

Spring 2023

Sending and Receiving Internet Messages from Disconnected Areas

Abhishek Prakash Gaikwad
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Recommended Citation

Gaikwad, Abhishek Prakash, "Sending and Receiving Internet Messages from Disconnected Areas" (2023). *Master's Projects*. 1210.
DOI: <https://doi.org/10.31979/etd.beq8-ak3y>
https://scholarworks.sjsu.edu/etd_projects/1210

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Sending and Receiving Internet Messages from Disconnected Areas

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

Abhishek Prakash Gaikwad

May 2023

© 2023

Abhishek Prakash Gaikwad

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled
Sending and Receiving Internet Messages from Disconnected Areas

by
Abhishek Prakash Gaikwad

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

Dr. Ben Reed	Department of Computer Science
Dr. Navrati Saxena	Department of Computer Science
Bhushan Sonawane	Master of Science in Computer Science

ABSTRACT

Sending and Receiving Internet Messages from Disconnected Areas

by Abhishek Prakash Gaikwad

Over 62% of the world is connected to the internet with more than 6.9 billion smartphone users. The omnipresence of technology in the form of the internet and smartphones have led to constant research in improving communication throughout the world. But even today, 37% (2.9 billion people) are not connected to the internet even though most of the people in such areas have smartphones. To solve this problem of access to internet services in disconnected areas, a software-only mobile-first approach has been proposed for disconnected data distribution infrastructure which can support different internet applications in limited connectivity. A prototype application based on Signal Messenger has been created to allow users to send and receive internet messages without the need for internet connectivity. This solution can help bridge the digital divide, improving access to critical communication services in disconnected areas.

ACKNOWLEDGMENTS

I would like to acknowledge and give my warmest thanks to my advisor and mentor Dr. Ben Reed for his patience, enthusiasm, and immense knowledge. His guidance, advice, and vision have made this whole project possible. Thank you for teaching me how to approach a difficult problem and bring an idea to reality.

I would also like to thank the rest of the committee: Dr. Navrati Saxena and Mr. Bhushan Sonawane, for their encouragement, insightful comments, and questions which helped shape this project.

Finally, I thank my fellow labmates of the Disconnected Data Distribution Group: Shashank Hegde, Anirudh KC, Aditya Singhanian, and Deepak Munagala for all the stimulating discussions and brainstorming sessions that helped shape this project.

TABLE OF CONTENTS

CHAPTER

1	Introduction	1
1.1	Disconnected Signal Messenger	3
1.2	Goals	3
2	Related Work	5
2.1	Using DDD for sending internet messages	5
2.2	Data Transfer	5
2.3	Infrastructure	6
2.4	Data Distribution	8
2.5	Security and Encryption	10
2.5.1	Signal Protocol	11
3	System Overview	17
3.1	DDD Architecture	17
3.2	Keys and IDs Generated for Registration and Session Management	20
3.3	Registration and Session Management	21
3.3.1	Client Side Operations	23
3.3.2	Registration	23
3.3.3	Verification	24
3.3.4	Finding Contacts in Signal Network	24
3.4	Receiving Messages	25
3.4.1	Initiate Receiving	25

3.4.2	Receiving Messages	27
3.4.3	Storing Messages	27
3.4.4	Sending Message Files to Disconnected Clients	27
4	Implementation	29
4.1	Signal Android	29
4.2	Bundler Signal Service	30
4.2.1	Operations	31
4.2.2	Dependencies	35
4.2.3	File Structure, Key Exchange and Storage	36
5	Experiments	38
5.1	Signal Android Application	38
5.2	Bundler Signal Service	39
6	Future Work	42
6.1	Limitations	42
6.2	Enhancements in Bundler Signal Service and Signal-Android	42
6.3	Collaboration with Signal Developers	43
6.4	Potential of DDD for Various Internet-based Applications	43
7	Conclusion	44
	LIST OF REFERENCES	46
	APPENDIX	

LIST OF TABLES

1	Keys and IDs in Disconnected Signal Messenger	21
2	Dependencies for Bundler Signal Service	35
3	Files and Directories of Bundler Signal Service	37

LIST OF FIGURES

1	Disconnected Data Distribution of Internet Data	2
2	Daknet	9
3	X3DH Asymmetric Key Generation Algorithm.	12
4	Double Ratchet Algorithm	14
5	Sesame Algorithm for Asynchronous Session Management.	15
6	DDD for Signal Messenger	17
7	Sequence Diagram of Registration	22
8	Sequence Diagram of Receiving Messages on Signal in DDD	26
9	DDDKeys Attributes and Constructor	30
10	Directories, Files and Databases of Bundler Signal Service	36
11	Keys File Stored at Bundle Client	39
12	Finding Phone Numbers Pre and Post Registration Attempt	40
13	Attempt to Decrypt Messages on Bundle Signal Service	41

CHAPTER 1

Introduction

The Internet is a tool for information, communication, and entertainment. It has become an unavoidable part of peoples' life, with the average time spent by an individual online increasing every year [1]. One of the most important uses of the internet is communication which takes place by sharing data. The internet offers multiple ways to communicate, from instant messaging to placing calls. Moreover, internet communication ensures an instant connection between people on opposite sides of the world. Today, society is heavily dependent on this digital connectivity, and people who lack internet access are at a disadvantage. Internet access is not universal, and there are still many areas where people lack reliable connectivity. This digital divide can have serious consequences, particularly for those who are already marginalized or disadvantaged. It can limit educational opportunities, hinder economic development, and exacerbate social inequalities.

During times of crisis, such as natural disasters or pandemics, internet access can become even more critical, as it may be the only means of communication and information dissemination. However, during such events, internet outages, and disruptions can occur, leaving even those who normally have access without reliable connectivity [2]. One of the most relevant examples of disconnected areas is remote villages having poor internet connectivity. The lack of internet is a significant issue in underdeveloped nations. During the Covid-19 pandemic, many parts of the infected nations were in a state of lockdown, and students who lived in poorly connected areas have suffered a lot as they did not get class content [3]. Ensuring universal connectivity must be a priority if we are to build a truly inclusive and equitable digital society.

This research attempts to solve the internet connectivity problem in disconnected

areas by treating it as a data transport and distribution problem. There have been many attempts to provide connectivity to remote areas like [4, 5, 6]. All these projects worked initially but lack sustainability due to manual intervention, maintenance, or additional infrastructure. This project plans to make use of advancements in smartphones to create a Disconnected Data Distribution (DDD) network of internet data where internet data can be transported to a group of disconnected users with the help of people (host phones) moving between disconnected areas and connected areas. Figure 1 shows the movement of data to and from disconnected regions with the help of host phones.

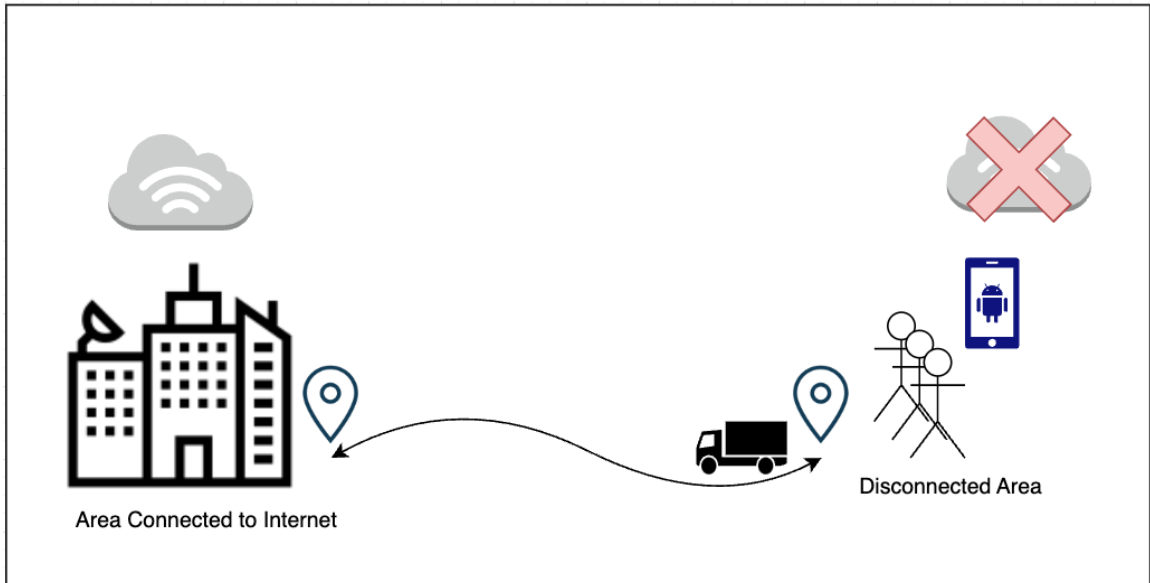


Figure 1: Disconnected Data Distribution of Internet Data

DDD is a form of Delay Tolerant Network (DTN) [7] which is built upon the idea of an opportunistic mobile social network [8]. DDD is a software-only solution using internet servers and standard Android phones where only some of the phones have internet access. Phones can share data between them using applications designed for DDD which use technologies like WiFi Direct [9] that create an ad-hoc wireless

network between two devices that allows them to transfer data.

1.1 Disconnected Signal Messenger

Many different internet services can function in a disconnected setting like email, social media, streaming websites, etc. Signal Messenger [10] is known for its strong encryption and privacy features, which make it an preferred option for users who are concerned about their online security. By creating a version of Signal that can function without an internet connection, remote users will be able to communicate securely and reliably even when traditional network infrastructure is unavailable. Signal Messenger is based on the Signal protocol, which offers end-to-end encryption for voice calls and instant messaging. Even in disconnected settings, users can generate the necessary keys to join the Signal Messaging Network. However, due to transport limitations and delays, the platform currently does not support voice calls, group messaging, or media sharing.

This research mainly focuses on registration and session management of the disconnected Signal Messenger. These features will allow users in remote areas to register themselves in the Signal network and initiate conversations with their contacts. Additionally, the ability to receive messages from other users in the Signal network will enable users in remote areas to stay connected with their communities and receive important updates and information.

1.2 Goals

By creating the underlying infrastructure for a disconnected data distribution network, the project aims to provide a reliable and sustainable solution for internet connectivity in remote areas. This infrastructure will be critical in enabling the transfer of internet data between connected and disconnected areas using host phones

The development of applications and libraries that enable users to receive internet

data from disconnected areas is also a critical aspect of the project. These applications will need to be designed to work seamlessly with the DDD network and will need to make use of technologies like WiFi Direct to enable the transfer of data between host phones and end-users.

Developing a disconnected version of the Signal messaging app presents a considerable challenge due to its current direct reliance on Internet communication. The aim is to enable data transfer over DDD and maintain privacy while still communicating with the Signal servers. By creating such a version of Signal that can function in a disconnected setting, users in remote areas will be able to communicate securely and reliably, even when traditional network infrastructure is unavailable.

Finally, implementing the Signal protocol in an offline environment will pose a substantial technical hurdle since the private keys utilized for end-to-end encryption must remain on the phone to safeguard the privacy of the transmitted data over the network. Ensuring that end-to-end encryption is maintained while data is being transmitted over the DDD network will require careful attention to security and encryption protocols.

CHAPTER 2

Related Work

This section provides an overview of past and current research in the field of data distribution and data transport. It also discusses several networks that operate in disconnected environments. Finally, it covers various security and encryption algorithms that are used or relevant to the research.

2.1 Using DDD for sending internet messages

Internet Messaging (IM) is a communication medium, where two or more people communicate with each other over the internet. Numerous internet messengers exist today like WhatsApp, Facebook Messenger, Snapchat, and Telegram. However, these messengers are not open source, and the original source code is not made freely available for other developers to work upon. Open Whisper Systems, founded by Marlinspike, created Signal Messenger [10] in November 2015. Signal Messenger is an open-source instant messaging application. Signal Messenger uses Signal Protocol to provide end-to-end encrypted messaging and calling which is now used by most messaging applications today [11]. An application based on Signal messenger which wraps around the protocol is developed as a part of this project which can help users communicate with each other securely without being connected to the internet using our DDD architecture.

2.2 Data Transfer

One way to resolve the internet connectivity problem is to examine it as a data transport problem. There are a lot of techniques available to transfer data wirelessly between mobile phones without the Internet. Haartsen and Mattisson [12] eliminated the use of cables to transfer data between mobile phones and computers by creating a cheap, power-efficient radio chip for wireless connectivity called Bluetooth. Walton [13] had a similar approach while creating RFID (radio frequency identification)

which is used by Near Field Communication (NFC) technology to transfer data over short-range without the use of cables between two electronic devices. Even though revolutionary for their time, these techniques are now outdated, slow, and functional only in short-range.

A different area of investigation on data transport focuses on high-speed data transfer. Cao and Yin [14] explored a more specific scope where they tried to transfer data from mobile devices using USB OTG technology. Similarly, other techniques to transfer data from the phone using USB cables and SD cards require additional hardware to transfer data. Though these techniques transfer data at high speed, additional hardware makes them inconvenient.

Another solution is to use Wi-Fi Direct [9], where data transfer takes place by establishing a Wi-Fi connection between the devices through a mobile hotspot available in all modern smartphones. Using this technology, it is possible to circumvent the obstacle of fast data transfer without using additional hardware in the form of SD cards or USB cables. The range of data transfer can also be effectively better compared to Bluetooth and NFC as the Wi-Fi hotspot covers a larger area. Thus, the limitations of short-range and slow file transfer speed using NFC and Bluetooth and the inconvenience of additional hardware can be addressed by using Wi-Fi Direct as a tool for data transport. Smartphones can rapidly share data with each other using Wi-Fi Direct and create a disconnected data network without being connected to the internet. Thus the DDD architecture leverages Wi-Fi Direct to transfer data from the user's phone to the host phone.

2.3 Infrastructure

The current internet infrastructure is host-centric. Information-centric networking (ICN) [15] introduces uniquely named data wherein data becomes independent of

the underlying applications making it easy to cache and replicate. Content-Centric Networks (CCN) [16] also work in a similar manner where content is made directly addressable and routable by uniquely naming content instead of the address. DDD architecture can make use of these techniques to distribute the bundled data amongst mobile phones and in turn, make it available to a bundler server (host-centric internet gateway).

Mesh networks [17] connect multiple different nodes (mesh routers or clients) together and route data to and from clients efficiently. The nodes in a mesh network dynamically self-organize and self-configure to collectively take responsibility for delivering the data hence providing better efficiency and reliability and reducing installation overhead.

A more decentralized approach can be seen in wireless ad hoc networks or mobile ad hoc networks (MANETs) [18]. Nodes can be removed or added anytime and there is no concrete structure or certain paths laid while setting up the nodes. Nodes dynamically forward data based on the network connectivity and routing algorithm in use. Opportunistic Networks [8] are an evolution of MANET where Opportunistic networks don't assume that there always exists a path from source to destination. Opportunistic networks try to find out patterns between the nodes to perform routing and data sharing. Considering the current rise in the adaption of mobile devices, DDD focuses on adapting to these networks for better routing and delivery of our bundles in areas not connected to the internet.

The problem of continuous connectivity is planned to be solved by a version of Delay Tolerant Networks (DTN) [7] architecture specially designed for modern smartphones. The routing protocol will be a one-hop "store and forward" approach where data is moved closer to our bundler server through a host phone.

2.4 Data Distribution

Sneakernet is an informal term used in the computer industry for transporting data using physical drives such as hard drives, USBs, floppy disks, or compact discs (CDs) rather than transferring it on a computer network like the internet [19]. Sneakerternets are used when data transfer is impractical due to connectivity, or bandwidth issues, or if the data is isolated. Google has used a sneakernet to transport large datasets from the Hubble Space Telescope [20]. Google Cloud gives a feature to their customers where they can import data into the cloud using sneakernet [21]. Data has different values at different times. It is not possible to transport time-sensitive information such as news through the sneakernet as by the time it might be received by the recipient it might not be useful.

A narrower scope was researched by Petland et al. in DakNet [4], a wireless network provider for rural areas of India and Cambodia. It is an ad hoc network that uses wireless technology to provide asynchronous digital connectivity. DakNet transmits data over a short point-to-point link between kiosks and portable storage devices. The Daknet project was implemented with a kiosk point in each village, as depicted in Figure 2. Villagers would visit this point to access all the updates. Apps developed for DakNet allowed villagers to access government and commercial services. Using this approach, the recipient might receive the data in comparatively less time than the sneakernet.

Parallel to DakNet, the Wizzy Digital Courier [5] is a project to distribute data like email to places with no Internet connection, via a USB flash drive. In this project, the data which is normally carried by dial-up internet connections was instead physically carried using USB sticks between user locations and high bandwidth drop-off points that were connected to the internet. Both the Wizzy Digital Courier and DakNet initially worked very well but lacked sustainability. The use of additional

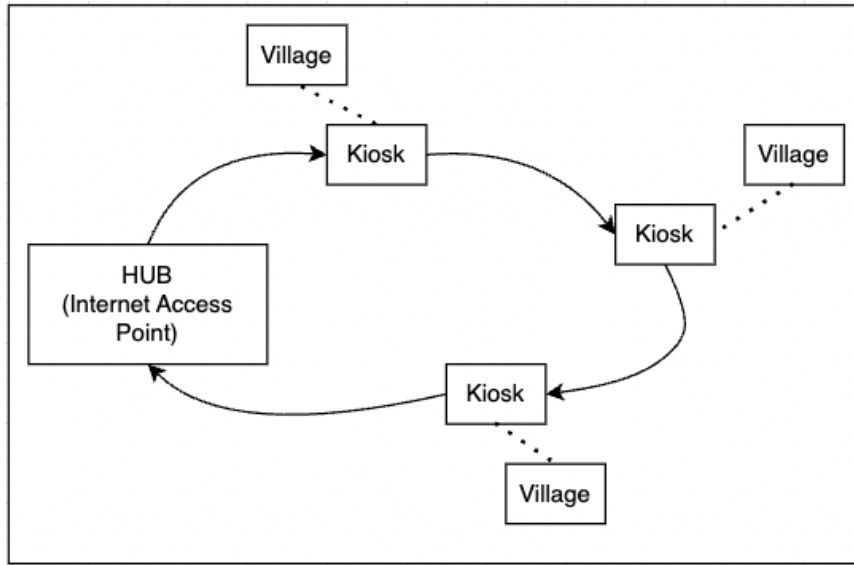


Figure 2: Daknet

infrastructure to transport the data leads to a lot of inconveniences, management, and upfront investment. Also, physical assistance from humans was involved in various steps of data transportation and distribution.

Shah et al. [6] go beyond the traditional infrastructure used for data transportation by exploiting the presence of mobile entities called MULEs which pick up data from the sensors at close range, process it, and then deliver it to wired access points. This completely eliminates human assistance by using MULEs to perform the task of data collection and delivery. This approach has carved a path for further research that focuses on data distribution without human assistance and with no additional infrastructure to transport and distribute Internet data.

Recently, Apple released a new update for its "Find My" application which enabled users to locate the devices not connected to WiFi or cellular or even those which are switched off [22]. This feature leverages Bluetooth [12] and proximity to other Apple devices to relay the lost device's location. When a lost device is offline but in close proximity to another device, it can connect to that nearby device over

Bluetooth, which can then relay its location information to the user. This feature greatly enhances the trackability of lost devices, increasing the likelihood that they can be found and retrieved.

The architecture of DDD uses smartphones as MULEs [6] and the Bundler application as a way of communicating with other nodes (mobile phones) in the network without human intervention. Similar to Apple's "Find My" application which uses Bluetooth to relay device information, the Bundler application uses Wifi Direct to transport bundles between devices.

2.5 Security and Encryption

Security is a critical concern in any network, but it becomes even more important in a disconnected network like DDD where multiple untrusted parties are involved in data transmission. The Signal protocol is known for its strong encryption and privacy features, and by building on this protocol, the DDD network can ensure that messages sent between users remain secure and protected from unauthorized access or manipulation.

In public key cryptography (or asymmetric cryptography), there are two keys namely the public key and the private key. These keys are generated by a certain cryptographic algorithm. Anyone can encrypt data using the public key but the same data could be decrypted only by its corresponding private key. These keys need to be exchanged with other clients in order to provide encryption of messages over untrusted hosts (phones) and servers. One of the first and most common algorithms for key exchange is the Diffie-Hellman (DH) Key Exchange algorithm [23], which securely exchanges keys through untrustworthy communication mediums. In its simplest form, the protocol uses the multiplicative group of integers modulo p , where p is a prime number and g is a primitive root modulo p . The possibilities that a shared secret key

can take is from 1 to $p-1$ based on these two values.

One variant of Diffie-Hellman is the Elliptic-curve Diffie-Hellman (ECDH), with the main difference being the group that is being chosen to compute the secret keys [24]. DH uses a multiplicative group of integers modulo a prime number whilst ECDH uses a multiplicate group of points on an elliptic curve. This technique is derived from Elliptic-curve cryptography (ECC). One of the fastest ECC curves [25] which is not covered in any patents and the reference implementation is public domain software is the Curve25519 elliptical curve. It offers 128 bits of security (256 bits key size) and can be perfectly integrated with ECDH. During registration, the Signal Protocol uses ECDH with Curve25519 in its Extended Triple Diffie-Hellman (X3DH) Key Agreement Protocol [26].

2.5.1 Signal Protocol

Signal Protocol is a state-of-the-art encryption protocol that provides end-to-end encryption for secure communication. It was developed by Open Whisper Systems, which is now a part of Signal Messenger LLC. The protocol is designed to ensure that messages, voice and video calls, and other forms of communication remain private and secure between the sender and the intended recipient. Signal Protocol is widely recognized as one of the most secure and reliable encryption protocols available, and it has been adopted by many popular messaging and communication platforms, including WhatsApp, Signal, and Facebook Messenger. Signal Protocol consists of X3DH protocol [26] for shared key generation, Double Ratchet Algorithm [27] to establish secure communication channels between the sender and recipient, and finally the Sesame protocol [28] for session management.

Using the X3DH protocol, a shared secret key (SK) between two parties is established who want to communicate with each other. The EC keys involved in

this protocol are Identity Key (IK), Ephemeral Key (EK), PreKey (PK), and Signed PreKeys (SPK). If Alice wants to communicate with Bob using X3DH, Alice should fetch the PK bundle of Bob from the Signal server. It is assumed that Bob has published IK_b , SPK_b , prekey signature $Sig(IK_b, Encode(SPK_b))$ and a set of one-time prekeys ($OPK_{b1}, OPK_{b2}, \dots$) during the time of his registration. Alice checks the PK signature of Bob and aborts if the verification failed. Alice then generates an EK pair with public key EK_a . If the bundle in Figure 3 does not contain OPKs, the shared secret is calculated by her in the following way:

$$DH1 = DH(IK_a, SPK_b)$$

$$DH2 = DH(EK_a, IK_b)$$

$$DH3 = DH(EK_a, SPK_b)$$

$$SK = KDF(DH1 \parallel DH2 \parallel DH3)$$

If the bundle does contain OPKs of Bob, then the calculation gets modified to add another DH operation:

$$DH4 = DH(EK_a, OPK_b)$$

$$SK = KDF(DH1 \parallel DH2 \parallel DH3 \parallel DH4)$$

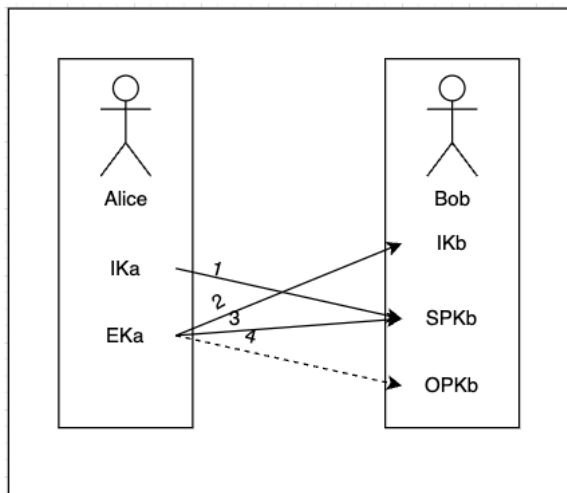


Figure 3: X3DH Asymmetric Key Generation Algorithm.

The next step according to Signal protocol is the exchange of encrypted messages with the use of the newly created shared secret key which is done by the Double Ratchet algorithm [27]. The term "Double Ratchet algorithm" refers to the encryption technique that employs two types of ratchets: the Symmetric-key ratchet and the Diffie-Hellman ratchet. The algorithm's key feature is the KDF chain, a cryptographic function that takes input data and the KDF key and returns output data. Each message is encrypted using a message key generated by the Symmetric Key ratchet, with a constant and chain key serving as input. The constant is included for break-in recovery purposes.

The Double Ratchet combines the Symmetric key ratchet with the Diffie-Hellman ratchet to provide forward secrecy. Each party generates its own DH key pair. While communicating with other parties they send their public key in the header info of the message, on the receiver side they perform a DH ratchet step with a private key of their own, and the new public key. As a part of the DH ratchet step, the receiving side updates its DH key pair and uses this for all further communication. Due to this phenomenon, after each sending and receiving cycle DH keys are changed which results in a ping-pong behavior.

The illustration in Figure 4 demonstrates the practical application of the double ratchet algorithm, with Alice initiating communication to Bob. The algorithm begins by Alice receiving a Prekey bundle containing Bob's Identity Key, One-time Prekey, and Signed Prekey. From this information, Alice creates a Shared Secret Key (SSK). Alice then combines Bob's Diffie-Hellman (DH) public key with her own DH private key to generate a DH shared secret. This secret is then passed to the root KDF function along with SSK to produce a root key (RK) and a chain key (CK). The CK is used in the Symmetric-key ratchet step where the KDF function is utilized with the message, and the output is the encrypted message (A1) and the new CK. Bob

then performs a similar ratchet operation on his end to decrypt the message.

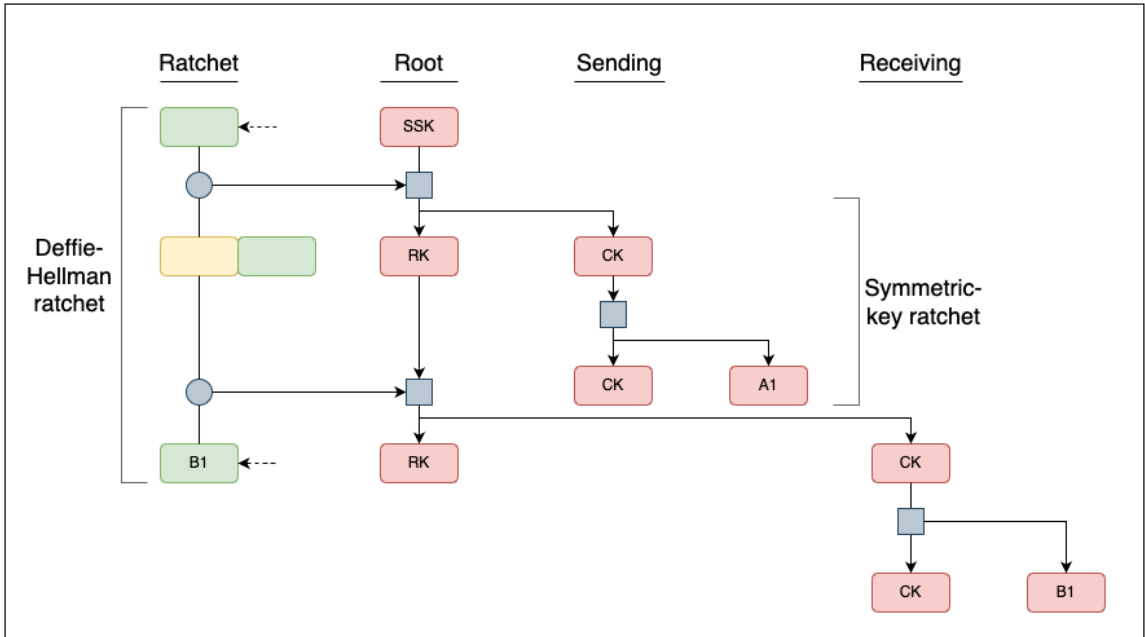


Figure 4: Double Ratchet Algorithm

Lost and out-of-order messages are also handled in double ratchet by maintaining two counters in the header message of each message. Each message has information about the length of the previous message chain and its current message number in sending chain. Using these two counters, the double ratchet stores the keys which are responsible for decoding these messages.

Messages encrypted using the double ratchet algorithm generated with X3DH key agreement protocol are managed with the Sesame session management algorithm [28]. Sesame is a generic session management protocol and can be used in DDD with certain changes specific to our disconnected data network. Sesame generates a user id for each participant which is either a username or a phone number. Participants can have multiple devices each identified by a unique device id. Servers hold data regarding all participants' user id and corresponding active device ids. Figure 5 illustrates how a session is created between two device IDs, which is identified by a session ID.

Each participant has a mailbox on the server where they can receive messages. If a participant wants to initiate a conversation, they can retrieve the user's device ID mapped to their user ID from the Signal server. They can then send an initial message that contains the sender's IK, user ID, device ID, and public key in the message header.

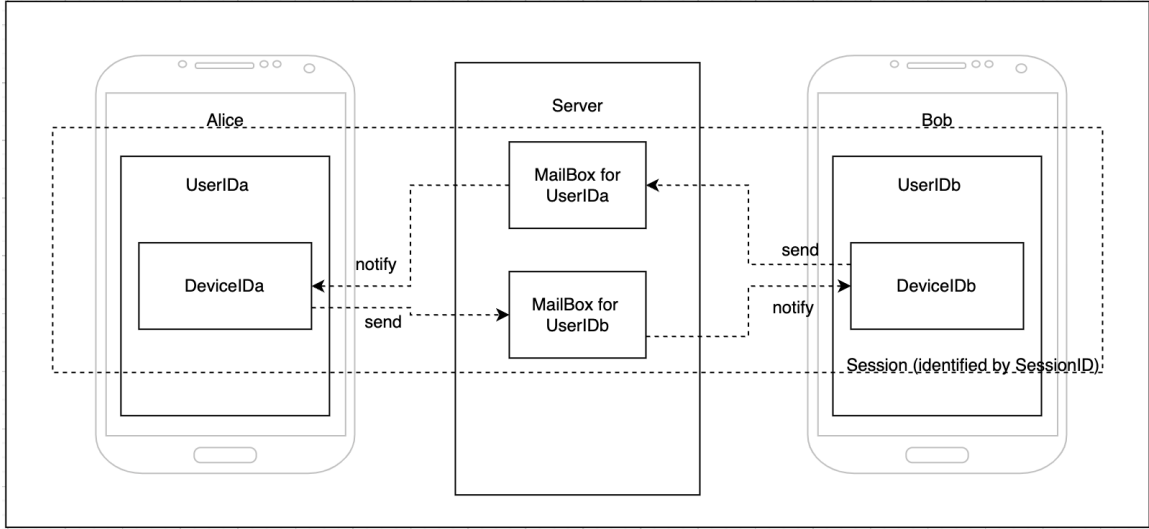


Figure 5: Sesame Algorithm for Asynchronous Session Management.

One of the most important components is establishing trust between the Bundler Signal Service and the Signal server, as it is going to directly communicate with the Signal server. Making HTTP requests is not safe as data is not encrypted during the transmission. HTTPS protocol is built upon HTTP which provides encryption and verification. HTTPS uses TLS (SSL) to encrypt normal HTTP requests and responses, and also digitally signs them.

All communication between the bundler server and the signal server is going to take place using a Secure Sockets Layer (SSL) [29] which in a way establishes an encrypted link between a server and a client. SSL is more widely known as Transport Layer Security (TLS). When a client tries to connect to the server which is secured

by SSL, the client and server establish an SSL connection using the process called “SSL Handshake”. Three keys take part in this handshake public, private, and session keys. The public and private keys are used to create only used to create a symmetric session key which is then used to encrypt all the transmitted data.

Today, most organizations have a Public Key Infrastructure (PKI) [30] to govern the encryption keys and management of digital certificates. PKIs are commonly used in websites to hold SSL certificates so that site visitors know they are communicating with an intended recipient. A similar requirement of managing the public keys of participants and their association with user id is done on the Bundler Signal Service. For every registration request, the Bundler Server is going to create a new user id (phone number) manually or through the help of tools and services like Twilio, Google Voice, etc., and associate the keys with this id. All further communication with Signal servers will be with this generated user id.

Overall, by using all these techniques - creating secret keys using X3DH, signing keys with ECDH with Curve25519, communication with Double Ratchet protocol, session management using Sesame, and requests between servers with HTTPS and PKI-like entities to manage keys and association with user ids DDD is implementing a strong security framework for its communication systems, which should help to protect the confidentiality, integrity, and authenticity of its messages.

CHAPTER 3

System Overview

The focus of this section pertains to the DDD architecture, registration, and message-receiving sequence flows, with an in-depth examination of how the Signal Protocol's key pairs are both stored and transferred and which components have visibility to specific parts of the keys. The use of sequence diagrams further clarifies how information flows through the entirety of the system during registration and messaging.

3.1 DDD Architecture

This section discusses the DDD architecture used to develop the disconnected version of the Signal Messenger. It briefly talks about all the different components that are involved in DDD and how they work together to enable users to register themselves in the Signal network and participate in messaging with other registered users, as depicted in Figure 6. The main focus of this section is to discuss the secure transmission of disconnected user data between components, ensuring privacy.

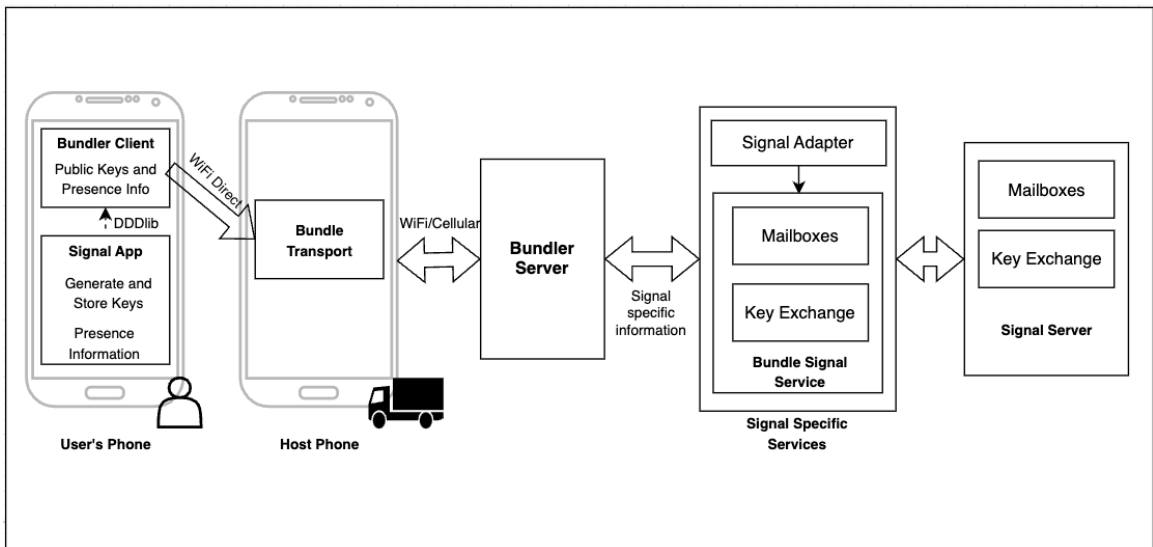


Figure 6: DDD for Signal Messenger

- Signal App

As a part of this research, a wrapper over the original Signal application is created from which users can register themselves in the Signal network and participate in messaging with other registered users. The Signal application will be responsible for creating and storing keys and the offsets required for X3DH [26] and Double Ratchet Algorithm [27] and also other keys required by the Sesame Protocol [28].

- DDDlib

DDDlib will be installed as a part of the DDD framework. It is the official library of DDD which contains a set of APIs that can perform multiple different operations like fetching data from DDD-based applications or sending data to DDD-based applications.

- Bundler Client

Bundler Client is an Android application dedicated to doing all the heavy-lifting jobs such as creating bundles, encrypting them, and then transporting the encrypted bundles to the host phone. The Bundler Client application will reside on the user's phone. All disconnected applications using the DDD framework will have a dependency on the Bundler Client to send the bundles to the host phone.

- Bundle Transport

All the information from the disconnected applications will be encrypted and transformed into bundles which will then be sent to the Bundle Transport application present on the host phone through WiFi Direct [9] once the host phone comes within range of the user's phone. Bundle Transport stores all the user's bundles on the host phone, bundles are encrypted and thus immune to

man-in-middle attacks.

- Bundler Server

Once the host phone comes into an internet-connected area, all the bundles will then be transported to the Bundle Server using cellular data or WiFi. The bundle server is responsible for decrypting the bundles and then routing them to the correct service adapter based on the decrypted data.

- Signal Service Adapter

The DDD architecture is designed to support various internet services. To accommodate each disconnected application, an adapter service is provided, to which the bundler server will send the application-specific data. In the case of the Signal Messenger, a Signal Service Adapter is established as a gateway for all Signal-related requests within DDD.

- Bundler Signal Service

In DDD, all the disconnected applications will have their own service to interact with the actual application services. In the disconnected version of Signal, a Bundler Signal Service routes all the requests from the Signal Service Adapter to the actual Signal server. Other than routing requests, it will also contain mailboxes for each UserID which are used to store the encrypted receiving messages and databases to store information related to keys, offsets for pre-key bundles, contacts, and other passwords related to each Signal account.

- Signal Server

Signal Messenger [10] maintains a centralized server on which encrypted messages between two parties are transmitted. Signal servers also facilitate the discovery of contacts who are also registered on Signal. If the user wants to communicate with one of their contacts, they can get the public keys and Prekey bundle from

the Signal server to start the X3DH protocol and decide upon a secret shared key.

In conclusion, the DDD architecture is a robust framework that enables the disconnected version of Signal to operate seamlessly. All the components work in tandem to ensure that users can register themselves in the Signal network and participate in messaging with other registered users. The Signal Server facilitates the discovery of contacts and enables encrypted message transmission between two parties. Overall, the DDD architecture offers an innovative solution to the challenges associated with operating a disconnected version of a messaging application like Signal.

3.2 Keys and IDs Generated for Registration and Session Management

Table 1 provides an overview of the various keys and IDs used in the Disconnected Signal Messenger. These keys and IDs play an important role in ensuring secure communication between users on the Signal Protocol. Each key or ID has a specific purpose, and their usage is associated with different components of the Signal Protocol. This table presents a summary of these keys and IDs, their descriptions, how they are used, and when they are generated, providing a comprehensive guide to understanding the technical aspects of Signal's secure messaging system.

Table 1: Keys and IDs in Disconnected Signal Messenger

Title	Description	Usage	Generated at
Identity Key (IK)	Unique and constant for the user.	X3DH	Installation
Signed Prekey (SPK)	Periodically changing (e.g., weekly/-monthly) and signed with IK.	X3DH	Installation
One-time Prekeys (OPKs)	Disposable and they get deleted from the server each time some user requests a public key bundle from the server.	X3DH	Installation
Ephemeral Key (EK)	User needs to update this key periodically on the phone.	X3DH	Installation
Secret Key (SK)	The result of X3DH for each session. Keeps changing after each ratchet.	Double Ratchet	X3DH
DeviceID	Uniquely identifies each device in a session.	Sesame	Installation
UserID	Username or phone number identifying each user in the Signal Protocol.	Signal Protocol	Bundler Signal Service
Message Key	Every message sent or received is encrypted with a unique message key. The message keys are output keys from the sending and receiving KDF chains.	Double Ratchet	Symmetric-key ratchet
Account Identification (ACI) and Phone Number Identification (PNI)	Unique IDs are generated by Signal to identify an account and a phone number associated with that account. Every registered user has ACI and PNI which are used to establish identity with server	Signal Protocol	Signal Server

3.3 Registration and Session Management

Ensuring user data privacy is a primary goal of DDD. Figure 7 illustrates how keys are securely transferred throughout the architecture and outlines the necessary steps to register a user on Signal from a disconnected area. Additionally, it highlights how users can initiate conversations with their Signal contacts while maintaining data

privacy.

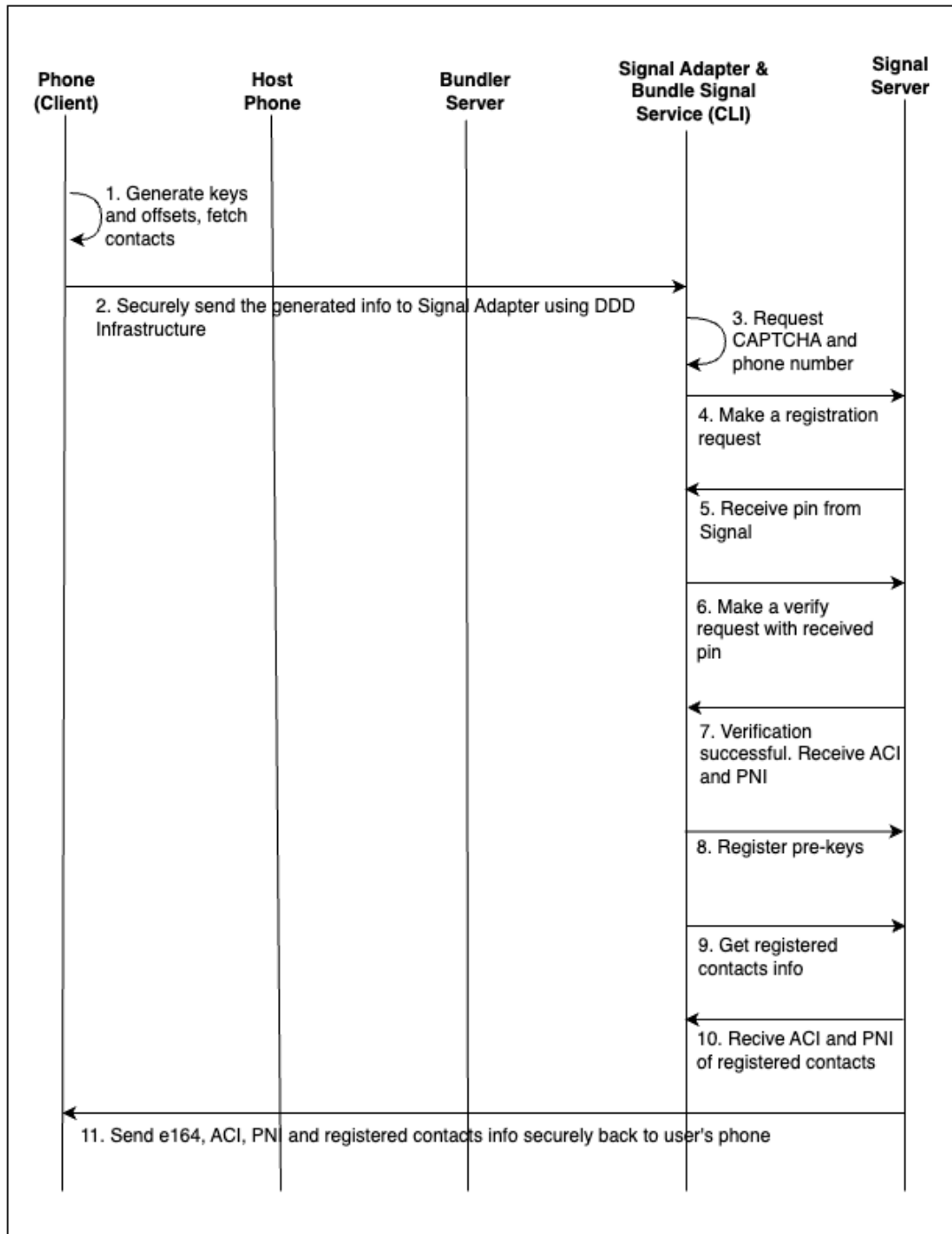


Figure 7: Sequence Diagram of Registration

3.3.1 Client Side Operations

A disconnected user can join the Signal network by installing the disconnected Signal Android application, which is built on the open-source Signal-Android Messenger. Once installed, users initiate registration on the app, generating and storing necessary keys on their phone. The public keys are serialized and stored in a JSON file along with contacts, which are fetched with user permission. The JSON file is then sent to the Bundle Client app and transported through the DDD infrastructure. Private keys used to sign and decrypt messages remain on the user's phone, ensuring message privacy. Steps 1-2 in Figure 7 denote the client-side operations.

3.3.2 Registration

After the registration bundle reaches the server, it is directed to the Signal Adapter, which serves as an intermediary between the Bundler Server and the Bundle Signal Service. The Bundle Signal Service is a Command Line Interface used to manage the state of disconnected users, including key and contact-related information, and message mailboxes for each phone number. It also creates payloads that interact with Signal Servers. During registration, Signal requires a phone number that can receive a one-time verification pin. However, since disconnected users do not have access to their phone numbers, a new one is created using tools like Google Voice. Signal has implemented a security measure requiring a CAPTCHA during the initial contact with Signal servers. This CAPTCHA can be generated from the Signal website and can be added to the headers of the request. After generating a phone number and CAPTCHA, a registration request is made, resulting in the phone number receiving a pin that has limited validity. The registration operation on Signal Adapter is represented by Steps 3-5 in Figure 7.

3.3.3 Verification

After receiving the verification code on the phone number, the operator will add this number to a "verify" request through the CLI (Bundle Signal Service) to the Signal Server. The request includes only two parameters - the E164 formatted phone number and verification code. If the verification request is successful, the phone number is registered on the Signal network and can be used to communicate with other registered users. Signal uses two unique byte arrays, ACI (Account Identifier) and PNI (Phone Number Identifier), to identify each user for communication purposes.

For all subsequent requests after registration, ACI and PNI must be included in the headers to authenticate with the Signal Server. After successful verification, the next step is to register all the Prekeys initially generated on the client's phone. Each account must register the public part of the Identity Key, Prekey, and Signed Prekey. Once Prekeys are registered, other users can use the user's ACI, PNI, Identity Key, Signed Prekey, and one Prekey of the Prekey bundle to establish a shared secret using the X3DH algorithm [26]. In Figure 7, steps 6-8 illustrate the verification process in Signal registration.

3.3.4 Finding Contacts in Signal Network

After registration, the next step is to determine which of the client's contacts are also registered on Signal Messenger. The received bundle contains all the contacts from the client's phone. Using the Bundler Signal Service, a "getRegisteredUsers" request is made, which iterates over all the phone numbers provided by the client and returns the PNI and ACI of the registered users. The Bundler Signal Service generates a JSON message with all the contacts registered on Signal and their ACI and PNI, which is then sent back to the user. This provides a good identification measure to check which of their contacts are registered on Signal.

Finally, the response information, including registered contacts, E164 number, ACI and PNI of the user, is populated in a JSON message and sent to the Bundler Server via the adapter, which then sends it to the user's phone through the DDD infrastructure. Once the user receives this information, they can start sending and receiving messages from their contacts. Steps 8-11 in Figure 8 describe how Signal determines if a contact is registered on the Signal network and about what information is sent back to the user.

Overall, the Signal registration process for disconnected users involves several client-side operations, including generating and storing keys, fetching contacts, and creating a registration bundle that is transported through the DDD infrastructure. After the bundle reaches the server, it goes through the Signal Adapter and Bundle Signal Service, where the phone number is verified through a CAPTCHA and a one-time verification code. Once verified, the user's contacts are checked for registration on the Signal network, and the response information is sent back to the user. Signal's registration process provides a secure and privacy-focused way for disconnected users to join the network and communicate with others.

3.4 Receiving Messages

The ability to receive messages while in a disconnected area is a crucial feature of Signal in DDD, and the message receiving flow facilitates this functionality. This flow involves the transmission of information to the client without requiring an explicit request from the client. Figure 8 illustrates the flow of information through the infrastructure for receiving messages in disconnected Signal.

3.4.1 Initiate Receiving

When a host phone intends to visit an area with no network connectivity, it sends a request to the Bundler Server to obtain all the bundles that need to be delivered to

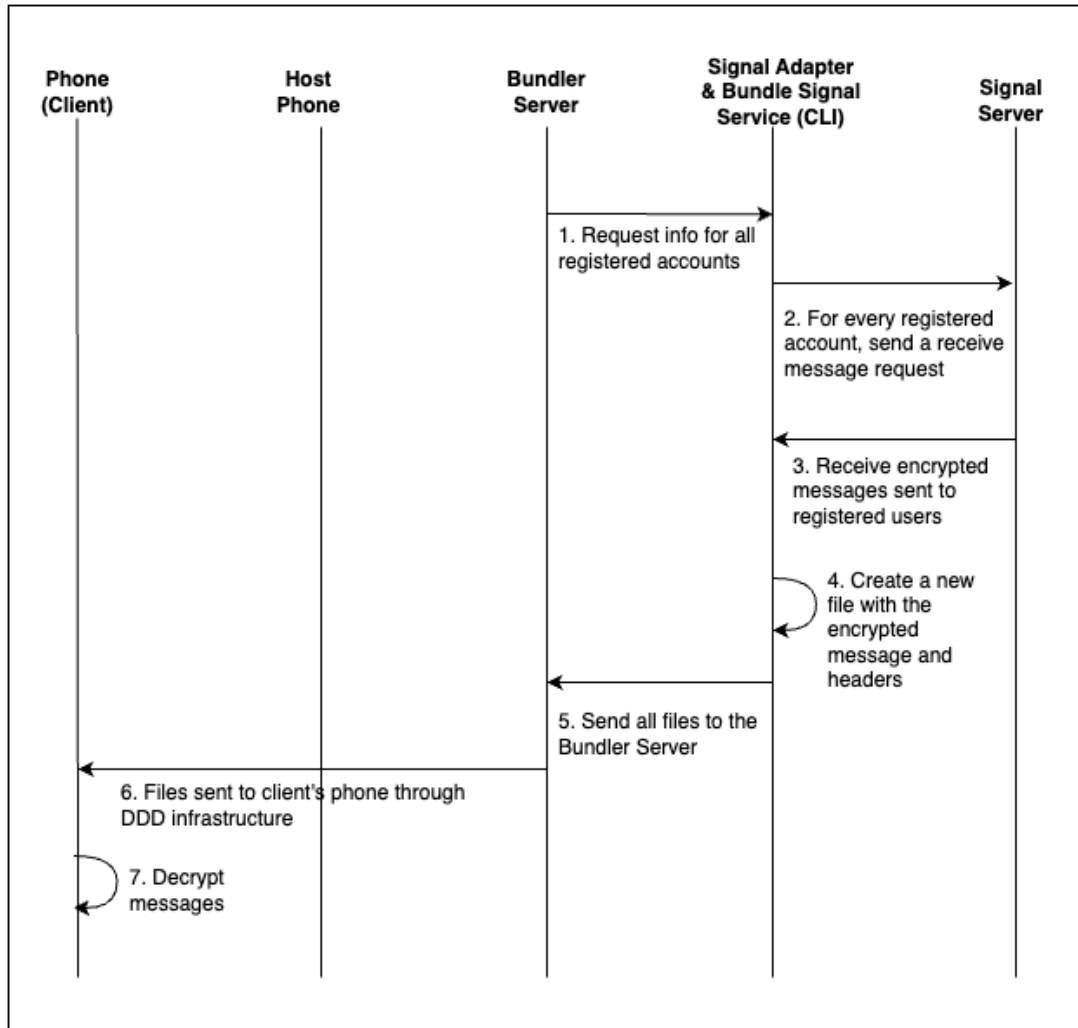


Figure 8: Sequence Diagram of Receiving Messages on Signal in DDD

the disconnected area. The Bundler Server then initiates a gRPC request known as `prepareData` to all adapters, seeking the information to be delivered to clients. As Signal Messenger is presently the prototype, the Signal Adapter receives a `prepareData` request to sync all Signal-related information with the client. Step 1 in Figure 8 represents the initiation stage.

3.4.2 Receiving Messages

Upon receiving the prepareData request, the Signal Adapter proceeds to request the Bundler Signal Service (or Signal CLI) to execute the receive command for all registered users. The Signal CLI attempts to retrieve messages from the Signal Server for all disconnected users registered on Signal through DDD. The messages for each user are stored in mailboxes on the Signal Server. All the user's messages are encrypted using Double Ratchet Algorithm [27]. The messages is stored in an envelop which contains headers with information about timestamps related to the message, sender's phone number, sender's ACI and a message body.

However, these messages cannot be decrypted on the Bundler Signal Service, since the user has not shared their private keys with the server. To decrypt the messages, the user needs their Identity Key, Signed Prekey, and Prekey Bundle with offset, which were used to encrypt the messages. Steps 2 and 3 in Figure 8 illustrate the process of triggering the receive command on the Signal CLI.

3.4.3 Storing Messages

The Bundler Signal Service assigns a unique ID to each account for identification purposes. Using this ID, the service creates directories to store the account's databases and messages. As illustrated in Figure 10, each message is stored as a separate file within the directory. The message file includes all the essential information required for deserializing the envelope on the client side and initiating decryption. Step 4 in Figure 8 demonstrates the message file creation step.

3.4.4 Sending Message Files to Disconnected Clients

After all messages are received, the generated files are sent back to the adapter, which in turn sends them to the Bundler Server. Once received, the server packs the files into encrypted bundles and sends them to the host phone that initiated

the prepareData request for the disconnected area. Upon reaching the disconnected area, the host phone sends the bundles to the client's phone through Wi-Fi Direct using the Bundler Client application. The Bundler Client then decrypts the bundles using APIs from DDDlib, which are integrated within the application, and sends the messages to Signal Messenger. The messages are decrypted using the keys created during registration based on the Double Ratchet Protocol [27]. Finally, the decrypted messages can be viewed and replied to by the user.

CHAPTER 4

Implementation

This section discusses the integration of DDD-specific changes into the client application, as well as the implementation of the Bundler Signal Service.

4.1 Signal Android

The Disconnected Signal Messenger is an Android app that is derived from the open-source Signal Android app. It can be installed on any Android device without requiring any complicated setup procedures. All cryptographic operations in the disconnected messenger are handled by the lib-signal-java library. The main objective of the Disconnected Signal Messenger application is to perform cryptographic operations such as creating the necessary keys, serializing them into JSON, and encrypting and decrypting messages using those keys. Additionally, it communicates with the Bundle Client application to facilitate the sending and receiving of encrypted message bundles.

Upon launching the application, the user is prompted to grant permission to access the contacts. Upon granting access, the "DDDKeys" class is used to generate all the necessary keys. The SecureRandom algorithm is used to generate random offsets. The elliptic curve algorithm [24] from libsignal-java is used to generate all the Identity Key pairs. One hundred Prekeys are generated and stored in the prekey bundle. A Signed Prekey is generated by creating a new key-pair and signing it with the private Identity Key. All of these keys are necessary for establishing presence on the Signal server. Using the public Identity Key, Signed Prekey, and one key from the prekey bundle, other users can create a shared secret key with the disconnected user using the X3DH [26] algorithm. Messages can be encrypted based on this shared secret key and the Symmetric ratchet and Diffie-Hellman ratchet as part of the Double Ratchet Protocol [27]. Figure 9 illustrates the attributes of the DDDKeys class along

with the constructor used to populate these attributes.

```
public class DDDKeys {  
  
    private int preKeyIdOffset;  
    private int nextSignedPreKeyId;  
    private int pniPreKeyIdOffset;  
    private int pniNextSignedPreKeyId;  
    private IdentityKeyPair identityKey;  
    private IdentityKeyPair pniIdentityKey;  
    private SignedPreKeyRecord aciSignedPreKey;  
    private SignedPreKeyRecord pniSignedPreKey;  
    private List<PreKeyRecord> preKeys;  
    private List<PreKeyRecord> pniPreKeys;  
  
    public DDDKeys() {  
        this.preKeyIdOffset = new SecureRandom().nextInt(Medium.MAX_VALUE);  
        this.nextSignedPreKeyId = new SecureRandom().nextInt(Medium.MAX_VALUE);  
        this.pniPreKeyIdOffset = new SecureRandom().nextInt(Medium.MAX_VALUE);  
        this.pniNextSignedPreKeyId = new SecureRandom().nextInt(Medium.MAX_VALUE);  
        this.identityKey = generateIdentityKeyPair();  
        this.pniIdentityKey = generateIdentityKeyPair();  
        this.aciSignedPreKey = generateSignedPreKeyRecord(identityKey, nextSignedPreKeyId);  
        this.pniSignedPreKey = generateSignedPreKeyRecord(pniIdentityKey, pniNextSignedPreKeyId);  
        this.preKeys = generatePreKeyRecords(preKeyIdOffset, 100);  
        this.pniPreKeys = generatePreKeyRecords(pniPreKeyIdOffset, 100);  
    }  
}
```

Figure 9: DDDKeys Attributes and Constructor

Once the necessary keys are created, they are serialized into JSON and stored within the Signal application. This includes the public Identity key, Prekey bundle, Signed Prekey and a list of all the contacts available on user's phone. The serialized JSON is then transferred to the Bundle Client application using Android Intent. The disconnected Signal application will wait for the registration response from DDD which contains the E164 phone number, ACI, and PNI of the disconnected user, as well as the necessary contact information for registered Signal users.

Overall, the Disconnected Signal application plays a crucial role in enabling communication in disconnected settings. By generating the necessary keys and facilitating encrypted message transfer, it ensures secure communication between users without requiring an internet connection.

4.2 Bundler Signal Service

The Bundler Signal Service is a Command Line Interface (CLI) designed for the disconnected Signal messenger in the DDD architecture. It serves as the sole

communication channel with official Signal servers and acts as the entry point to Signal. The service performs various complex operations like generating payloads required for Signal requests, managing and storing users' data including keys, contacts, and more.

The Bundler Signal Service is designed as an executable JAR file that can be deployed on any machine with Java 17 or higher. As part of the current DDD infrastructure, it is being hosted on a Linux-based machine along with the Signal Adapter.

4.2.1 Operations

The Signal CLI or the Bundler Signal Server is designed to perform the following operations:

- **Register**

Register a phone number with SMS. It requires three parameters:

- **a**: The E.164 formatted phone number.
- **ddd**: A JSON file containing all the public keys and offsets required to set up an account on Signal. This file is created on the client's phone and then transferred through DDD to Bundler Signal Service.
- **captcha**: The captcha result obtained from the Signal website.

During the registration process, the first command executed from the CLI is "register". The command attempts to parse the "-a" parameter, and if it detects a new account, it generates a unique ID and stores it in the accounts.json file. Based on this ID, a corresponding file and directory are created in the Signal project for maintaining the user's state, as shown in Figure 10. Next, the CLI parses the second parameter, which is the keys file generated on the client side. The public keys are stored in the ID file, while the Prekeys are ignored until

the user is authenticated on the server. The ID file also keeps the location of this keys file for future use. The third parameter is the CAPTCHA, which is required initially to authenticate communication with the Signal servers. Signal requests a CAPTCHA code to be sent as a header with the register request.

The Signal Server has two environments, Production and Staging. The CLI uses the production environment, which is quite strict. Sending incorrect payloads, attempting to fetch messages on behalf of other users, decrypting messages with wrong keys, or retrying too frequently can result in a temporary ban from the server. After sending the register request, the Signal server will send a six-digit pin to the phone number specified in the parameters for registration.

Example:

```
./signal-cli -a +11234567890 register --ddd \location\of  
\bundle --captcha signal-captcha://.....
```

- **Verify**

Verify the number using the code received via SMS. It has two parameters:

- **a**: The E.164 formatted phone number.
- **pin**: A six-digit temporary pin received from Signal as an SMS for two-factor verification.

After receiving the pin on the phone number, it is used as a parameter for the verify command to authenticate the user on the Signal server. If the pin matches, Signal returns an ACI and PNI id, which are unique to the user. Subsequent requests to the server should include these ids in the headers to authenticate the user.

The next step is to register the user's Prekeys with the Signal server, which is optional but recommended. The location of the keys file is obtained from the

ID file, which stores the state of the user. The Prekeys and Signed prekeys are then registered with the Signal server. It is important to note that registering Signed Prekey is mandatory, as without registering one-time Prekeys, the X3DH calculation only uses three DH operations. The calculation involves deriving a shared secret key (SK) using the following operations:

$$\text{DH1} = \text{DH}(\text{IKA}, \text{SPKB})$$

$$\text{DH2} = \text{DH}(\text{EKA}, \text{IKB})$$

$$\text{DH3} = \text{DH}(\text{EKA}, \text{SPKB})$$

$$\text{SK} = \text{KDF}(\text{DH1} \parallel \text{DH2} \parallel \text{DH3})$$

If the bundle contains one-time Prekeys, then the calculation is modified to include an additional DH operation:

$$\text{DH4} = \text{DH}(\text{EKA}, \text{OPKB})$$

$$\text{SK} = \text{KDF}(\text{DH1} \parallel \text{DH2} \parallel \text{DH3} \parallel \text{DH4})$$

By using these keys, other users will be able to establish a shared secret key with the disconnected user and initiate secure communication.

Example:

```
./signal-cli -a +11234567890 verify 123456
```

- **Receive**

Query the Signal server for new messages for a registered phone number. All encrypted messages are downloaded and saved as files in a specific directory. It has one parameter:

- **a**: The E.164 formatted phone number.

When a host phone intends to bring internet data with it to a disconnected area, the receive command is triggered through the Signal Adapter to CLI for all disconnected users. Unlike other commands, the receive command employs

a ReceiveHandler that uses the Sesame algorithm [28]. All incoming messages from the user are stored in a mailbox on the server. Using this ReceiveHandler, the messages are synced to Bundler Signal Service's mailbox and transferred as Signal envelopes (Figure. 5). These envelopes contain information like sender and destination ACI, timestamp, content length, and content body, and other headers necessary for the double ratchet to operate. The messages are then received in the user's directory under the "msg-cache" directory and serialized in JSON format before being sent to the respective clients (Figure. 10). Only the private keys generated on the client's phone can decrypt these messages, providing end-to-end encryption for all messages.

Example:

```
./signal-cli -a +11234567890 receive
```

4.2.2 Dependencies

Table 2 presents the dependencies imported in the Signal-Android and Bundle Signal Service projects. Gradle is utilized as a dependency manager for the Android project, whereas Maven is utilized for the Service. The Lib-Signal library is used for generating keys and signatures. The Lib-Signal Client library has all the APIs needed for calling the Signal servers. The Lib-Signal library requires Bouncy Castle internally to carry out specific cryptographic algorithms. OkHttp is employed to interact with the server, and FasterXML is used for key serialization.

Table 2: Dependencies for Bundler Signal Service

Name	Description	Version	Artifact ID
Patched Lib-Signal Service Java	A Java library extracted from Signal Android implementation for implementing the Signal protocol including Double Ratchet, X3DH, and Sesame protocol.	2.15.3	libsignal-service-java
Lib-Signal Client	This library contains all the APIs used by the official Signal client and server.	0.22.0	libsignal-client
SQLite JDBC	A library for accessing and creating SQLite database files in Java. It is used to create and store public keys and contacts on Bundler Signal Service.	3.40.1.0	sqlite-jdbc
OkHttp	A Java library that makes HTTP requests load faster and save bandwidth. It is used to interact with Signal Server.	4.10.0	okhttp
Bouncy Castle	A crypto package that contains implementations of cryptographic algorithms.	1.70	bcprov
FasterXML	JSON parser and generator library used for creating and parsing bundle messages.	2.14.2	jackson-core
SLF4J	The Simple Logging Facade for Java (SLF4J) is a library used for showing logs, debug messages, and errors.	2.0.6	slf4j-api

4.2.3 File Structure, Key Exchange and Storage

Signal maintains the state of the disconnected user using directories and databases. Each user is assigned a unique six digit ID, which is mapped to attributes such as E164 phone number, ACI, and PNI in the accounts.json file. A file and folder with the ".d" extension is created for each ID, with the file containing necessary variables such as offsets, serialized public keys, passwords, and device IDs. The directory stores a database file that includes tables for prekeys, contact information, and more. Incoming messages are also stored in the directory as a mailbox for the user. To provide a better understanding of the file structure, details on the files and directories are provided in Table 3, and the directory structure is illustrated in Figure 10.

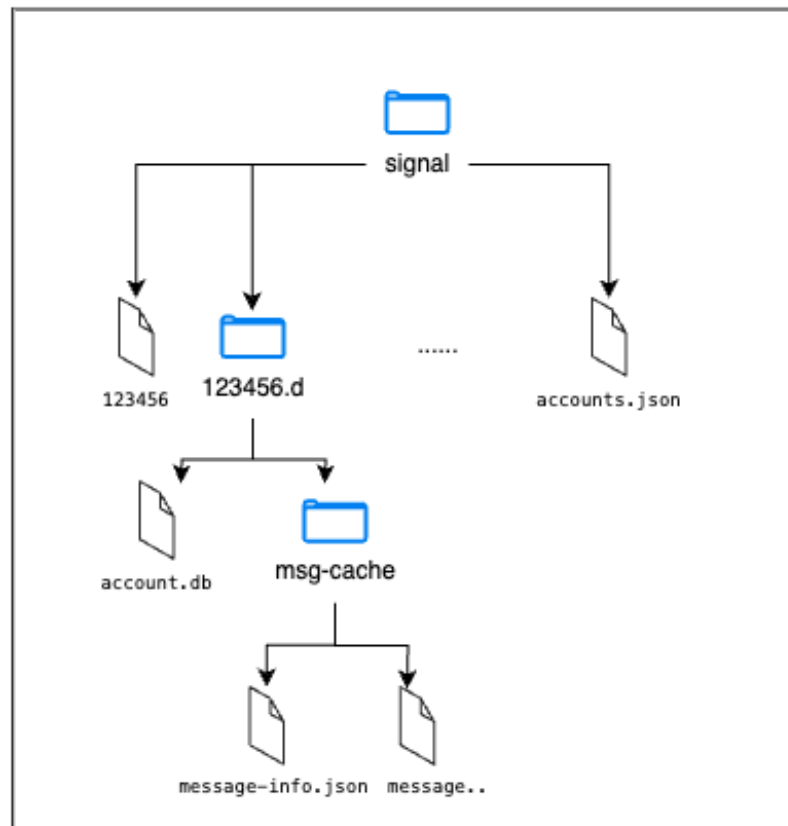


Figure 10: Directories, Files and Databases of Bundler Signal Service

Table 3: Files and Directories of Bundler Signal Service

Name	Type	Description
signal	Directory	It is created in the home directory. It is used to store all files related to Bundler Signal Service.
accounts.json	File	The data is organized based on E164 formatted phone numbers and is stored in directories with unique IDs associated with each phone number. This includes information related to ACI (Account Identifier), PNI (Phone Number Identifier), and other associated IDs. Within each directory, the user's information such as their keys, offsets, and contacts are stored.
123456 (ID for registered user)	File	A JSON file for every user identified by ID allocated by the Bundler Signal Service. This file stores the public counterpart of ACI and PNI Identity Keys, offsets for Prekey bundles, passwords, and session-related information.
123456.d	Directory	A directory to store user-related keystore database files and encrypted incoming messages (mailbox).
account.db	File	Each registered user has this SQLite file that stores all key stores, contacts, and other related information required to interact with the Signal protocol. It mainly acts as a persistent storage mechanism for key management.
msg-cache	Directory	This directory stores all incoming encrypted messages for the specific user with headers and timestamps.
message-info.json	File	This file maintains a list of all the encrypted incoming messages with their sent, received, and viewed timestamps. This file is mainly used by the Signal Adapter to maintain a list of newly received messages since the last sync.
message..	File	This file contains the actual encrypted Signal message which is received for the registered user. Since private keys are missing on Bundler Signal Service, there is no way to decrypt this message on the server side. This message with its respective Prekey offset when transferred to the client can be decrypted and viewed.

CHAPTER 5

Experiments

This section presents the results of testing the Disconnected Signal application in various scenarios, such as user registration, message reception, and user-related data maintenance. The test results demonstrate the feasibility and effectiveness of the Disconnected Signal application in a disconnected setting.

The messaging application is currently operating on Android 12, while the Signal Adapter and CLI are running on a Linux machine equipped with Java 17. The application programming interfaces (APIs) required to communicate with Signal servers leverage features exclusively available in Java 17 and newer releases. The Disconnected Signal messaging application is derived from the Signal-Android repository [31], while the CLI is created from an additional open-source project known as Signal-CLI [32]. Table 2 lists the libraries being utilized in the projects along with their corresponding versions.

5.1 Signal Android Application

For this project, a modified version of the Signal Android application has been developed. Within this modified application, a specific class called "DDDKeys" is responsible for generating all necessary keys for registration. When the user opens the Signal application for the first time and clicks the "continue" button on the main screen, the application fetches contacts from the phone using the Contacts Provider, an Android content provider component. After this step, a DDDKeys object is created and serialized into JSON format. This JSON object is then stored as a file on the user's phone.

Once the file is created, the Signal app sends an Android Intent containing the file contents to the Bundle Client app. The Bundle Client app then stores the file in its directory, where it remains until the transport application comes within range of

the client phone to retrieve the bundles. Figure 11 shows the keys file stored in the Bundle Client’s application directory.

▼ com.ddd.bundleclient	drwxrwx--x	2023-04-21 15:58	4 KB
> BundleTransmission	drwx-----	2023-04-21 16:00	4 KB
> cache	drwxrws--x	2023-04-21 16:00	4 KB
> code_cache	drwxrws--x	2023-04-21 16:00	4 KB
> databases	drwxrwx--x	2023-04-21 16:00	4 KB
> Keys	drwx-----	2023-04-21 16:00	4 KB
▼ send	drwx-----	2023-04-21 16:34	4 KB
▼ org.thoughtcrime.securesms	drwx-----	2023-04-21 16:34	4 KB
1.txt	-rw-----	2023-04-21 16:34	35.9 KB
metadata.json	-rw-----	2023-04-21 16:34	99 B
> Shared	drwx-----	2023-04-21 16:11	4 KB

Figure 11: Keys File Stored at Bundle Client

5.2 Bundler Signal Service

The last Signal related transactions happen on the Bundler Signal Service where the bundle of public keys arrives. Bundler Signal Service is a Gradle project designed to be operated as a CLI. To register a user, two things are needed to be manually entered - Signal CAPTCHA and a phone number. The CAPTCHA is generated from a website hosted by Signal and the phone number is generated using internet-based tools like Google Voice. To test if the registration is happening, one can try to search for this generated number on the actual Signal application. As this number is newly created and not registered, the Signal application will throw an error saying that the phone number is not registered on Signal (Figure 12).

Once the CAPTCHA and phone number are generated, they need to be passed to the register command of Signal CLI along with the bundle location. The bundle is unparsed and the Identity Keys are created based on this bundle information. The bundle location is also stored as Prekeys and Signed Prekeys are only registered after the user has been verified on Signal. After the generation of Identity Keys, the CLI

sends a registration request on behalf of the phone number and requests a pin for verifying the phone number.

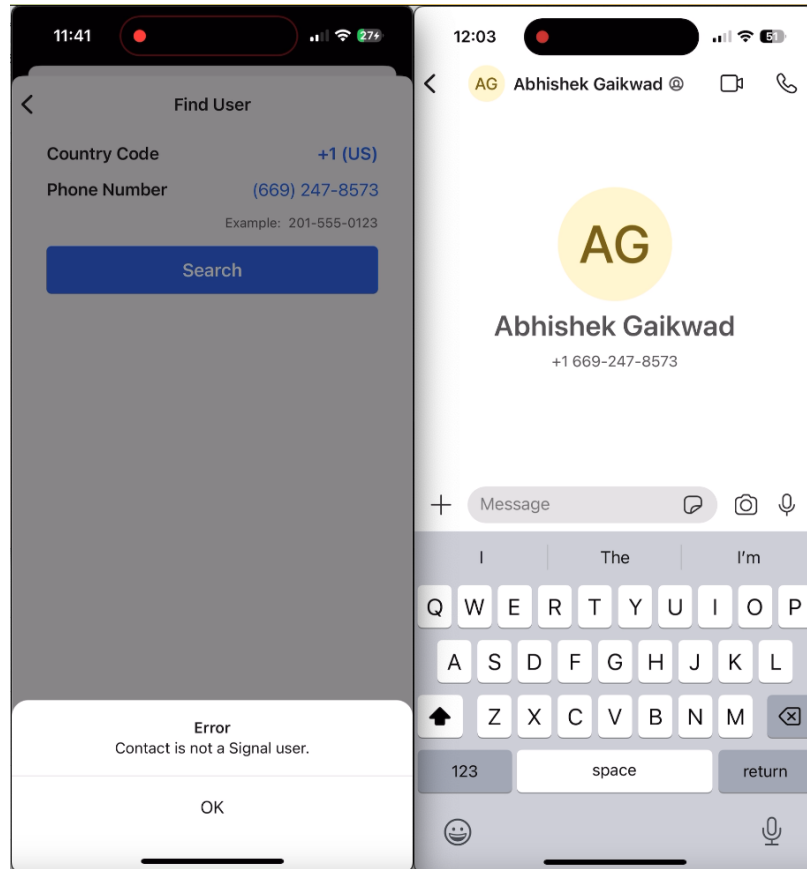


Figure 12: Finding Phone Numbers Pre and Post Registration Attempt

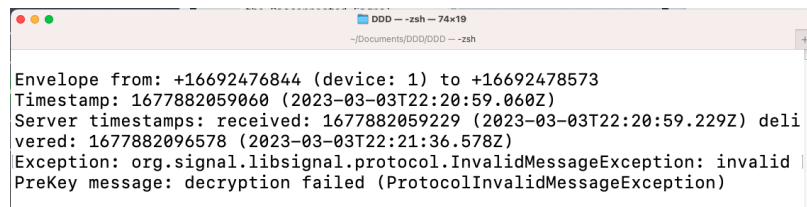
A temporary pin should be received after a successful register request. This pin is then used in the verification command on the CLI. If the verification is successful, the Signal server returns two unique IDs called ACI and PNI. Every registered user on Signal has these two unique IDs. All subsequent communication to Signal Server must use these two IDs in the header for identification purposes. The next step is to register all the Prekeys received in the registration bundle which is achieved by calling the register Prekeys API. When Prekeys are registered, other users can find the phone number that was being registered (12) and initiate a conversation using the Identity

Keys and one of the Prekeys from the Prekey Bundle uploaded during registration.

After registration, if one attempts to find the phone number on Signal Messenger, a message window will appear stating that the account is registered on the Signal network and can be communicated with, as shown in Figure 12.

As the user is now in the Signal network, other registered users will be able to message the user. If other users want to message the DDD user, they will fetch the public part of Identity Key of the user from Signal server, Signed Prekey and one Prekey bundle. Based on these keys, the other user can create a shared secret key using X3DH protocol [26]. The first message which is sent to the DDD user will be encrypted using this shared secret key. Signal doesn't use same keys to encrypt all the messages. All subsequent messages in the conversation will be generated based of this shared secret key using the Double Ratchet Algorithm [27].

An attempt to decrypt the received messages using random keys generates a decryption failed exception (ProtocolInvalidMessageException) as shown in Figure 13. The encrypted messages will only be decrypted by the private counterpart of the keys which exist on the clients phone. Those private keys never leave clients phone thus providing end-to-end encryption for messages over the entire DDD network.

A terminal window titled "DDD -- zsh -- 74x19" with a path of "~/Documents/DDD/DDD -- zsh". The output text is as follows:

```
Envelope from: +16692476844 (device: 1) to +16692478573
Timestamp: 1677882059060 (2023-03-03T22:20:59.060Z)
Server timestamps: received: 1677882059229 (2023-03-03T22:20:59.229Z) deli
vered: 1677882096578 (2023-03-03T22:21:36.578Z)
Exception: org.signal.libsignal.protocol.InvalidMessageException: invalid
PreKey message: decryption failed (ProtocolInvalidMessageException)
```

Figure 13: Attempt to Decrypt Messages on Bundle Signal Service

CHAPTER 6

Future Work

This section discusses the various limitations that arise when using Signal in a disconnected environment. The limitations are examined, and potential solutions for addressing them in the future are explored.

6.1 Limitations

The Signal messaging protocol requires periodic rotation of certain keys, as listed in Table 1. However, the current implementation does not allow for automated key rotation. Operator intervention is required on the Signal Adapter to generate a phone number and CAPTCHA. In the context of a disconnected setting, it is not feasible to conduct voice calls, and our current implementation does not allow for group chats, sharing of media, or voice notes. In the event that a disconnected user loses their device, there is no mechanism for them to log in to the previously assigned phone number.

In the subsequent sections, potential solutions to the aforementioned limitations will be discussed, along with a few novel ideas.

6.2 Enhancements in Bundler Signal Service and Signal-Android

The Bundler Signal Service is a good base for other developers to start upon. Currently, the scope is limited to registration, key management, storage of user-related info, establishing presence, and receiving messages. The flow for sending messages from the Signal-Android app still needs to be worked on. Currently, the message can only have text information, Signal app supports images, videos, voice messages, documents, and stickers which need to be added to Bundler Signal Service.

There is no support for group chat which also can be added. The encryption protocol for group chat works differently than one-to-one messaging. Fetching profile information such as last seen, display picture and status can also be synced on

disconnected Signal application.

Signal also supports voice and video calls, but as we are in a disconnected setting where information gets relayed through different nodes over time this will never be possible.

6.3 Collaboration with Signal Developers

The collaboration between DDD and Signal developers could be very beneficial for disconnected users who rely on DDD's version of Signal Messenger. Currently, the Bundler Signal Service creates a new phone number using Google Voice for every disconnected user to receive the SMS verification needed for registration. However, if Signal developers integrate DDD's architecture into their official app, it could streamline the registration process and improve the overall user experience.

6.4 Potential of DDD for Various Internet-based Applications

The main idea of disconnected data distribution over limited connectivity can be applied to a plethora of other internet-based applications. Content sharing or streaming application like YouTube and Netflix can have their disconnected versions where disconnected users will be able to request a certain video that can be transferred to them over the DDD network. Other services like e-mail or social media can function in a disconnected setting where users will be able to send or receive encrypted emails over the network and post a picture or get the latest updates from people who the disconnected user is following on social media websites. This is a completely new domain, which has not been worked on. If successful, companies that want to cater to disconnected users and improve their user base can design APIs specifically to handle requests from disconnected users.

CHAPTER 7

Conclusion

The DDD architecture is designed and implemented on the foundation of privacy. Data is transported in the DDD in form of encrypted bundles. Host phones responsible for transporting user data from disconnected areas to connected areas have no idea about the contents of the data. The private keys that are involved in Signal protocol [26, 27, 28] never leave the user’s phone, hence making decryption of the messages in the bundle impossible achieving complete privacy.

Several previous projects in related research areas have struggled due to their dependency on costly infrastructure, human intervention, and ongoing maintenance. In contrast, DDD is a software-only solution using normal Android phones and servers. Users don’t need any additional equipment beyond their smartphone to participate in the disconnected network and use supported applications.

DDD architecture constitutes many different components. The client-side components, such as the disconnected Signal Messenger and Bundler Client Android application, are responsible for generating encryption keys and bundling data before sending it to the host phone. The Bundle Transport application operating on the host phone is responsible for transporting the data to the Bundler Server, which then decrypts the bundles and forwards them to the application-specific adapter. The Bundler Signal Service is a crucial component in this architecture, it maintains the disconnected users’ information and provides an entry point to the Signal messaging services. Overall, this architecture prioritizes modularity, scalability, and maintainability, which can be essential for building complex software systems that can evolve over time where a new application can be onboarded with ease.

The aim of this research is to address the digital divide in areas with limited connectivity by enabling users to join a messaging network while maintaining their

privacy. This is accomplished through the use of Signal Messenger, which runs the Signal Protocol in a disconnected setting, ensuring that users' private keys, necessary for message signing and decryption, remain on their phones. By adopting this approach, the digital divide can be narrowed, and access to critical communication services can be extended to those who are currently underserved.

LIST OF REFERENCES

- [1] S. R. Department, “Worldwide digital population july 2022,” <https://www.statista.com/statistics/617136/digital-population-worldwide/>, Sept 2022, (Accessed on 12/01/2022).
- [2] G. Butler, “Extreme weather causes outages in the us and australia,” <https://www.datacenterdynamics.com/en/news/extreme-weather-causes-outages-in-the-us-and-australia/>, March 2022, (Accessed on 12/01/2022).
- [3] A. F. Per Engzell and M. D. Verhagen, “Learning loss due to school closures during the covid-19 pandemic,” <https://www.pnas.org/doi/10.1073/pnas.2022376118>, April 2021, (Accessed on 12/01/2022).
- [4] A. Pentland, R. Fletcher, and A. Hasson, “Daknet: rethinking connectivity in developing nations,” *Computer (Long Beach, Calif.)*, vol. 37, no. 1, pp. 78--83, 2004.
- [5] eZ. Systems, “Wizzy digital courier,” <https://web.archive.org/web/20120204071251/http://wizzy.org.za/>, (Accessed on 12/01/2022).
- [6] R. C. Shah, S. Roy, S. Jain, and W. Brunette, “Data mules: modeling and analysis of a three-tier architecture for sparse sensor networks,” *Ad hoc networks*, vol. 1, no. 2, pp. 215--233, 2003.
- [7] K. Fall, “A delay-tolerant network architecture for challenged internets.” ACM, 2003, pp. 27--34.
- [8] A. Chaintreau, A. Mtibaa, L. Massoulie, and C. Diot, “The diameter of opportunistic mobile networks.” ACM, 2007, pp. 1--12.
- [9] D. CAMPS-MUR, A. GARCIA-SAAVEDRA, and P. SERRANO, “Device-to-device communications with wifi direct: Overview and experimentation,” *IEEE wireless communications*, vol. 20, no. 3, pp. 96--104, 2013.
- [10] “Signal messenger,” <https://signal.org/en/>, (Accessed on 12/01/2022).
- [11] M. Marlinspike, “Whatsapp’s signal protocol integration is now complete,” <https://signal.org/blog/whatsapp-complete/>, (Accessed on 12/01/2022).
- [12] J. C. Haartsen and S. Mattisson, “Bluetooth-a new low-power radio interface providing short-range connectivity,” *Proceedings of the IEEE*, vol. 88, no. 10, pp. 1651--1661, 2000.

- [13] V. Coskun, B. Ozdenizci, and K. Ok, “The survey on near field communication,” *Sensors (Basel, Switzerland); Sensors (Basel)*, vol. 15, no. 6, pp. 13 348--13 405, 2015.
- [14] M. Cao and P. Yin, “Research of usb otg technology in image high-speed data transfer,” pp. 261--264, 2014.
- [15] K. Yu, S. Eum, T. Kurita, Q. Hua, T. Sato, H. Nakazato, T. Asami, and V. P. Kafle, “Information-centric networking: Research and standardization status,” *IEEE access*, vol. 7, pp. 126 164--126 176, 2019.
- [16] S. H. Ahmed, S. H. Bouk, and D. Kim, *Content-Centric Networks An Overview, Applications and Research Challenges*, 1st ed. Singapore: Springer Singapore, 2016.
- [17] A. Cilfone, L. Davoli, L. Belli, and G. Ferrari, “Wireless mesh networking: An iot-oriented perspective survey on relevant technologies,” *Future internet*, vol. 11, no. 4, p. 99, 2019.
- [18] A. N. A and L. Rajabion, “Data replication techniques in the mobile ad hoc networks: A systematic and comprehensive review,” *International journal of pervasive computing and communications*, vol. 15, no. 3, pp. 174--198, 2019.
- [19] “Sneaketnet,” <https://en.wikipedia.org/wiki/Sneaketnet>, (Accessed on 12/01/2022).
- [20] C. Farivar, “Google’s next-gen of sneaketnet,” <https://www.wired.com/2007/03/googles-next-gen-of-sneaketnet/>, (Accessed on 12/01/2022).
- [21] “Storage transfer service - google cloud,” <https://cloud.google.com/storage-transfer-service>, (Accessed on 12/01/2022).
- [22] J. Clover, “Find my app: Everything to know,” <https://www.macrumors.com/guide/find-my/>, Feb 2022, (Accessed on 04/10/2023).
- [23] W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644--654, 1976.
- [24] M. Amara and A. Siad, “Elliptic curve cryptography and its applications.” IEEE, 2011, pp. 247--250.
- [25] D. J. Bernstein, “Curve25519: New diffie-hellman speed records,” in *Public Key Cryptography - PKC 2006*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207--228.
- [26] M. Marlinspike and T. Perrin, “The x3dh key agreement protocol,” <https://signal.org/docs/specifications/x3dh/>.

- [27] T. Perrin and M. Marlinspike, “The double ratchet algorithm,” <https://signal.org/docs/specifications/doublerratchet/>.
- [28] T. Perrin and M. Marlinspike, “The sesame algorithm: Session management for asynchronous message encryption,” <https://signal.org/docs/specifications/sesame/>.
- [29] A. Satapathy and J. Livingston, “A comprehensive survey on ssl/ tls and their vulnerabilities,” *International Journal of Computer Applications*, vol. 153, pp. 31--38, 11 2016.
- [30] A. Albarqi, E. Alzaid, F. A. Ghamdi, S. Asiri, and J. Kar, “Public key infrastructure: A survey,” *Journal of information security*, vol. 6, no. 1, pp. 31--37, 2015.
- [31] signalapp, “Signal android (github),” <https://github.com/signalapp/Signal-Android>, (Accessed on 04/23/2023).
- [32] AsamK, “signal-cli (github),” <https://github.com/AsamK/signal-cli>, (Accessed on 04/23/2023).