important to consider when designing a system that is capable of receiving many thousands of requests from our "bus drivers".

CHAPTER 5

Design of Signal Plugin

With this scalable server, our main service inside of it will be a Signal plug-in, that will be specifically forwarding and receiving requests from Signal. When a clients requests gets off of one of the buses and forwarded to this plug-in, a set of important operations must happen. Firstly, the bundle needs to be "unbundled" exposing the raw encrypted bit stream that is the Signal message. These bits are encrypted by the phone that it came from, allowing us not to worry about privacy of a user's message. Next, the plug-in needs to be able to figure out who this request is coming from and how to authenticate with the Signal service. In order for this to happen, a few important bundles of information are needed unless the client decides to sign up through our registration service using a Google Voice number. The bundles that are needed include the public identity key, the signed prekey and any amount of one time prekeys that was discussed in the X3DH section. These are needed so that once their number is registered other clients can send them messages. This flow is seen in figure 8. As privacy is the main focus, it is important to note that our application can never read the messages being sent through our system. We allow the client to handle the Double Ratcheting protocol while we talk to Signal as a middleman. This allows the plug-in to act as a middleman with none of the normal exploits that can occur.



Signal plug-in acting as a man-in-the-middle

Figure 6

5.1 Formalized Design Flow

For someone to use the Signal Plug-in in a complete offline and delay tolerant manner, a user may sign up through our service. By signing up through the Signal plug-in it allows us to capture all of the necessary information that the plug-in can use later to open a session on there behalf. In order to open a session to listen and to send messages using the open source libsignal-java package[17], the plug-in will need to retrieve a set amount of keys for the X3DH protocol. Once the X3DH handshake is complete Alice and Bob can exchange messages encrypting each one using the Double Ratchet protocol. Now when a recipient decides to reply to the user who used our platform to message them, their messages will come to our Signal plug-in that is openly listening for replies. When we receive replies we will re-bundle those replies up to forward them back to our bundler server which will then attempt to return the message to the original person' s client app.

5.2 Challenges of libsignal-java

The general appeal of using Signal messaging over other messaging apps is the fact that signal operates as an open source client and server. Anyone can visit Github.com and glance over source code to determine if Signal is legitimately encrypting and delivering a message. With this open source nature, Signal has created a library in many different languages that allows a developer to interact with any Signal server, whether it's the official servers or a locally run one. While this library empowers developers to make their own applications interacting or using signal, the first hurdle that this created was the complete lack of documentation. Researching exactly what parameters are needed to send and receive messages led to a struggle of finding out what parameters were required for what objects. It also seems that most of the available documentation is completely out of date with the code. Signal was very lackluster on the requirements of many of the KeyStores that were needed in order to register and also create a session with the Signal server. The little information they provided required a lot of context that was not given that the development team had. It's obvious that as an open-source project, resources for Signal documentation are slim. Also, when trying to communicate with a real Signal server, our Signal plug-in needs a real certificate inside of the required TrustStore object. This TrustStore object is a child of a KeyStore object that stores this valid certificate. This certificate is self signed and is not verified by a certificate authority. This was odd as Signal is a major company with an open source library made to interact with their services. Since Signal is an end to end encrypted service, Signals servers may not find it necessary to be externally authorized by a CA as a man in the middle attack would yield no information leakage as it's all encrypted.

5.3 Registration

When a user first signs up for Signal their client device generates a few required keys that are important when communicating with Signal. These generated keys that our Signal Plug-in requires are properly split in a way where we do not gain access to keys that allow us to spoof ourselves as the client. We do not take in the private identity key, and we deem the client to keep it. This split key management allows the client to retain total control over how they encrypt their own messages. These keys are described as an identity key, several prekeys and a signed prekey. These keys are important as this bundle of keys is sent to Signal in order to send to clients who wish to communicate with you. They are used in order to generate the initial message for the X3DH process to begin. The identity key is an elliptical curve public and private key pair. The numerous prekeys also use the same elliptical curve public and private key generation as the identity key. The difference between these keys is that the prekeys are disposable and are sent to the server on a timely basis. These prekeys are used to calculate the secret key or SK that is used for the initial messages in the X3DH handshake shown in figure 9. If a client requests a bundle that does not have any disposable prekeys, a calculation can still be done that will still allow the SK to be generated for X3DH. This is possible because Signals server being out of prekeys is a very real scenario. If a client does not reconnect to the servers for months, there needs to be an alternate way to send them a first message. That's why there are alternate calculations that can take place in order to send the initial message. These generated keys are then sent up to the Signal servers for Signal to distribute to any potential client that wants to communicate with your registered number. Within the implementation it is extremely important to keep these keys in a durable storage as the identity key is used to communicate and start sessions with other users through the Signal servers. When a user is fully registered after doing a

voice or SMS verification, a UUID is assigned to the client that also should be saved in a durable fashion. This UUID is also an important user identification attribute that the Signal servers use when verifying requests along with the identity key. Without these two entities, a session cannot be opened to receive or send messages. This storing of UUID and public identity key is similar to a PKI, as we are associating our users with these two pieces of information in order to always forward their messages to the correct person. The Signal plug-in will always use this PKI-like system to lookup the information that is needed to open a session to send a message. Our Signal plug-in will need to receive both of the public identity key and the UUID in order to effectively send and receive messages in place of the client. Both of these pieces of information have nothing that can compromise the user. The reason that the public identity key and the UUID stay secure is because if a malicious user were to obtain these entities they would not be able to send a message that the end user could decrypt. This is because the Signal plug-in is not managing any of the actual message encryption. If a malicious messages are sent to the recipient, their device would simply not be able to decrypt and throw an error. This is because the malicious sender would not have any idea what the stolen users private key is that is being used to encrypt messages. That key is purposely left on the clients device, allowing the plug-in to just act as a proxy for sending messages. This reinforces our end to end encrypted model throughout the application. The more security we can offload of our application the less information leakage is possible.

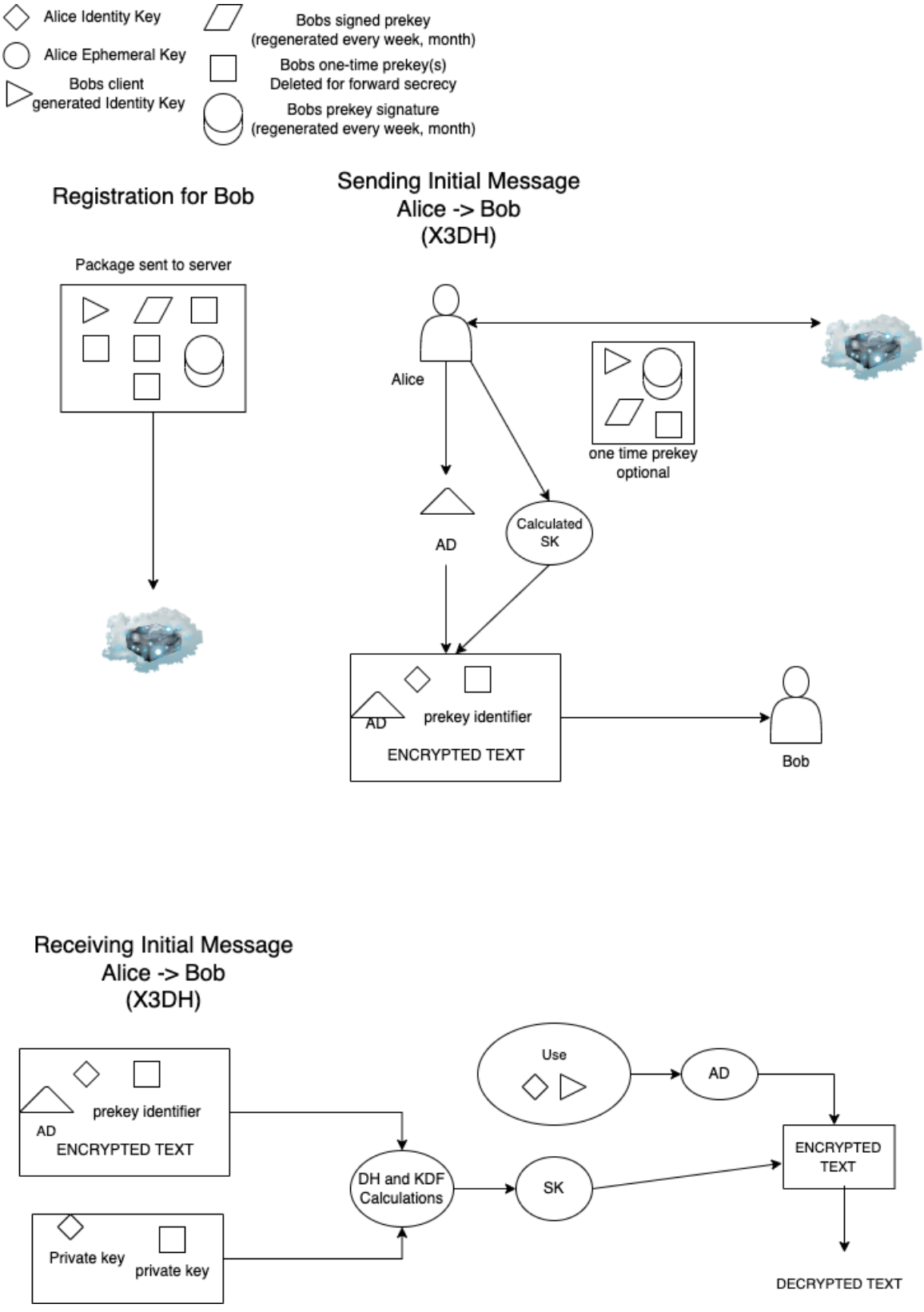


Figure 7: Registration and Sending First Message

5.4 Sending the First Message

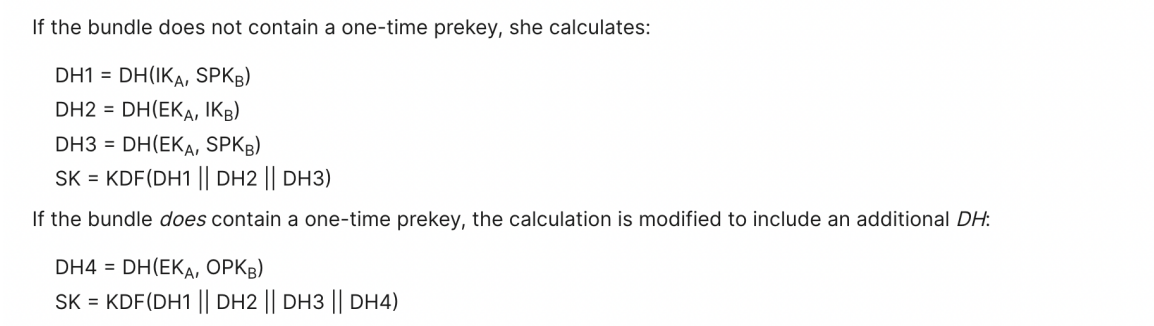


Figure 8: Secret Key calculations [8]

Let's say a potential client, Alice, wants to communicate with Bob. Alice needs to retrieve her bundle of keys from Signal and formulate a session with Bob. This session involves the X3DH handshake that figure 8 depicts. The set of calculations can be seen that result in the successful secret key that allows the first message in the chain of communication. In that diagram one can notice that a variety of different keys are used that were mentioned above and some that were not. Not only does Alice need to use her own keys when generating the secret key, but Alice needs to use the keys from Bob's bundle that they retrieve from the signal server. The necessary keys needed for such a handshake are the identity public keys, signed prekey pairs, some quickly generated ephemeral key pairs, and optional prekey pairs denoted in figure 8. Once these calculations are all completed, Alice deletes her ephemeral private key and the outputs used to create the final secret key. Alice then appends an additional piece of information that is the encoding of both Bob's and Alice's identity key, AD, and uses it along with the secret key to finally encrypt the ciphertext. In figure 8, it displays the computation that is done in order to receive the secret key. In this equation, IK are the identity public keys, the SPK are the signed prekey pairs, EK are the ephemeral keys that are generated and all have subscripts denoting whether they

are Alice's or Bob's respected key. When Bob receives this initial message, his task is to retrieve Alice's identity key and the ephemeral key from the header of that message. Then Bob retrieves his own keys that are needed, such as his identity private key and those private keys corresponding the his signed prekey and optional prekey. He then does his own calculations according to figure 8 which yield him his own secret key. Follwing that, he also generates his own AD and uses the secret key and AD to decrpyt the ciphertext. If this operation fails, Bob aborts this session and deletes all related information. If it is successful, Bob deletes any one time prekeys that were used in order to keep forward secrecy enforced. This is an extremely important step because if someone is listening as a man in the middle and manages to decode the secret key and AD, then they will not be able to use that same key to decrypt future or past messages. The idea of a unique key being used per message ensures that user data is kept as secret as possible in the event of a key leakage.

5.5 Sending Messages

In order to send messages following this handshake, the client side Alice, needs to refresh this secret key and AD information every set amount of time. Inside of our Signal plug-in, multiple in memory keystores need to be created to start a session in order to open a connection to the Signal servers. A key store is needed for each type of significant key needed for X3DH and the Double Ratchet protocol. Once their key stores have been formed, a session with the Signal servers needs to be built that passes these key stores and the receipts phone number that the app will be sending messages to. This creation of a session creates and object that is then used to retrieve a prekey bundle that belongs to the desired recipient. With this prekey bundle fetched, we can process the delivered keys in order to commence a proper X3DH handshake. Once this handshake is completed an initial message can be sent. This process is repeated

for every message that wants to be sent. Now this method above is well documented in Signals documentation but their lib-signal api lacks the proper api to do this exact task. They refer the client to grab a prekey bundle located on the server that contains the above needed keys that the recipient sent to the signal server on sign up. In their testing, instead of mocking a request to their server with the correct api call that one may use, they hard code in key bundles to use. There is no clear way on how to get their key bundles, and I think the reason may be that Signals api is not built to communicate perfectly with their own service. Its built to be used with any generic server running their signal-server code. They assume that whatever server you are talking with will have an endpoint to get a prekey bundle, but they may not want to expose that information in this api. They may see it as some sort of information leakage that they do not want to contribute to. Those original api calls have not changed at the same rate as the api documentation itself when examining the git commit history of the repository as well. Our application does not need to worry about that cipher text being encrypted, as it was already encrypted by the sending device. That message is end to end encrypted. We allow the sending and receiving devices to do the encrypting before we ever get the message; thus we do not need to worry about doing any encryption on our end.

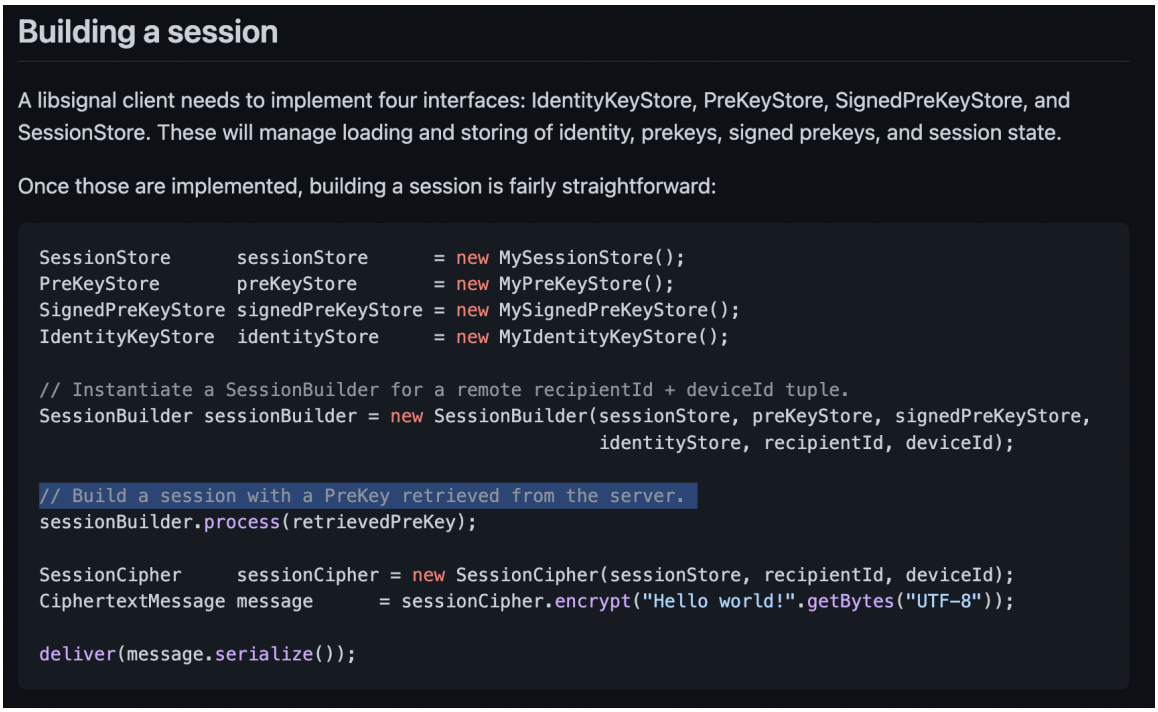


Figure 9: Gap in Documentation [17]

5.5.1 Exploring Alternate Methods to Sending and Receiving Messages

As stated above the defined and documented way to send messages is not relevant to how the current signal api actually behaves in a real message environment. Instead many recommendations in the Signal community point to use the more well documented and maintained components of Signal-Android[18]. Deep within the codebase running the android app, there lies signalservice code that is similar to an api fashion. By injecting these dependencies inside of our existing signal-plugin program, it can utilize many of the battle tested sending methods inside of Android. Inside of this android package, to send messages a client must create SignalServiceMessageSender objects that then have the ability to send different types of messages to a recipient. These messages consist of sending read receipts, typing indicators and many other common communications that happen within the Signal app. In order to send these messages up to the Signal server, this api triggers the

program to open a pipe directly linking the Signal plug-in to the server. Think of this pipe as a web-socket. It allows the free flow of packets between the Signal plug-in and server. This type of intercommunication between the two parties has a great benefit to transferring information quickly in both directions once the pipe is established. These pipes facilitate the transportation of a messaging concept derived in Signal called envelopes. This process works in reverse with an object notably named SignalServiceMessageReceiver. This object also allows a client to send different types of packets as discussed above and open the same type of pipe object. When a pipe is opened by a client, they may request to receive any messages that have been earmarked to be delivered to them.

5.5.2 Signal Envelopes

When sending a message through a client to Signal, it would examine the incoming message to decide which user should receive it. Upon delivery it would consult the "from" aspect of the message to see who sent it. Signal decided that it's better that Signals servers do not need to know who sent the message just where it needs to go. This protects the privacy of its users even more as they are slowly working on covering up metadata tracks. These leakages of metadata could enable large agencies or governments to potentially use this known data to prove that Alice and Bob were indeed talking during a certain time period. Signal also made it a goal that this "fromless" technology should still allow for rate limiting and abuse protection when dealing with automated bots spamming messages. Another key aspect of this new protocol is that a client receiving a message should be able to validate the senders identity to prevent spoofing and provide reassurance that the message they received is truly from the correct person[19].

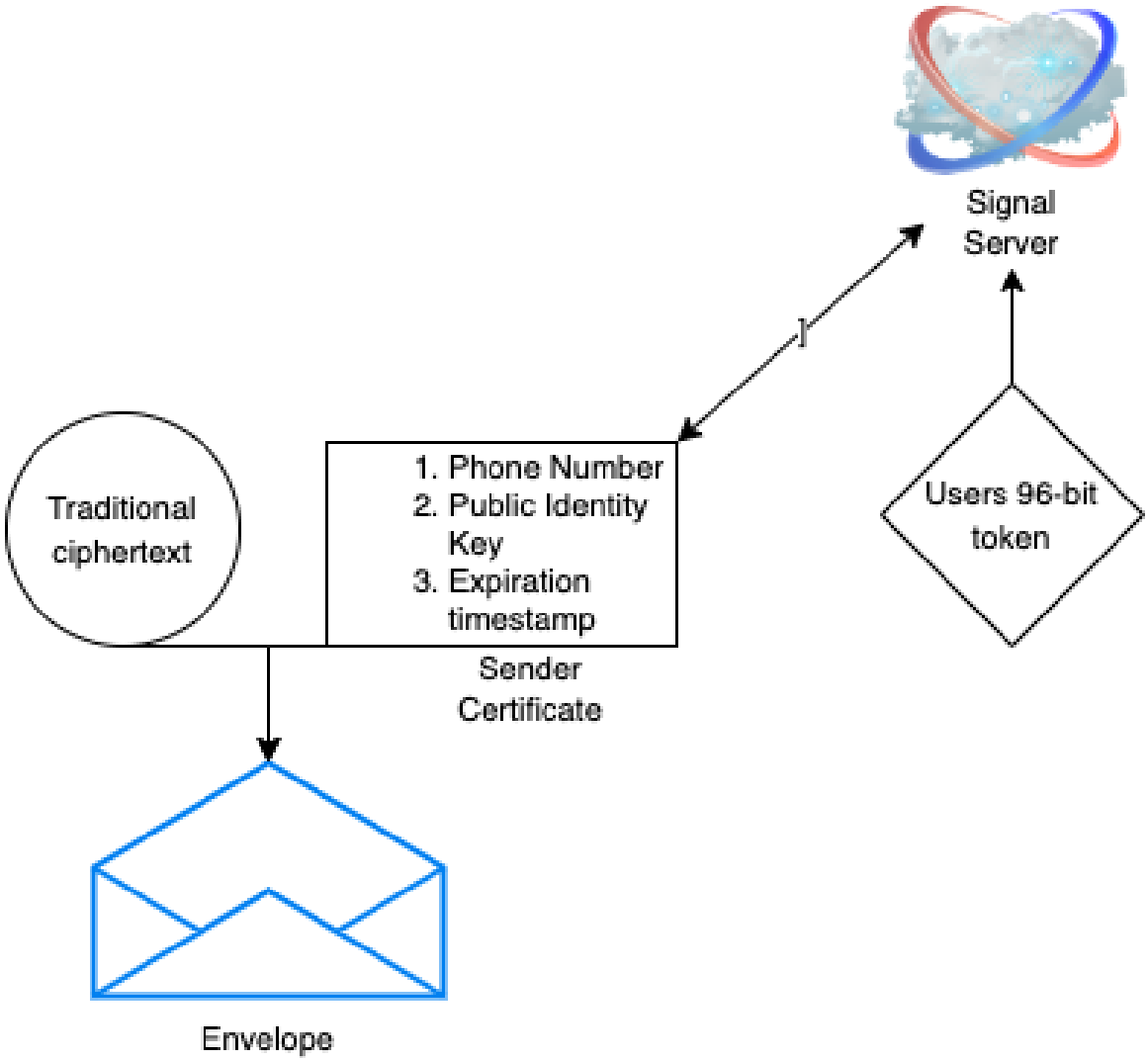


Figure 10: Signal Envelope format

5.5.3 Spoofing Prevention

In order to stop spoofing in this new paradigm, Signal required clients to generate a short term sender certificate from Signals servers that prove their identity. This certificate contains their phone number, public identity key, and an expiration date. This certificate is included in sent messages.

5.5.4 Abuse Protection

In order to combat abuse and spam, Signal asks that clients create a 96-bit token derived from their profile key and send it to the Signal servers. There is no expiration on this token, the only time this token needs to be updated is when a user's Signal profile is edited in a way that alters their identity. An example of an alteration that would call for an update is if a user changed the phone number that their account was associated with. Signal then requires clients to verify and prove that they know information about that token for a user in order to send a "fromless" or "sealed sender" message to that user. These profile keys are exchanged with your contacts on Signal or other people or groups that are granted permission. Having this requirement of needing to actually approve and know someone to get messages from them brings down the chance of spam and abuse. This setting can be disabled by any Signal user if they choose to receive messages from anyone that is not included in their contacts.

5.5.5 Encrypting & Sealing the Envelope

The cipher text that is traditionally in a Signal message is still encrypted with the Signal protocol that has been discussed, but this sender certificate is also encrypted using the sender and recipient identity keys. When the recipient receives this envelope on the other side, they can decrypt and validate the sender certificate and process the message. Signal sees this type of hidden metadata as the future of privacy. By making an effort to shield this type of data they are sticking up for their users privacy in more ways than typical consumers are aware of.

5.6 Client Requirements for Double Ratchet Protocol

In order for all of the above things to work in a disconnected fashion, our Signal plug-in needs to be able to retrieve prekey bundles for the desired client. When a Signal client wants to encrypt their messages they need to follow the proper X3DH

handshake and Double Tatchet protocol noted above. To do this, we need to pass the client what it needs. The flow would be as follows, a client Alice wants to message Bob a new message about her day. If Alice and Bob have never talked before, they will need to establish their initial secret ephemeral key to pass information about their Double Ratchet keys. So, Alice's client would issue a getPreKeys bundle to our Signal plug-in, and hold the message that needs to be sent. Once that request is fulfilled, likely in a day or two depending on our "bus traffic", the client can properly attempt to encrypt its message to send up to our Signal plug-in. As discussed previously, Signals lib-signal api lacked any way for us to retrieve these prekey bundles. They assumed that the client would be able to use the api to grab them, but it was obvious that their documentation did not line up with the current capabilities. With the introduction of this new Android api there seems to be routes built into it where one can request a prekey bundle based on the recipients address. This enables our application to truly remain clueless as to what messages are being sent through our service. This will also guarantee protection to the client's messages in their transit on the bus network. In terms of this envelope "fromless" protection that was discussed earlier, our program will have to know where to send the message when we open our pipe to Signal. We can protect the actual Signal server from knowing where the message is, and as an open source application we will not log this type of information.

CHAPTER 6

Experiment

The following section will detail the results of the Signal plug-in. This will walk through how the program functions and important inner workings of some of the interactions with Signals servers.

6.1 Registration

In order to successfully register for an account on Signal, our program needed to keep track of the many keys that were discussed in the above sections. To do this, the program had implemented KeyStore object that upon initialization would create the required keys that Signal needs in order to allow the client to register. In a full stack build of this application, our Signal plug-in will not have to generate any of these keys. The client side application on the users phone will generate and save the keys on their side, and then pass up the key information to our client in order to register. Once these keys are generated and saved, the next step is to configure the URLs that our request is going to query. This was a challenge as Signal does not advertise its own routes publicly. To find these routes I needed to "inspect element" on the Signals web app to see where requests were actually going. These routes are needed in order to create SignalServiceConfiguration that is needed to instatiate the registration object. The SignalServiceConfiguration Object has many URLs that it needs to successfully query the Signal servers. It holds the content delivery URL, the contact discovery url and the actual Signal server address. Once this object is built and configured, we then can pass it along with a few other parameters into the SignalServiceAccountManager object. This object is the main line of communication to make requests to get a verification code that is needed in order to verify a user.

```
/**
 * Construct a SignalServiceAccountManager.
 *
 * @param configuration The URL for the Signal Service.
 * @param uuid The Signal Service UUID.
 * @param e164 The Signal Service phone number.
 * @param password A Signal Service password.
 * @param userAgent A string which identifies the client software.
 */
public SignalServiceAccountManager(SignalServiceConfiguration configuration,
                                   UUID uuid, String e164, String password,
                                   String userAgent)
```

Figure 11: SignalServiceAccountManager Constructor[17]

When we make those requests to Signal we need to accompany them with a one-time captcha code that can be manually found on Signals captcha verification webpage, https://signalcaptchas.org/registration/generate.html. This is needed in order to prevent bots from signing up. This is noted because once the verification code is sent from Signal, no other request requires a captcha to perform requests. When we verify with Signal that this code is correct and this user may sign up, there are quite a few parameters that they require. The request requires a Signaling Key which is a 52 byte random integer, a registration id that is made up of a 14 byte number that is unique to the client, an access key that is also a random 52 byte array that is used to identify the request. Once these parameters are defined and in place, the code along with this information can be sent up to Signals servers who will return a 201 upon account creation. Some noted behavior that Signal can display is a 403 authorization failure. It seems to be an inconsistent yet frequent error when signing up for an initial account. When examining the Signal Server code, not much could be made as to why it returns that error code in the registration flow.

6.2 Message Sending and Receiving

To send and receive a message in Signal our Signal plug-in needs to use the registration id that was generated previously and use the identity public key to

create a SignalProtocolStore. This SignalProtocolStore is used to verify the request to allow the request to go to the Signals server and be authenticated and allowed. When sending a message our plug-in needs to make a request to build an object known as SignalServiceMessageSender. This object needs the earlier referenced SignalServiceConfiguration object with the appropriate URLs to point requests in the right place. It also needs the earlier UUID, associated phone number, and a SignalServiceMessagePipe that is ultimately a webs socket. In order to receive messages that are earmarked for delivery, the above process needs to be accomplished but with the SignalServiceMessageReceiver object. This object takes the same basic parameters as the similar sending object. Once these objects are successfully built they can be used to open a pipes to the Signal servers. These pipe will facilitate the sending and receiving of a message using the appropriate object. The function call that is being used to send individual messages allows a singular envelope to travel to Signal, but if there are multiple messages queued it can be called multiple times. For the receiver object this pipe can be read continuously every set amount of time to query for messages. Another strategy would be to grab all of the available messages once per day, or before a "bus driver" is leaving. This would allow us to optimize the retrieval of messages based on time and plan for bandwidth constraints that Signals servers may have. There is also a Signal mailbox that is present on the Signal official server[22] that our plug-in may be able to stay synced with in order to get messages for our users as they are received by the Signals servers. This approach requires much more cooperation from Signal because our server will be constantly trying to stay in sync with the mailbox, which may get the plug-in blocked mistakenly. When the bundling service is complete. Instead of sending raw text through with the encryption happening on the Signal plug-in, our Signal plug-in will be able to accept the already encrypted bytes and through it down the pipe in order to have Signal receive it.

CHAPTER 7

Future Works

7.1 Enabling Group Chats

To build in the functionality of group chats, our plug-in needs to support a new type of protocol that enables sending messages to large groups privately. Signal messaging involves some important aspects,

- Forward Secrecy and Deniability
- Transcript Consistency

Replicating these in a group setting is difficult using the traditional logic as the setup phase for a conversation is much more involved. To set up a session with multiple people one needs to

1. Talk to each member and create a session id.
2. Begin an exchange of pairwise keys to each member using long term identity keys
3. Enact a key exchange that will prove that each member has signed ephemeral keys
4. Take all authenticated keys and create a group key agreement.
5. Make sure everyone is agreeing and see the same thing

Now this process is much more involved but the part that makes this specifically hard for our DTN application is the number of network hops needed for such exchanges[20]. According to Matthew Van Gundy, the creator of this protocol, the amount of hops needed to just establish the session can be anywhere from 4 * N to 12 * N where N is the number of group members[21]. Also factor in that our server may not have access to some of the keys needed to sign and authenticate different keys

throughout the process. As this session plays out it can become too many hops for this to be a practical and seamless experience for the end user. That is, unless Signal works directly with our lab to create a better solution for a DTN environment.

7.2 Working Directly with Signal

As this project develops our lab hopes that Signal as a platform will make it easier for a delay tolerant version of the app to be functional. As of right now, when a user signs up they must have internet to receive the verification code through Google voice or must have cellular signal and internet to receive the code on their native phone number. When sending and receiving messages we also must store there long term identity public key as well as there UUID that is bound to the device they signed up on. Storing as little information about users is the best strategy moving forward for our labs application and with these requirements that Signal currently has for their compatible libraries it leaves us no other choice. Signal should drop the verification using a phone number or automate the process in partnership with Google voice when signing up from a delay tolerant network environment. If they could figure out a alternative verification method they would be opening their application to those who have never had an internet connection in there life. This could bring many more users and enable people to be more connected globally. Moreover, if they could integrate into this DTN paradigm, they may be able to adapt their current libraries to use our Signal plug-in and keep more information encrypted and private per user. That is the goal with this proof of work our lab is conducting using Signal.

7.3 Expanding Application Compatibility

Signal made the perfect application to demonstrate that harnessing the power of DTNs and distributed anonymous transport is possible in a encrypted messaging instance. This demonstration is meant to serve as a working example of how any

company can integrate their application to work in this same environment. By working with our future open api that will be application agnostic, companies can take advantage of a completely untapped market. Streaming services can charge small monthly fees by allowing users to request content to watch offline the next day. Social media companies can integrate themselves into a completely new demographic and allow them to share their experiences with others across the world. By simply creating a future api that is compatible with not just signal but forwarding any type of packet to a specified service, we can ensure compatibility with many different models. Alternately, companies can take the first step and attempt to design their own api's specifically designed to handle DTN requests.

CHAPTER 8

Conclusion

As more and more individuals remain disconnected and lack internet access there still remains a disparity of information and communication throughout the world. Our lab took this problem and attempted to transform this disconnected reality for many people and empower them to communicate with loved ones and friends. Many related studies proved the efficacy of such a solution in many different applications which validated the need and prospect of creating something like this. We wanted it to be usable and accessible to users and a low cost to maintain to developers. These characteristics were at the forefront when tackling the challenging design and details.

The Signal application creates a safe space for anyone to message each other free of government or company oversight. It brings back privacy that many consumers have lost in other areas of technology. They have managed to package such a complicated protocol into such an easily used application. The steps that are done to encrypt a message have been vetted and utilized by many other encryption driven messaging apps. This gives out lab confidence that this was the correct proof of concept application to trial for our delay tolerant network delivery system and infrastructure.

Creating this plug-in for Signal was no easy task and brought lots of roadblocks with it in its development cycle. This created many unique challenges that could only be solved with immense knowledge of how the Signal protocol functions and the necessary information needed for each step without compromising our users privacy. Signals protocol is highly documented on a surface level but lacks a lot of documentation in the code implementation. This makes working with it a challenging and experimental environment.

Our labs system ensures the fair and end to end encrypted delivery of messages from anyone who wants to utilize our system. The entirety of the stack and the

scalability of our Signal plug-in promises that a users messages will eventually be delivered in a secure fashion. Enabling this broad level of connection among the worlds can help foster new ideas and spread creativity to those who may not have access to it currently makes this a truly fulfilling project.

LIST OF REFERENCES

[1] Two Thirds of the World's School-Age Children Have No Internet Access at Home, New UNICEF-ITU Report Says. UNICEF,

[2] Fainmesser, Itay P. "Digital Privacy." Management Science, 15 Aug. 2022, https://pubsonline.informs.org/doi/10.1287/mnsc.2022.4513.

[3] "Developing Open Source Privacy Technology That Protects Free Expression and Enables Secure Global Communication." Signal Foundation, https://signalfoundation.org/en/.

[4] "Signal." GitHub, https://github.com/signalapp.

[5] "Signal Private Messenger - Apps on Google Play." Google, Google, https://play.google.com/store/apps/details?id=org.thoughtcrime.securesmsamp;gl=us.

[6] "Moxie Marlinspike Leaves Encrypted-Messaging App Signal." BBC News, BBC, 11 Jan. 2022, https://www.bbc.com/news/technology-59937614.

[7] Buchanan, William J (2022). X3DH (Extended Triple Diffie-Hellman) in Go. Asecuritysite.com

[8] Specifications the X3DH Key Agreement Protocol." Signal Messenger, https://signal.org/docs/specifications/x3dh/.

[9] Specifications; the Double Ratchet Algorithm." Signal Messenger, https://signal.org/docs/specifications/doubleratchet/.

[10] A. Galati, T. Bourchas, S. Siby, S. Frey, M. Olivares, and S. Mangold, "Mobile-enabled delay tolerant networking in rural developing regions," IEEE Global Humanitarian Technology Conference (GHTC 2014), 2014.

[11] S. Grasic and A. Lindgren, "Revisiting a remote village scenario and its DTN routing objective," Computer Communications, vol. 48, pp. 133–140, 2014.

[12] A. Bujari, S. Gaito, D. Maggiorini, C. E. Palazzi, and C. Quadri, "Delay tolerant networking over the Metropolitan Public Transportation," Mobile Information Systems, vol. 2016, pp. 1–14, 2016.

[13] "WaterChat: A group chat application based on opportunistic mobile ...," 07-Jul-2017. [Online]. Available: http://www.jocm.us/uploadfile/2017/0809/20170809102915600.pdf. [Accessed: 01-Nov-2022].

[14] Hashimoto, Katsumata, S., Kwiatkowski, K., Prest, T. (2022). An Efficient and Generic Construction for Signal's Handshake (X3DH): Post-quantum, State Leakage Secure, and Deniable. Journal of Cryptology, 35(3). https://doi.org/10.1007/s00145-022-09427-1

[15] Bhutta, Muhammad Nasir Mumtaz, et al. "Public-Key Infrastructure Validation and Revocation Mechanism Suitable for Delay/disruption Tolerant Networks." IET Information Security, vol. 11, no. 1, 2017, pp. 16–22, https://doi.org/10.1049/iet-ifs.2015.0438.

[16] Kleppmann, Martin. Designing Data-Intensive Applications: The Big Ideas behind Reliable, Scalable, and Maintainable Systems. O'Reilly, 2021.

[17] Signalapp. "Signalapp/Libsignal-Protocol-Java." GitHub, https://github.com/signalapp/libsignal-protocol-java.

[18] Signalapp. Signalapp/signal-Android. GitHub, https://github.com/signalapp/Signal-Android

[19] Technology Preview: Sealed Sender for Signal. Signal Messenger, https://signal.org/blog/sealed-sender/.

[20] "Private Group Messaging." Signal Messenger, https://signal.org/blog/private-groups/.

[21] Improved Deniable Signature Key Exchange for Mpotr. https://matt.singlethink.net/projects/mpotr/improved-dske.pdf.

[22] Signalapp. "Signalapp/Signal-Server: Server Supporting the Signal Private Messenger Applications on Android, Desktop, and IOS." GitHub, https://github.com/signalapp/Signal-Server.