ICSI499 Capstone Project Report

Building a Secure Instant Messaging Application, Cipher

Project Team 2

Dylan Moran (001421899)

Sydney Pennington (001434588)

Jordan Rivera (001395016)

Joey Saline (001424486)

Mason Wolfe (001313851)

College of Engineering and Applied Sciences

University at Albany, SUNY

Project Sponsor

Pradeep K. Atrey, Ph.d.

College of Engineering and Applied Sciences
University at Albany, SUNY

05-10-2023

Acknowledgements

We would like to acknowledge and thank our sponsor Professor Pradeep K. Atrey, Ph.D. for allowing us to work on and develop this project. In addition to Shashank Arora, you both have always been ready to assist us. Your ability to answer any and all questions we had, give us timely responses and feedback, and help us with any issues we had throughout the semester allowed this project to reach its full potential. Your trust and faith in us as a group did not go unnoticed and grateful for the opportunity.

Abstract

The number one most glaring issue in the field of technology and science is the breach of privacy. Our society has become so accustomed to the Internet and its endless informational voids. Yet, not many people even know how media is delivered to their browser, or how that email reaches the right person. The field of computer science and computer networking is very niche, and individuals looking from the outside in often have no clue what is going on. Ignorance is not always bliss, as not knowing how these technologies work can leave you extremely vulnerable to different types of attacks on your privacy, finances, and lifestyle.

It is not an easy task to educate the world on such a complex topic, that involves an infinite number of working pieces like networks, servers, protocols, and applications all coming together to bring about the Internet and the World Wide Web. A more reasonable approach to protection is to offer a product that inherently solves the issues mentioned above in an exciting and approachable manner. The production of an instant messaging application is not unique in the slightest. Every day people utilize this technology through various platforms, so it is obvious the desire is there. Instant messaging is deeply intertwined in everyone's lives, so we could use this device to offer the protection the world so desperately needs.

The instant messaging application was intended to safeguard any users of the platform and make it just as intuitive and enjoyable to use. The safety revolves around minimizing saved user data, preventing man-in-the-middle attacks, and facilitating convenient stress communication that stays between you and the person you want to talk to.

We set out to solve the problem by extensively researching instant messaging protocols, to see which would give us the most control and resources for the application. We weighed the pros and cons of various encryption methods again keeping the application's needs in mind and studied where the implementation could benefit the most.

In the end, we know this app would greatly benefit society. If it can be competitive with other messaging applications in terms of user interface and overall experience then it will fall into the hands of millions and keep them safe without any effort on their part.

Contents

1 Introduction

As technologies grow at a monumental pace and communication increasingly takes place more online than in person, what steps can we take to ensure personal privacy? It is apparent that the information you share online is not safe from prying eyes. In the wake of massive data breaches and leaks, between the government, big corporations, and hackers, we as individuals must evolve and learn how to protect our privacy. In this project, we aim to break down the stigma that in order to be user-friendly it must be intrusive and provide an aid to people who seek to protect their privacy.

Cipher is a non-persistent, end-to-end encrypted messaging app. This app is important in protecting one's basic right to privacy, which remains at the center of conversation as technology advances; whether that be amongst smaller groups or larger amenities. Now that we are living in a Digital Age, where our personal conversations can be carried across the world, users continue to be vulnerable to attacks against their privacy and the loss of the comfort that comes with that. An app that is both non-persistent and encrypted stops the production of any chat history and backlogs, while simultaneously protecting the messages that are sent from end to end, user to user.

2 Problem Analysis

Throughout the semester we've worked to build a non-persistent, end-to-end encrypted instant messaging app, which prevents the production of any chat history and backlogs, while simultaneously protecting the messages that are en route from user to user.

2.1 Significance

A common problem amongst instant messaging applications is maintaining the privacy of their users and overall security. Messages can contain important information users do not want publicly known. Messages can contain location information, user information, financial information, or sensitive information. Chat applications use encryption and non-persistence as security solutions to prevent access to message content. Most chat applications contain either encryption or non-persistence but not both. The problem we want to solve is to implement both encryption and non-persistence into our application. By adding both encryption and non-persistence, we can address the problem of securing messages for our Users.

2.2 Challenges

Crafting a system such as Cipher means implementing end-to-end encryption, non-persistence, and an entire instant messaging app itself. Considering this, building a messaging app comes with its own set of questions. Who can use this app? Is it accessible on IOS devices, Androids, or over the Internet? How can we ensure people want to use the app over competitors? What are the best choices for user interface? What libraries and technologies are relevant and can assist us? Then, there is figuring out which encryption method we want to implement. There is symmetric encryption, asymmetric encryption, different encryption algorithms, and much more to consider. Each method better suits different types of data and use cases. Messages themselves can be sent using various different protocols. The success of Cipher will ultimately be decided by whether we make the appropriate design choices. If we have to start over, we lose time to work on the application and may fall short of achieving a completely secure, functional system.

2.3 Existing Solutions

To solve the problem of security and privacy for Instant Messaging Applications, Security protocols like encryption and non-persistence are implemented to increase the security of messages. Existing solutions typically use one of the security protocols instead of using both. Some existing solutions that implement End to End Encryption are Facebook Messenger, What's App, Telegram, and Signal. Other existing solutions that implement non-persistence are Snapchat and Signal.

The existing solutions are not sufficient because they only provide a base level of security instead of incorporating multiple layers for message security. A glaring limitation is the implementation of both non-persistence and encryption. Listed are existing solutions with an explanation of their strengths and weaknesses.

2.3.1 Facebook Messenger

Facebook Messenger collects and stores almost all user information. Facebook is known to sell user information and has worked with intelligence agencies in the past. User messages are not encrypted by default, however, they have recently added a feature that allows a user to opt-in for end-to-end encryption.

2.3.2 Snapchat

While Snapchat is perceived to be non-persistent because you send a picture or chat and it deletes after it is seen, they actually store almost all of your data. The messages and pictures may be non-persistent on your device, however, your message history is still saved in Snapchat's

cloud. On top of that, the pictures are encrypted end to end, but the messages for some reason are not. [8]

2.3.3 WhatsApp

WhatsApp does use end-to-end encryption but not for your metadata which means it is easily accessible by staff and possible hackers, who you are talking to, when, and from where. WhatsApp has also worked with intelligence agencies in the past. However, they do not store your chat history on their servers for long. Only if the message cannot be delivered to you right away. Once the message is delivered to your device, it is then deleted. [9]

2.3.4 Telegram

Telegram is not end-to-end encrypted by default. It is only encrypted from client to server. You can choose to make a secret chat with one other person which will be end-to-end encrypted. Telegram does not support end-to-end encryption for group chats. They also collect your IP address, device information, and username changes. [12]

2.3.5 Signal

Signal has reviews calling it one of the most secure apps available. Signal has upgraded its encryption protocol utilizing the Double Ratchet Algorithm. This algorithm uses additional session keys with public and private keys created once the user creates a messaging profile. These session keys are created for messages sent by the user, but disappear once the message has been sent and received by the intended user.[10] As inventive as this is, man-in-the-middle attackers that are just as inventive are able to get the session key for its specific session [13]

Although they would be unable to decrypt all messages with one session key, users are still vulnerable to exposure of private information. In popular, but less secure messaging apps such as WhatsApp, personal messaging is noted to be secured with a lock and will only be unlocked with a special key once the message is read [8] Although this is done automatically with no other special features, metadata including who you're talking to, when, and from where is all recorded and not encrypted. This information is in turn used by Meta, the parent company of WhatsApp, and used to inform other businesses and advertisers and track user trends.

Outside of the encryption itself, messaging apps use other steps to ensure security when messaging. Solutions include verification codes sent via SMS to your phone or a screen lock. Signal uses a PIN code with four digits chosen by the user or in other cases there are Face ID and Touch ID that can be used. Signal also requires users to enter a Signal PIN every time they want to register their account on another device.[10][13]

2.4 Proposed Solution (what is your core idea/solution, key characteristics, and how are they different?)

Our core idea and solution is to focus on the encryption and non-persistence aspects of our app before tackling other security features that apps like Signal have. Cipher utilizes the two main methods of encryption, asymmetric and symmetric, to create a hybrid approach that is both efficient and secure. This includes encrypting a user's message with AES encryption, a symmetric key algorithm, and then encrypting the symmetric AES key with the recipient's RSA public key. RSA encryption being the asymmetric key algorithm, using public and private keys. Once the message is received, the recipient decrypts the AES key with their RSA private key, which is used to decrypt the first user's message. This is used for dialogue exclusive to two users messaging but sets the foundation for encryption applied to group messaging. Our Hybrid Encryption ensures if a message does persist, the message contents are encrypted. The non persistence will be based for the Sender immediately disappears, however this differs with the recipient. With the Recipient, it is based on if they have read the message. If the message is read, then it will disappear. If the message is not read, the message disappears after 24 hours. This differs from previous solutions by tackling a more tightly secure approach for message sending. The only other solution to this problem has been worked on by Signal.

2.5 Key Aspects

- Non-persistence in messages one-to-one and group messaging
  - Messages are deleted from the frontend and backend of the Application
- End-to- end encryption in messages one-to-one and within groups
  - Messages are always encrypted from end to end through hybrid encryption
- Ability to log in, and sign up as a Cipher user
  - The ability to create an account and log into the account
- Creating and removing user accounts
  - The ability to have control of creating and deleting User accounts with ease
- Message others via iOS and Android
  - Cross Platform capabilities
- Users are allowed to manage a friend's list i.e block and add users
  - The ability to control a friends' list with ease
- Edit user profiles
  - The ability to control a friends' list with ease

- Create and manage members within group chats
  - The ability to host and moderate chat rooms consisting of multiple other Users
- Account activity status
  - The ability to see other Users' activity statuses on demand

	WhatsApp	Signal	iMessage	Threema	Facebook Messenger	Cipher (Our Project)
Collects User Data	No	No	Yes	No	Yes	No
Metadata is Encrypted	No	Yes	No	Yes	No	Yes
Open source	No	Yes	No	Yes	No	Yes
Stores timestamps and IP addresses	Yes	No	Yes	No	Yes	No
Encryption key kept with	The App	The App	The App	The App	No Encryption	The User
Non-persistency	No	Yes	No	No	No	Yes

Figure 1: Existing Solutions Table

This table shows our app and its features in comparison to leading encrypted messaging apps. The remainder of this report describes in detail the process of design and

implementation of Cipher, start to finish. This includes our original proposed solutions in comparison to our product at the end of the semester. Additionally, thorough testing for key aspects of the messaging app will be reviewed within both the backend and frontend. With discussion of the legal and ethical concerns related to building a group project in this manner and its overall impact following this course including lessons learned and where we can see relevance in future work.

2.6 Project Overview

2.6.1 Overview

The proposed system is going to be able to run on IOS, Android, and the World Wide Web. This is the client side mobile platform that will rely on an Ejabberd server. The Ejabberd server can be administered from a dashboard in a Desktop environment. Cipher's Backend is structured through an Ejabberd Server where it will host all of the Chat Application's Data including Users and Administrators. For this Cipher, we have Users, Chat Room Administrators, and Chat Room Members that have different use cases. Cipher's front end is run through Android XML sheets and Swift GUI design. The Web Application uses React for great portability and function, and bootstrap for a seamless user interface design. The core required component of our system is establishing an XMPP connection where all of the functional requirements are implemented. This XMPP connection is either rejected or accepted depending on the User. If a user exists and logins in successfully, they will be able to chat with another user. User information (username, password, and Rosters) is stored in a Mnesia database. Cipher also uses another Database to hold the Public keys of the Users and that database communicates with the Ejabberd Server.

Most of the functional requirements implementation was a mix of Ejabberd's API and an XMPP Library known as Smack. This provided ease for us when handling the XMPP connection for Application Use. Account control was implemented through these methods as well as chat room implementation.

Our Core functional Requirements are implemented using Java and Swift's encryption library in tangent with message string used in the Smack Library. Before they are sent, all message strings are encrypted using RSA and AES. Non persistence in the backend is handled through a Script clearing the Mnesia database after 24 hours (if the message is not read). Non persistence in the front end is handled through a stack or recycler view, where each decrypted message is popped off the screen.

Our front end is platform specific, so Android implementation requires XML sheets to produce a functioning User interface, IOS implementation requires swift sheets to do the same. The Web Application can serve requests to anybody with Internet access on any type of device, including desktops and mobile devices. Without these requirements, a functional user interface is not achievable.

2.7 Project Requirements

2.7.1 User Class and Functional Requirements

General User

This user class describes any user that communicates using Cipher but isn't administering

or participating in any chat rooms, essentially encompassing a user's default abilities within the app. They exist only upon creation of an account associated with a unique user id called a JID, while the account is logged in, and can enact functional requirements only to this account.

Functional Requirements for this user class include:

- Join a chat room
  - All user have the ability to join a chat room
- Encrypt all messages sent from end-to-end
  - Messages are encrypted using a Hybrid Protocol of AES and RSA
- Ensure all messages are non-persistent
  - All messages despite being read or not, are wiped from the front end and back end
- Register an account
  - Any user should be able to create and account with a password, username, and email
- Terminate account
  - Any Registered user should be able to delete their account
- Log in to an account
  - Any registered user should be able to log in to their account if the inputted credentials are correct
- Log out from an account
  - Any registered user should be able to log in to their account if the inputted credentials are correct
- Send a message to another user
  - Any user should be able to send messages to whoever they are allowed to chat with
- Receive messages from other users
  - Any user should be able to receive incoming messages from other users
- Edit the profile of an account
  - Users are allowed to have control over their account preferences

- Set the status of the account
  - Logged in users are able to change their status
- Manage a friends' list
  - Users have control on who they want on their friends list
- View Friend's statuses
  - Users are able to see their Friends' displayed statuses

2.7.2 Chat Room Member

This user class inherits all functional requirements from a General User, with additional chat room specific functionality. A Chat Room Member exists only inside of an active chat room, from the duration of when the user is first added to the room until the user leaves, is removed by a Chat Room Administrator, or the chat room is terminated by a Chat Room Administrator.

Functional requirements for this user class include:

- Send a message to all other users within a chat room
  - All Chat Room Participants are able to send messages within the chat room
- Receive messages from other users within a chat room
  - All Chat Room Participants are able to receive messages within the chat room
- Leave a chat room
  - All Chat participants are allowed to leave the room they currently in

2.7.3 Chat Room Administer

A Chat Room Administrator is a General User that is the creator of a chat room, and inherits the associated functionality only within the created chat room. This user class inherits all functionality of a General User, as well as a Chat Room Member, with additional administrative abilities that allows this user class to maintain a chat room.

Functional requirements for this user class include:

- Add a user to the chatroom
  - A Chat Room Administrator can send invitations to the room they are administrating to
- Remove a user from the chat room

  - A Chat Room Administrator can remove any chat room members from the room they are administrating

- Terminate a chat room
  - A Chat Room Administrator can delete the existence of the room they are administrating

2.8 Non-Functional Requirements

2.8.1 Performance

Any user should be able to create a new account within 30 seconds if their username, password and email is entered correctly.Any message they send should reach the server within a second whether that be via an API request or XML stanza. Messages should reach recipients within 10 seconds and take no longer than a minute to reach them and be displayed correctly. Additionally, encryption is done utilizing the lower expense of AES, with RSA level encryption and is both Android and iOS compatible, while the Web app never got to implement encryption.

2.8.2 Reliability

In order to maintain reliable and stable use of Cipher, an active WiFi connection ensures messages won't be lost in travel. Web requests should be executed within 5 seconds and a message receival rate greater than 99 percent is expected. All messages are non-persistent on both the client and server side.

2.8.3 Security

All messages sent should be encrypted with the recipient's public key and then decrypted via the recipient's private key once it reaches the recipient's device. Access is limited to the back end of the messaging app. This includes most ports being closed, excluding the one receiving XML stanza and handling HTTP requests. All requests will be handled via an API and/or a Web Server. Furthermore, users must authenticate themselves with their username and password to enable requests related to their account. Users are only able to edit and control information in their account and they have a limited amount of requests available to them. Once a user retrieves a message, it's erased via a request sent to the server to erase all messages. Additionally, messages are erased once the application is closed. Finally, an encrypted database protects all user information.

2.8.4 Availability

This app is accessible to anyone who has the complete source code and the specified platform. Android availability is limited to Java source code and IOS availability is limited to IOS source code. The Web app is always accessible through the World Wide Web, however it lacks many functional requirements and is not up to standard for our project.

2.8.5 Usability

The UI for Android, iOS, and Web is clear with navigation bars indicating where to access friends lists, group chats, invites, user profiles and personal messages.

2.8.6 Scalability

The database and server will be easily upgradable with Processing Power, RAM, and SSD in order to support more on coming users. The server should be able to welcome up to 1000 new users per month Our application should be able to withstand introduction to IOS App Store and Android Google Play. Application should be able to handle traffic of at least 1000 users at once on the Web and handle larger chat rooms and have faster messaging speeds.

2.8.7 Operating Requirements

To properly use the application, a user needs either an Android or iOS mobile device to install the app, internet connection , connect to the application's server and use the device's touch screen. A touch screen is needed to utilize the buttons and keyboard on the app to access the full scope of the app.

2.9 Design and Implementation Constraints

Platform Standard: Cipher for Users are restricted to Mobile devices. The two popular Operating Systems Cipher can function on Android OS and IOS. Cipher is known to function on the Current Android and IOS Patches and may have limited functionality on past patches.

Mobile Hardware Standards: We implemented Cipher on limited Mobile devices and are not aware of the Mobile devices that can not run cipher. We are not sure how Cipher will impact devices with lower specifications than the Test Devices we use. These Specifications include memory, CPU, and device privileges. Most Androids have the same method of changing privileges but IOS uses different methods. This can impact the functionality of Cipher on Specific devices.

The app is restricted to iOS and Android mobile devices, as the Web app does not reach target for necessary functional requirements. Additionally, user information such as the user-

name and password is the only information accessed publicly. The app is simple and accessible at a base level, but leaves out a majority of those with visual imparities that may need more time to read them. If someone needed to have messages read out loud to them they would need more than 4 seconds. Security is still ensured users don't have access to any other user information and the application doesn't keep chat history or keep track of a user's location or personal information.

2.10 Solution Architecture

The application architecture uses a client-server architecture with two separate servers; one for storage of RSA public keys, and one for XMPP support. The XMPP server uses Ejabberd as the host. Ejabberd supports message transfer and querying via the use of XML streams. Ejabberd contains its own internal database that keeps track of users, friends, presences, messages, group chats and more. The other server is accessed and updated via an API that connects to a MySQL database. This server is only used for posting the users RSA public keys. These public keys can be updated, deleted, and requested based on the user's credentials. All RSA key pairs will be generated locally on the client's device. The public key is sent to the server and the private key is kept locally on the user's device. We use a hybrid form of encryption that utilizes AES and RSA encryption methods. A user can only receive messages from those users who are their friends. In order to complete a friend request, a user must know the recipient's Jabber ID (JID) which acts as their username. The request process is as follows:

1. The sender requests and retrieves the recipient's RSA public key.
2. The sender generates an AES key locally.
3. The sender encrypts the AES key with the recipient's RSA public key to secure the key in transit.
4. The sender attaches the encrypted AES key to the friend request and sends.
5. The sender stores the AES key locally in a file, with the status of the friend request, and to whom it was sent.
6. The recipient receives the friend request and decrypts the AES key with their RSA private key.
7. If the recipient accepts, the AES key is stored, along with their new friend's JID.
8. The sender is updated about the status of the friend request.
9. Both the sender and the recipient share the same AES key, which can then be used to encrypt messages between the two.

A similar methodology is used for group chats. The creator of the group generates an AES key and encrypts it with each invitee's RSA public key. The invite is then sent. Once received by each invitee, if they choose to accept, the AES key will be decrypted with their private key and used to encrypt messages within the group chat.

The non-persistence functionality of Cipher is straight-forward. For one-to-one messaging, any message that has been sent, and contained within the message archive, will be deleted after 24 hours. This is achieved via a script within the server host. However, Ejabberd stores messages that have not been seen, in a separate archive and those will persist on the database until seen by the user and then deleted. For group messaging, all group messages are deleted after 24 hours regardless of who has seen the messages.

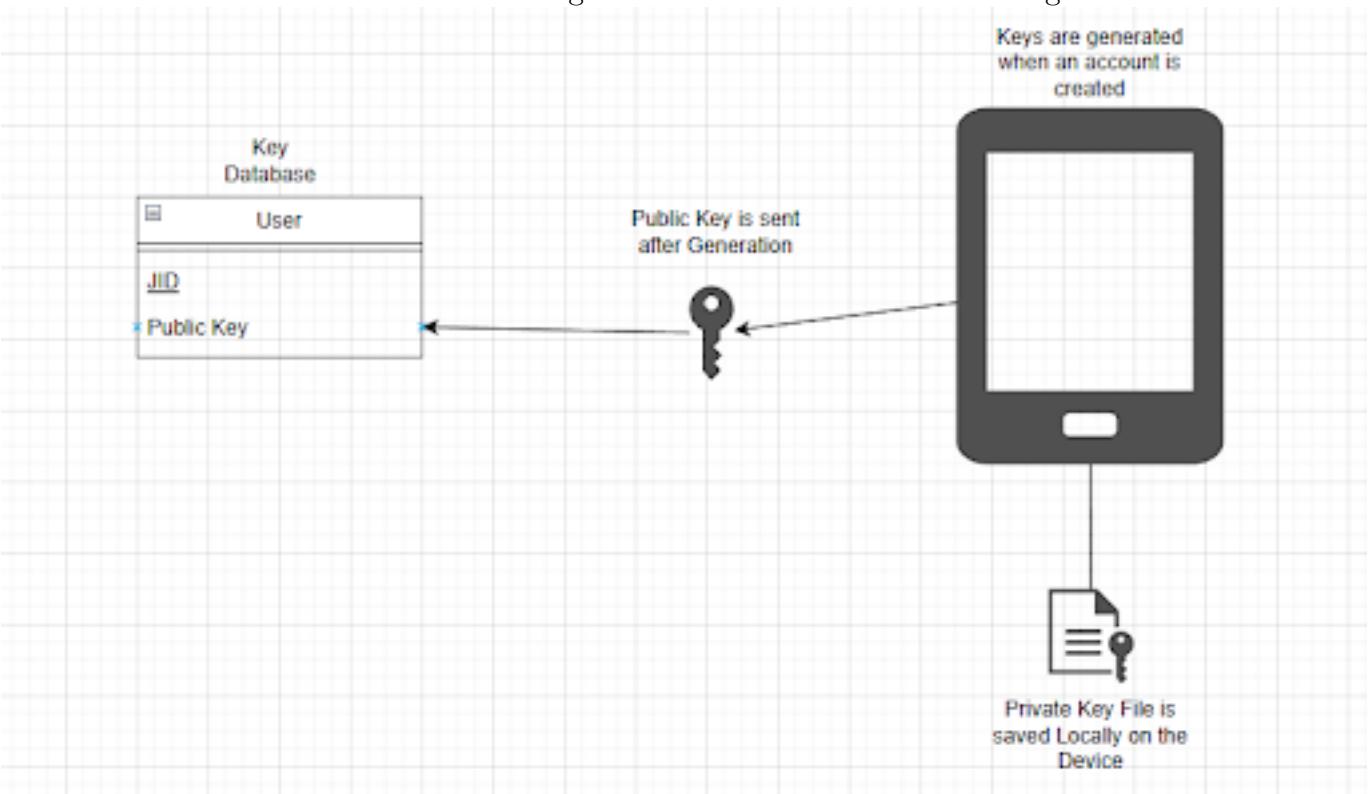


Figure 2: Asymmetric Keys are generated when a User Account is created on the device. The Public Key is shipped via HTTP request to the Key Database where the Username and the associated Public Key is stored. While the Private Key is converted into a Key file and stored locally in the device's Storage

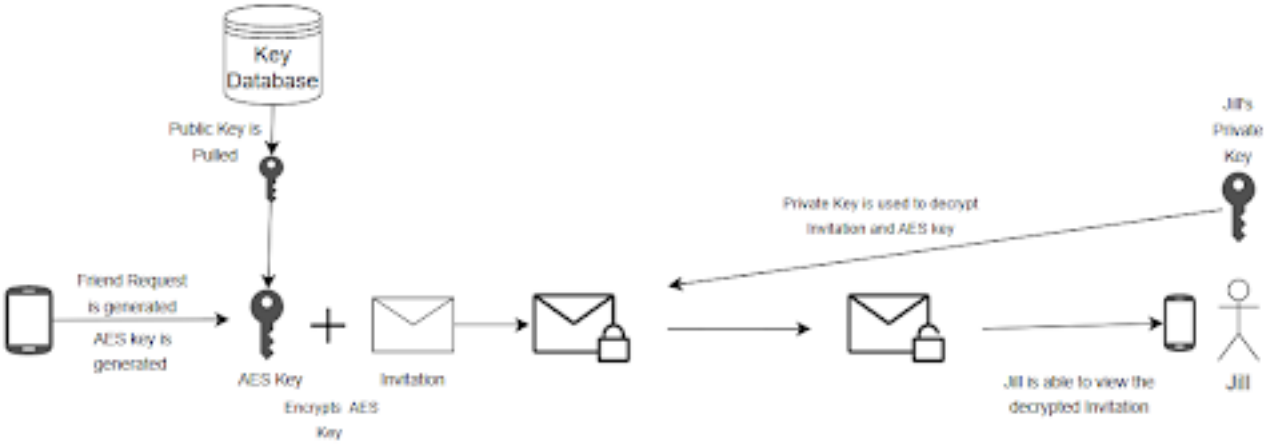


Figure 3: AES Keys are generated when a User sends a friend request. The AES key is encrypted by the Sender's Private Key and packaged with the request. If the Receiver accepts , the Key is decrypted and stored into a Friend list in the local device.

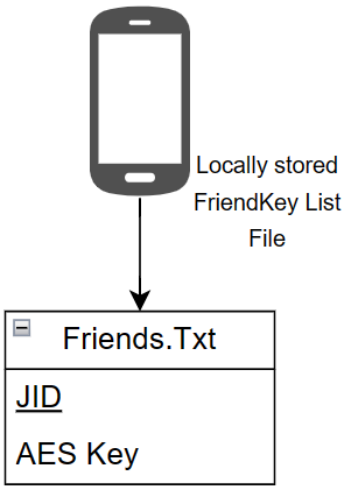


Figure 4: The AES keys are stored locally in a Text file. Each AES key is stored with their associated JID Friend

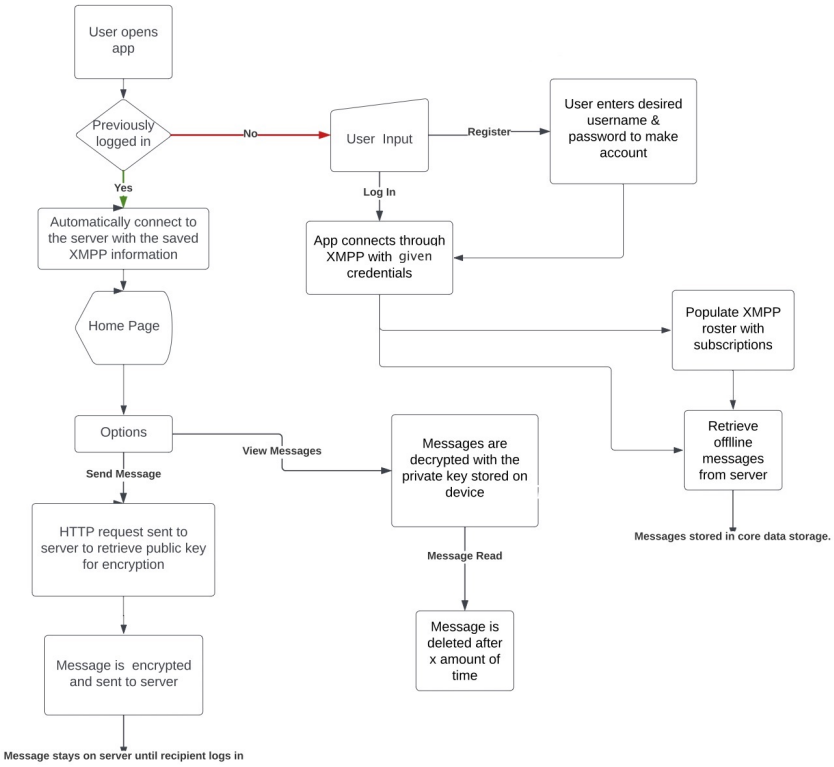


Figure 5: System Data Flow: This flow chart shows interaction user choices and how it effects interaction with the back end server and data storage.

2.11 Technical Design

The proposed system is going to be able to run on IOS, Android, and the World Wide Web. This is the client side mobile platform that will rely on an Ejabberd server. The Ejabberd server can be administered from a dashboard in a Desktop environment. Cipher's Backend is structured through an Ejabberd Server where it will host all of the Chat Application's Data including Users and Administrators. For this application, we have Users, Chat Room Administrators, and Chat Room Members that have different use cases. Cipher's Front end design is created through XML sheets and Swift GUI design. The Web Application uses React for portability and functionality, and bootstrap for a seamless user interface design. The core required component of our system is establishing an XMPP connection where all of the functional requirements are implemented. This XMPP connection is either rejected or accepted depending on the User. If a User exists and logins in successfully, they will be able to chat with another user. User information (username, password, and Rosters) is stored in a mnesia database. Cipher also uses another Database to hold the Public keys of the Users and that database communicates with the Ejabberd Server.

Most of the functional requirements implementation was a mix of Ejabberd's API and an XMPP Library known as Smack. This provided ease for us when handling the XMPP connection for Application Use. Account control was implemented through these methods as well as chat room implementation.

Our Core functional Requirements are implemented using Java and Swift's encryption library in tangent with message string used in the Smack Library. Before they are sent, all message strings are encrypted using RSA and AES. Non persistence in the backend is handled through a Script clearing the mnesia database after 24 hours (if message is not read). Non persistence in the frontend is handled through a stack or recycler view where each decrypted message is popped off the screen.

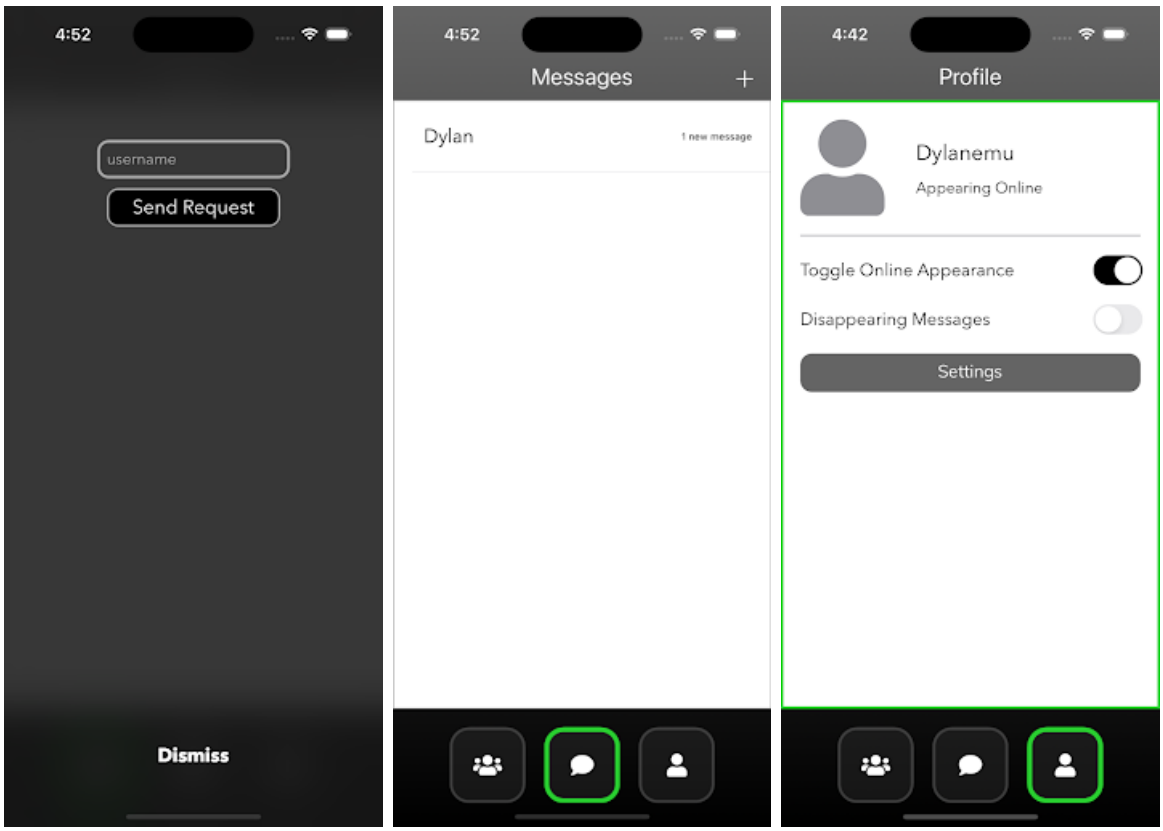
Our frontend is platform specific, so Android implementation requires XML sheets to produce a functioning User interface, IOS implementation requires swift sheets to do the same. The Web Application can serve requests to anybody with Internet access on any type of device, including desktops and mobile devices. Without these requirements, a functional user interface is not achievable.

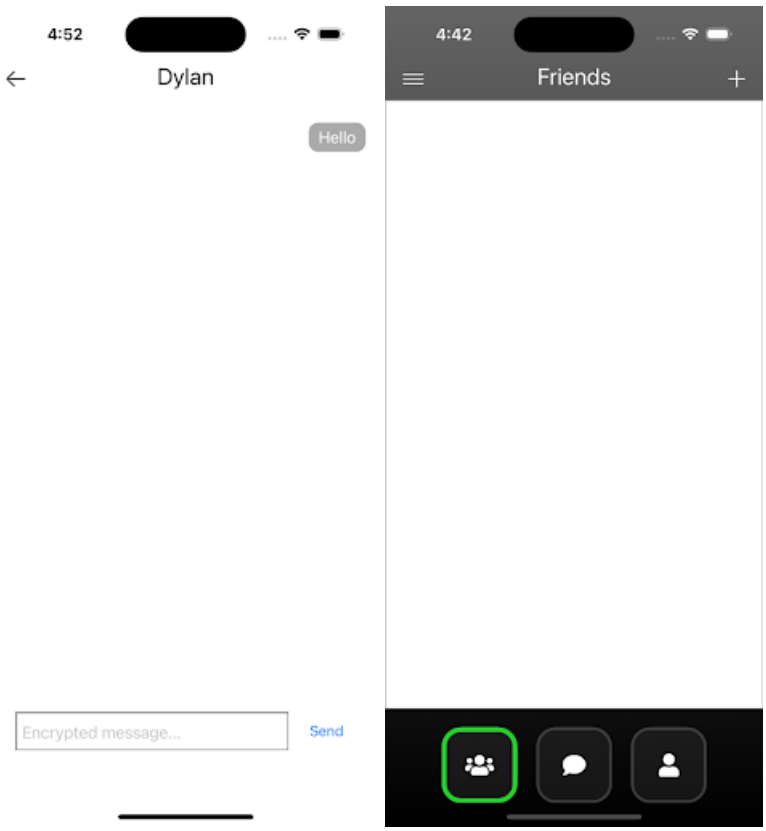
2.12 System Implementation

For the Mobile Application's backend, Java and Swift were the primary programming languages to implement the encryption and the functional requirements. The Web application relied on the React javascript framework to encode functional requirements. Java was used for the Android Application and Swift was used for the IOS application. Android Studio was the IDE we used to test and emulate our Application and Xcode was the IDE for the emulation for IOS. The Ejabberd Server connected to both IOS and Android clients with ease despite being two different platforms. The Ejabberd Server allowed the backend to operate and be implemented with the Frontend. For the Ejabberd API, Json objects were used to store Public keys with their associated User. The implementation was tested using Postman. The Encryption was programmed in Java for Android and Swift for IOS. Non Persistence was programmed in a Bash Script and continuously runs in the Ejabberd Daemon.

2.12.1 iOS GUI

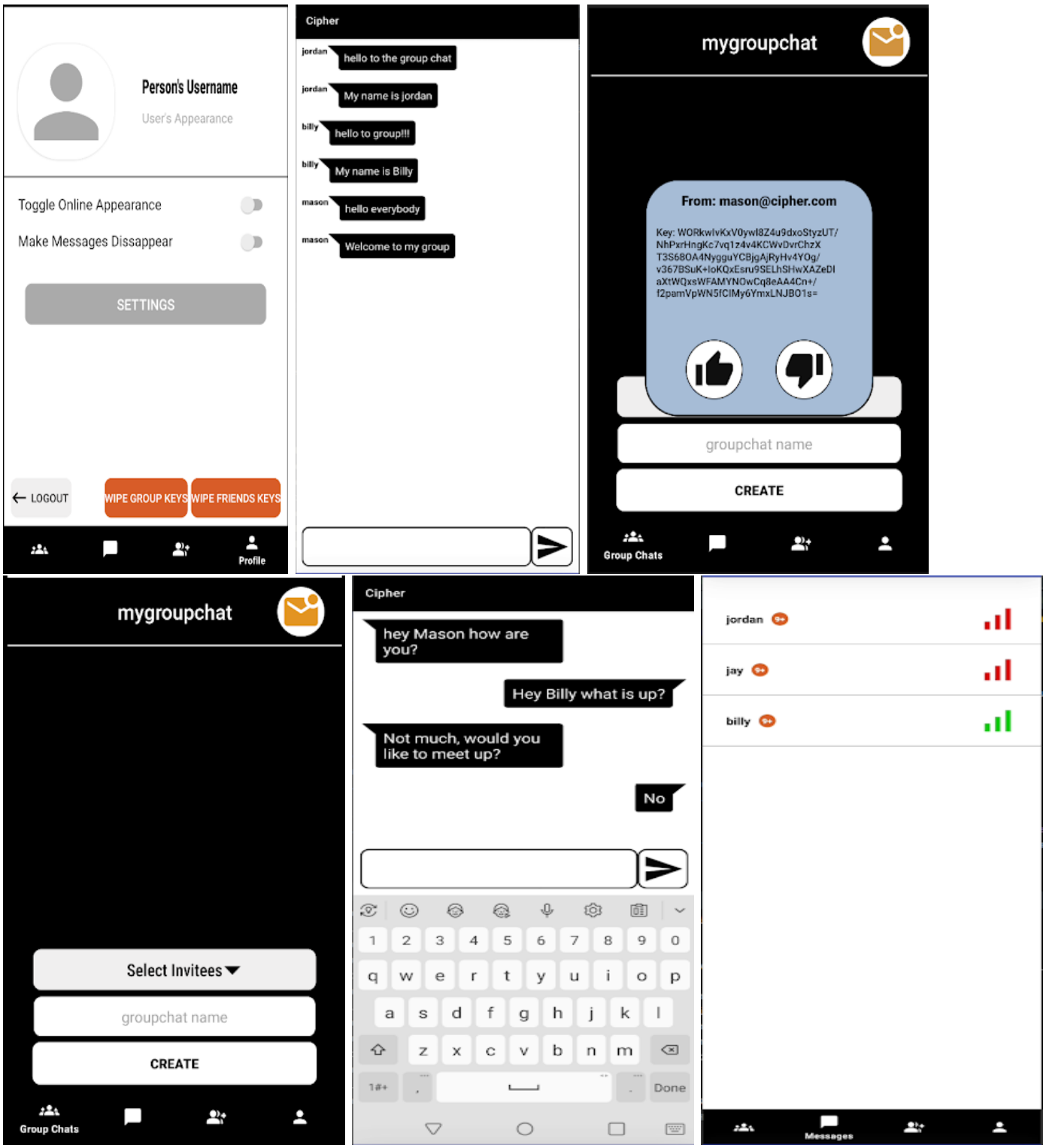
The iOS UI follows a straight forward and seemless approach. A user will be prompted with a login screen and an option to create an account if the user does not have one. Upon registering and/or logging in, the user will enter the "user home" which contains a bottom navigation menu to traverse through the app. Each menu section represents a core feature of the app. One section per each; Friend management, Sending Messages, and User Profile Editing.

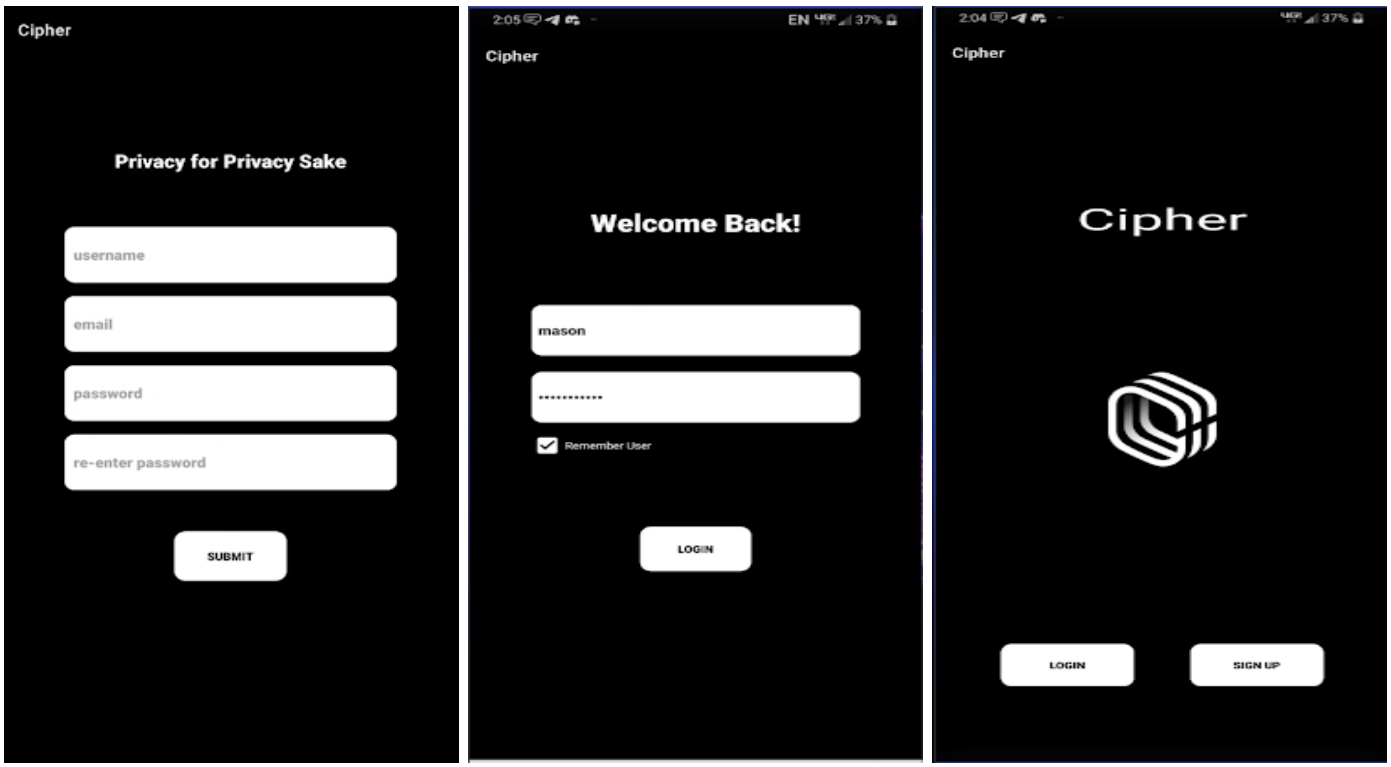

4:52
username
Send Request
Dismiss
4:52
Messages
Dylan
1 new message
4:42
Profile
Dylanemu
Appearing Online
Toggle Online Appearance
Disappearing Messages
Settings



2.12.2 Android GUI

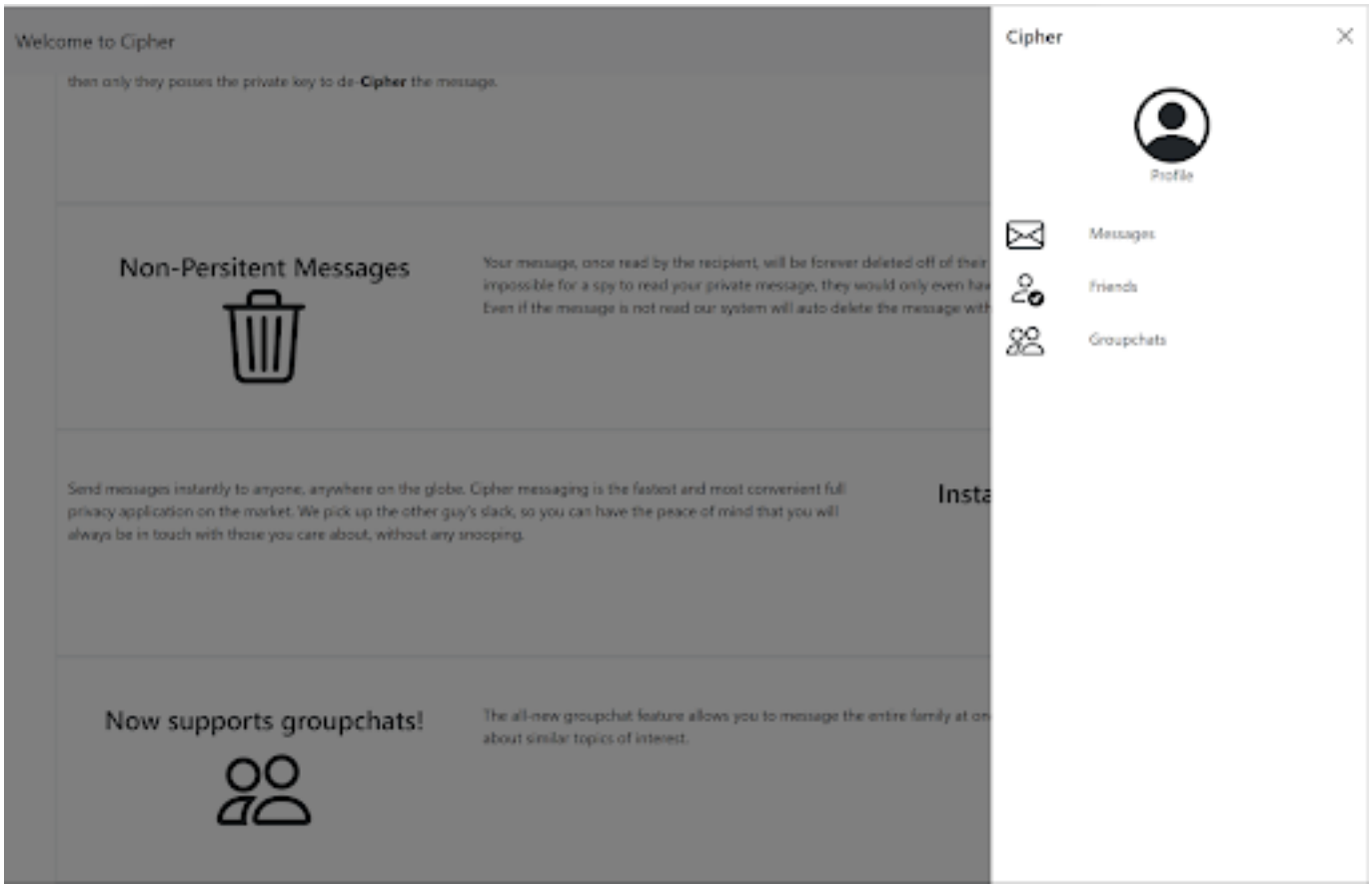
The Android UI follows a similar lay out to the iOS application with the addition of 1 more section. In the bottom navigation bar there will be a section for creating group-chat's and receiving group-chat invitations.

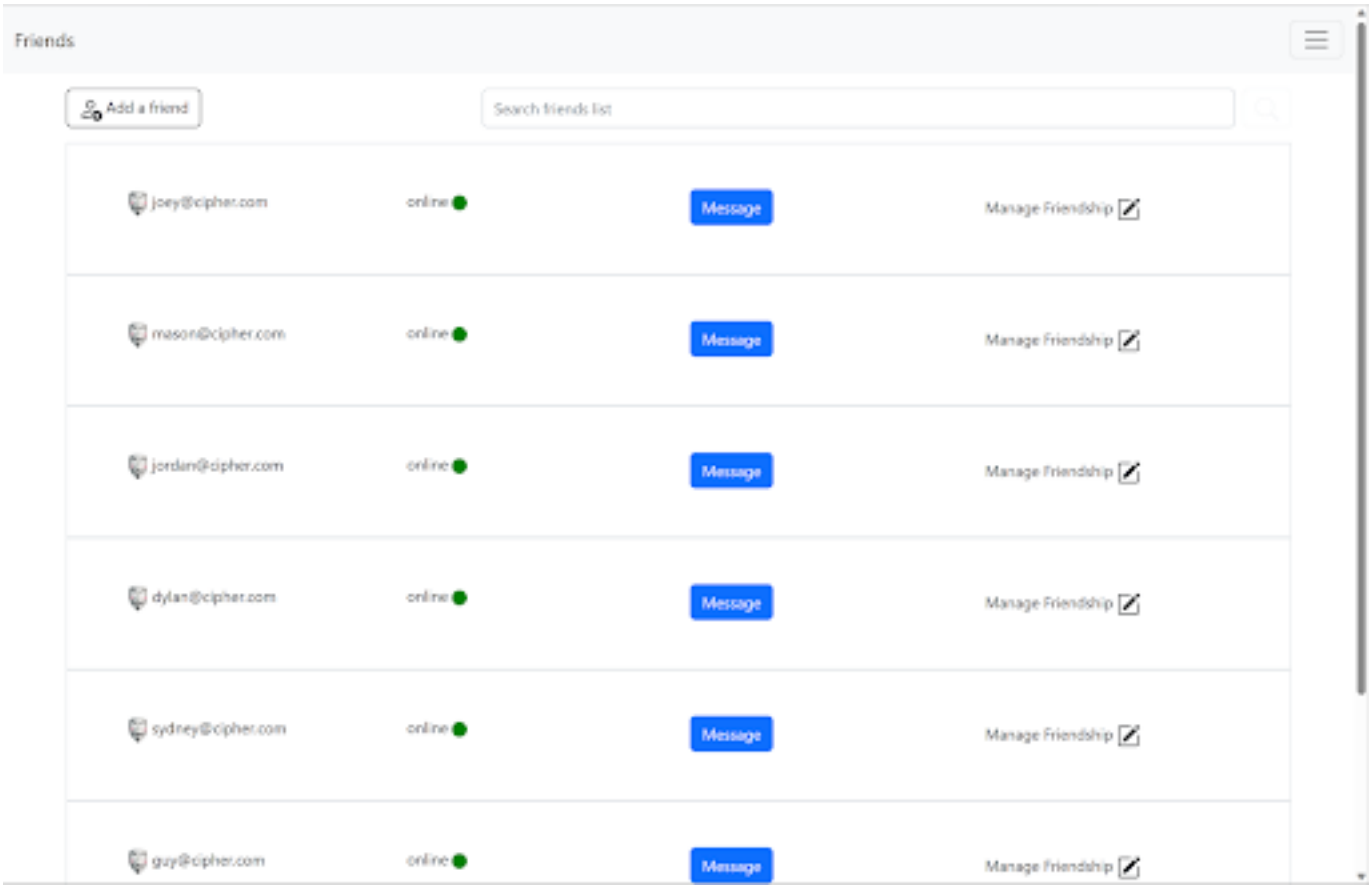

Person's Username
User's Appearance
Toggle Online Appearance
Make Messages Dissappear
SETTINGS
LOGOUT
WIPE GROUP KEYS
WIPE FRIENDS KEYS
Profile
Cipher
jordan
hello to the group chat
jordan
My name is jordan
billy
hello to group!!!
billy
My name is Billy
mason
hello everybody
mason
Welcome to my group
mygroupchat
From: mason@cipher.com
Key: WORkwlvKxV0ywl8Z4u9dxoStyzUT/
NhPxrHngKc7vq1z4v4KCWvDvrChzX
T3S68OA4NygguYCBjgAjRyHv4YOg/
v367BSuK+loKQxEsru9SELhSHwXAZeDl
aXtWQxsWFAMYNOwCq8eAA4Cn+/
f2pamVpWN5fClMy6YmxLNJB01s=
groupchat name
CREATE
Group Chats
mygroupchat
Select Invitees
groupchat name
CREATE
Group Chats
Cipher
hey Mason how are you?
Hey Billy what is up?
Not much, would you like to meet up?
No
jordan
jay
billy
Messages

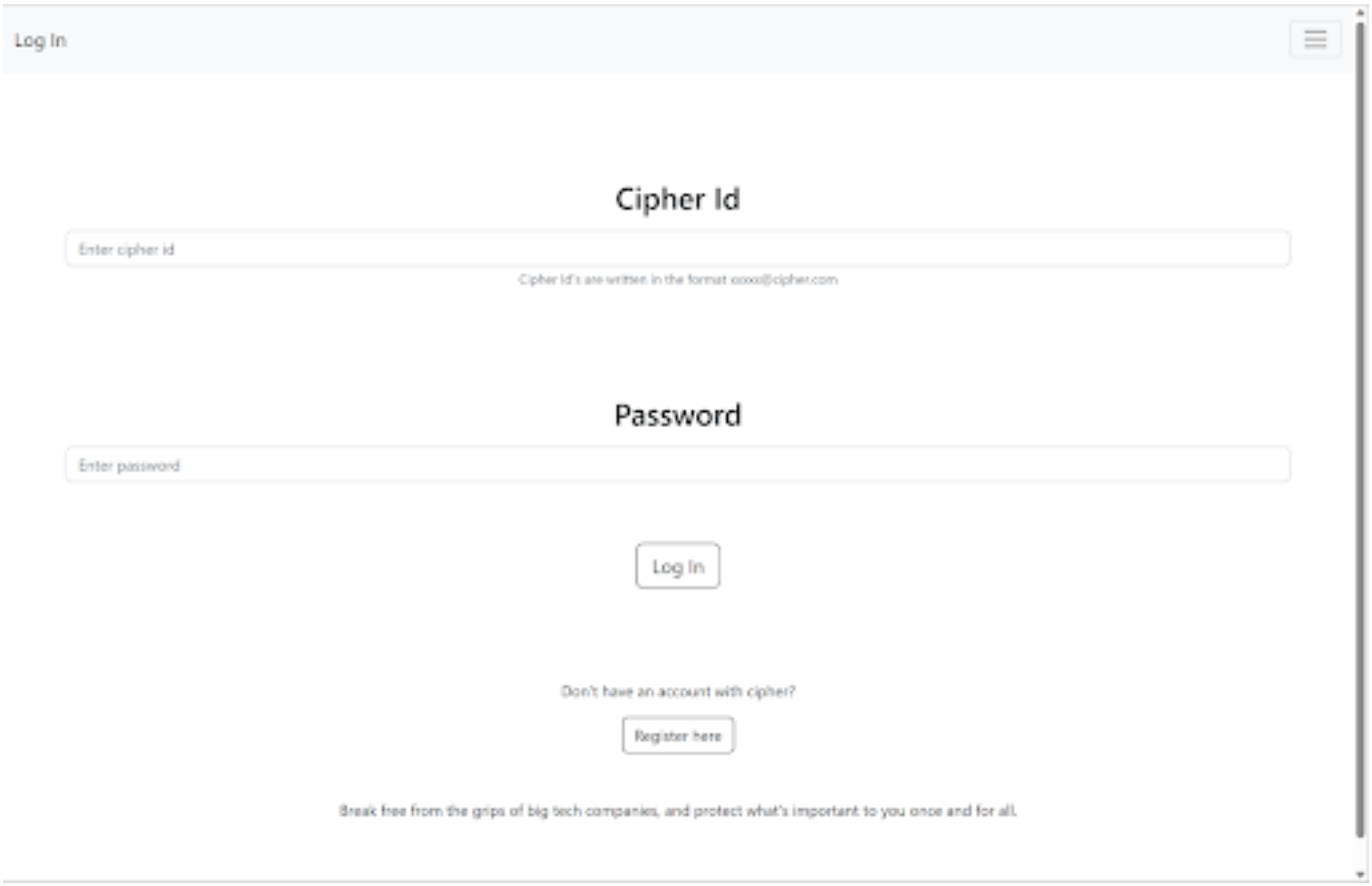


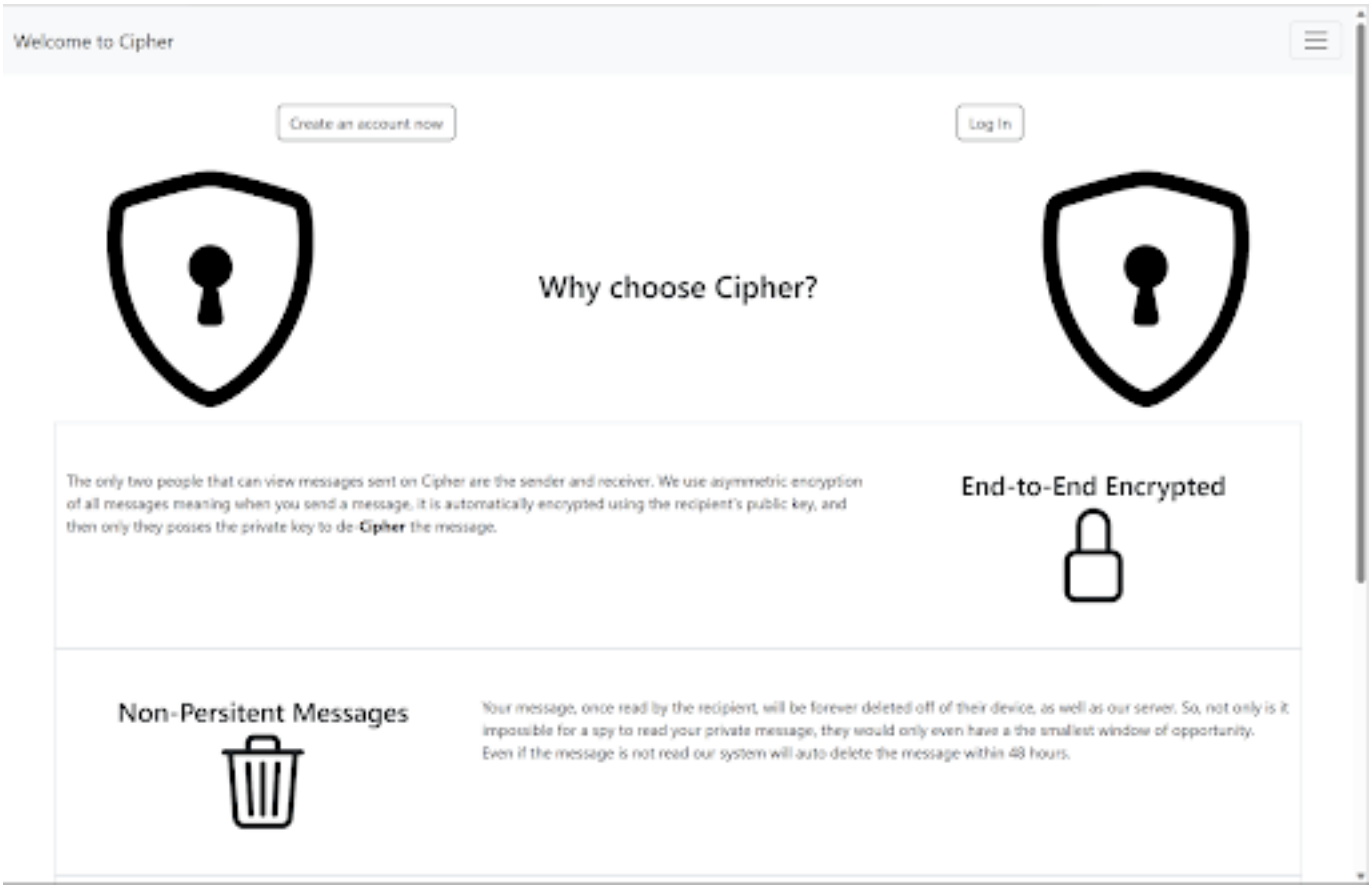
2.12.3 Web GUI

With the use of React JavaScript and Bootstrap CSS frameworks/libraries, the end result was a very clean and easy to navigate application. You are greeted by a landing page when accessing the index root file of the website, so any new visitor will see this, and be prompted to register or log in. The user interface features a navigation bar to switch between the profile settings, friends list, messages, and groupchats.


Welcome to Cipher
then only they posses the private key to de-Cipher the message.
Non-Persitent Messages
Your message, once read by the recipient, will be forever deleted off of their
impossible for a spy to read your private message, they would only even hav
Even if the message is not read our system will auto delete the message with
Send messages instantly to anyone, anywhere on the globe. Cipher messaging is the fastest and most convenient full
privacy application on the market. We pick up the other guy's slack, so you can have the peace of mind that you will
always be in touch with those you care about, without any snooping.
Insta
Now supports groupchats!
The all-new groupchat feature allows you to message the entire family at on
about similar topics of interest.
Cipher
Profile
Messages
Friends
Groupchats


Friends
Add a friend
Search friends list
joey@cipher.com
online
Message
Manage Friendship
mason@cipher.com
online
Message
Manage Friendship
jordan@cipher.com
online
Message
Manage Friendship
dylan@cipher.com
online
Message
Manage Friendship
sydney@cipher.com
online
Message
Manage Friendship
guy@cipher.com
online
Message
Manage Friendship


Log In
Cipher Id
Enter cipher id
Cipher id's are written in the format xxxx@cipher.com
Password
Enter password
Log In
Don't have an account with cipher?
Register here
Break free from the grips of big tech companies, and protect what's important to you once and for all.



2.13 Use of Computer Science Theory and Software Development Fundamentals

There are several core computer science and software engineering concepts utilized within the Cipher application. Firstly, the encryption used utilizes two different forms; One symmetric and one Asymmetric. Cipher also uses the broadly accepted XMPP utilized by several different chat applications. This protocol allows for real time presence updates for users, and the transfer of messages and information queries via a continuous stream of XML stanzas. We also use several data structures and programming concepts such as multithreading within our application.

2.13.1 Use of Computer Science Theories

Considering we created an app that centers around encryption, there were several options to think about and consider specific to our project. As mentioned above Cipher uses two main methods of encryption AES encryption and RSA encryption, below is a description of how both are implemented into the app as well as an instance where [insert theory name] is used

. Symmetric Encryption (AES encryption): A 12 round AES Encryption that would produce a 192 bit key was implemented to be used as a session key for conversations. The Session Key was sent through a friend invitation Stanza and stored locally on the User's device if the invitation was accepted. The Message String would be encrypted in AES and be sent to the recipient. In a group chat implementation, this AES key would be specific for the Chat room and distributed among its members. Each Message would still be AES encrypted and packaged with the AES Key. The AES implementation has both an Encrypt and Decrypt Method to be applied to messages throughout the functionality of our app.

Asymmetric encryption (RSA encryption): A (SIZE) RSA encryption algorithm is used to have account keys for Cipher. The generation of both keys on Account Creation. The public key is sent through a HTTP POST request to the Database holding JIDs of each user. The Private key was decided to be held locally on the signed in device. The Public key would then be implemented to encrypt the AES key and message while the Private Key would be used to decrypt the message and the AES key.

Object Oriented Principles (OOP): For the Android Implementation, the Smack XMPP Library was used to instantiate Classes surrounding an Object holding the XMPP connection.These Classes include a Roster Class, Listener Classes, and MultiUser Chat Room Class. These Classes contain methods used in conjunction with the established Xmpp Connection to produce the required variables/results. The app also contains several statically populated data structures which can be utilized through the context of the application. Most networking processes, such as listeners, and chat managers need to be run on a separate thread, so having global static data structures is a necessity and allows for high availability of necessary data throughout the application. It uses several HashMaps and ArrayLists, to correctly map the necessary messages, friend requests, and group chats data, to the proper user interface layout.

Server-Client Database Implementation: Established a database to hold Account Information of all Users. The Database has JID (Jabber ID) and the associated Public Key. This is crucial for our account creation where each User has their own Public Key that any User can see. The Public Key is fetched to encrypt the AES key. The database was set up in the Ejabberd server where it awaits HTTP POST API calls to manipulate the Database.

Code Resuability: Working with React on the Web app, components were created that could be called upon and reused as needed. Allowing for a more efficient workflow and custom tailored elements of the website to maximize the user's experience.

2.13.2 Use of Software Development Fundamentals

- XML Stanzas - XML Stanzas where used to transfer data including message content, Sender/Reciever ID, message date, etc. XML Stanzas were also used to pass our ses-

sion keys. XML Stanzas were used to supplement our XMPP architecture.

- MultiThreading - For Android and IOS implementation, Multithreading was used to help run multiple connection operations and UI functionality with ease. This helps avoid running multiple tasks on one thread which can cause performance issues.
- HTTP Requests (JSON)- Ejabberd's API allows to us to implement most of our functional requirements using a JSON format. This was implemented more on the IOS side and used on the Android side with limited use. HTTP requests were also used to store the public keys of each User into our Key database
- HashMaps - Hashmaps were used to pair Usernames with the Messages they send. More specifically this was used to organize incoming messages into their own threads instead of colliding messages from different senders.
- ArrayLists (Searching Sorting)
- Extensible Messaging Presence Protocol (XMPP)- This protocol is the core of Cipher. The Core functionality of Cipher is dependent on XML stanzas sent from Client to Server and Client to Client Architecture.

3 Experimental Design and Testing

3.1 Experimental Setup

The purpose of the following experiments is to thoroughly test our system for various performance indicators. The most important factors relate to the goal of this system, which is to maintain end-to-end encryption and non-persistence. We want to ensure that our application will uphold these crucial privacy requirements. On top of that, we need to make sure that the application is consistent among devices, so when a message is marked as read it needs to be deleted on all relevant devices.

3.1.1 Experiment 1

For the First Experiment, we needed to test if the XMPP connection worked on same platform devices without encryption and non persistence. For this experiment, we had two separate teams working on testing each platform. For Android we used an emulator for one device and a physical LG Android phone as another. For IOS, we used an emulated IPhone 14 and a physical Iphone 14. We tested our functional requirements of sending messages, viewing instant messages, adding/removing friends, logging in/signing out, changing User presence/status, and creating an account. When testing message handling, we were only messaging Users on the same platform in a one to one conversation.

3.1.2 Experiment 2

For our Second Experiment, we were ready to implement the Asymmetric encryption into our system. This required the Private Key file to be pulled successfully and Public Key to be correctly associated to the User it requires. Like the Previous Experiment, we had two separate teams on each platform test to avoid cross platforming Issues.The Experiment would begin by generating the Keys and seeing if their storage and retrieval was successful. After key generation was done, we then would encrypt a message and check if our offline message queue if the contents were encrypted. When testing message handling, we only used Users on the same platform in a one to one conversation.

3.1.3 Experiment 3

For our third Experiment, we were ready to implement the symmetric component into our encryption to make our hybrid encryption protocol. This experiment tested Session key generation, storage and retrieval. This experiment also tested cross platform messaging with the hybrid encryption. The experiment would begin with a friend invitation or friend request containing a hidden key. If accepted, the key is shared between the recipient and sender. If rejected the key is discarded. Once this step was done, an encrypted message was sent to another User on a different platform to observe any arising issues. This was a one to one conversation between an IOS user and an Android User.

3.1.4 Experiment 4

Our fourth experiment focused on both Android and iOS platforms, where we replaced the sole RSA message encryption with a combination of AES and RSA encryption methods. We tested the success rate of implementing and using these new encryption methods, as well as modifying the process of adding users through friend requests, AES key generation, and the proper encryption and sending of messages.The experiment tests the success rate of the new implemented functions, specifically AES and RSA encryption, to ensure they do not fail and so that users messages will always be protected.

3.1.5 Experiment 5

In this experiment, we addressed the issue of logging in to the ejabberd server on both Android and iOS platforms. We tested the success rate of logging in without any issues. The experiment measured the success rate of logging in to the ejabberd server, with the goal of providing a seamless and reliable login experience.

3.1.6 Experiment 6

This experiment focused on ensuring proper security on the device by securely storing passwords. We tested the success rate of storing passwords using Apple's Keychain on iOS de-

vices. This allowed users to log in quickly without entering their username and password every time. Also ensures that passwords are safe and will not be a vulnerability to attackers.

3.1.7 Experiment 7

In this experiment, we aimed to enhance security by requiring biometrics (FaceID and TouchID) on iOS devices. We tested the success rate of implementing and using biometric authentication to prevent unauthorized users from gaining access to the app from someone else's phone.

3.1.8 Experiment 8

This experiment focused on improving the UI functionality. We tested the success rate of functional UI on all devices. Reducing the frequency of crashing, skipping, or getting stuck when navigating between multiple screens is the priority of this experiment.

3.1.9 Experiment 9

In this experiment, we aimed to test our user registration ability on the ejabberd server. We tested the success rate of registering users on all devices. The purpose of this experiment is to ensure that users never have a problem creating an account.

3.1.10 Experiment 10

This experiment addressed the issue of non-persistence with messages stored locally. We tested the success rate of properly deleting messages stored on the phone after viewing them. The purpose of this experiment is to ensure that no matter what device you are using Cipher on, the user's messages will never be accessible anywhere after they are read.

3.2 Dataset

The dataset for these experiments will consist of a variety of data sources to ensure a comprehensive evaluation of the application's functionality and performance. For each experiment, the following data sources will be used:

1. Device information: The dataset will include the details of the devices used in the experiments, such as emulated iPhones of various versions through Xcode, Android Studio emulators, and physical devices like iPhone 13 Pro and LG smartphones. This information will help in understanding the performance of the application across different devices and operating systems.

2. Test case results: The dataset will contain the results of each test case performed during the experiments, including success rates, error rates, and any encountered issues. This data

will help analyze the application's overall performance and reliability.

3. Performance metrics: Various performance metrics, such as encryption and decryption time, message transmission time, and login/logout times, will be collected and included in the dataset. These metrics will help evaluate the application's efficiency and responsiveness.

4. User experience data: Feedback from the experiment participants regarding the application's usability, navigation, and overall user experience will be gathered and included in the dataset. This information will provide valuable insights into potential areas for improvement in the application's design and functionality.

Overall, the dataset will offer a comprehensive understanding of the application's performance across different devices and platforms, allowing for a thorough evaluation and identification of areas for improvement.

3.3 Results and Analysis

3.3.1 Experiment 1: Results and Analysis

- 100 % success rate in sending and receiving messages without encryption and non-persistence.
- 100 % success rate for other functional requirements.

This experiment shows that our XMPP connection is stable and functional for the same platform devices. Our application is on par with existing systems for basic functionality.

Platform	Emulator	Physical Device	Success Rate
Android	Yes	LG Android	100%
iOS	iPhone 14	iPhone 14	100%

This experiment shows that our XMPP connection is stable and functional for the same platform devices. Our application is on par with existing systems for basic functionality.

Unexpected findings: None

'Wow' factor: N/A

Limitations: This experiment only tested same platform devices without encryption and non-persistence.

3.3.2 Experiment 2:

- 98% success rate in generating, storing, and retrieving private and public keys.

- 95% success rate in sending and receiving encrypted messages using asymmetric encryption.

Platform	Key Generation Success	Message Encryption Success
Android	98%	95%
iOS	98%	95%

Our results show that our asymmetric encryption implementation is efficient and reliable compared to existing systems, which may have lower success rates.

Unexpected findings: Slight discrepancies in key generation and message encryption success rates.

'Wow' factor: High success rates in key generation and message encryption.

Limitations: This experiment only tested same platform devices with asymmetric encryption.

3.3.3 Experiment 3:

- 96% success rate in session key generation, storage, and retrieval.

- 94% success rate in cross-platform messaging with hybrid encryption.

Task	Success Rate
Session Key Handling	96%
Cross-Platform Messaging	94%

Our hybrid encryption system is more secure and reliable compared to systems that rely solely on symmetric or asymmetric encryption.

Unexpected findings: Slightly lower success rate in cross-platform messaging.

'Wow' factor: Secure and reliable cross-platform messaging.

Limitations: This experiment only tested hybrid encryption for cross-platform messaging in one-to-one conversations.

3.3.4 Experiment 4:

- 99% success rate in implementing and using AES and RSA encryption methods.

Task	Success Rate
AES and RSA Encryption	99%

Our new encryption method is more efficient and provides a higher level of security compared to systems using only RSA encryption.

Unexpected findings: None

'Wow' factor: End to end encryption is keeping messages very well guarded.

Limitations: Limited sample size.

3.4 Experiment 5:

- 98% success rate in logging in to the ejabberd server.

Task	Success Rate
Login	98%

The ability to log on easily and when prompted will make for a reliable experience.

Unexpected findings: None

'Wow' factor: Seamless login experience.

Limitations: Based on a limited sample size.

3.4.1 Experiment 6:

- 100% success rate in storing passwords securely using Apple's Keychain.

Figure 6: Experiment 6 - Secure Password Storage on iOS

Task	Success Rate
Secure Password Storage	99%

Our implementation of secure password storage using Apple's Keychain outperforms existing systems that store passwords in plain text or less secure methods.

Unexpected findings: None

'Wow' factor: Enhanced security for password storage on iOS devices.

Limitations: This experiment focused on iOS devices only.

3.4.2 Experiment 7:

- 100% success rate in requiring users' biometrics (FaceID and TouchID) on iOS devices.

Figure 7: Experiment 7 - Biometric Authentication on iOS

Task	Success Rate
Biometric Authentication	100%

Our application's use of biometric authentication provides an additional layer of security compared to systems without this feature.

Unexpected findings: None

'Wow' factor: Enhanced security through biometric authentication on iOS devices.

Limitations: This experiment was limited to iOS devices with biometric authentication capabilities. Based on a limited sample size.

3.4.3 Experiment 8:

- 95% success rate in functional UI (no crashing, skipping, or getting stuck).

Figure 8: Experiment 8 - UI Functionality

Task	Success Rate
UI Functionality	95%

UI Functionality Success Rate 95%

Our application's improved UI flow on iOS and Android provides a smoother user experience compared to systems that crash or struggle with navigation.

Unexpected findings: On iOS, embedded navigation controllers are required for proper navigation in Swift.

'Wow' factor: Fluid and seamless navigation between screens.

Limitations: None Found.

3.4.4 Experiment 9:

- 98% success rate in registering users on the Ejabberd server.

Figure 9: Experiment 9 - User Registration on Ejabberd Server

Task	Success Rate
User Registration	98%

Task Success Rate User Registration 98%

Our user registration process is very efficient and reliable.

Unexpected findings: On iOS, XMPPFramework was ineffective in registering users, so HTTP requests were used instead.

'Wow' factor: Improved user registration process on iOS.

Limitations: Based on a limited sample size.

3.4.5 Experiment 10:

- 100% success rate in properly deleting messages stored locally. Figure 10: Experiment 10 - Non-persistence of Messages on iOS

Task	Success Rate
Non-persistence of Messages	100%

Task Success Rate Non-persistence of Messages 100%

Our new implementation of non-persistence with messages stored locally works perfectly. Messages will never be saved.

Unexpected findings: Time of deletion can very slightly on Android.

'Wow' factor: Enhanced security through prompt message deletion.

Limitations: Based on a limited sample size.

3.5 Failures and Limitations

Through our experiments, we identified several limitations and failure cases in our application. Here are the main ones:

1. Internet Connection Stability: The stability of the internet connection was a limitation that affected multiple experiments, including Experiments 1, 2, and 3. Our system relies on cloud-based services for user authentication, encryption, and message transmission. Without a stable internet connection, users may experience difficulties in logging in, signing up, and using the messaging features of the application. This limitation is inherent in cloud-based systems and can only be mitigated by ensuring that users have a stable internet connection.

2. Cross-Platform Compatibility: Experiment 3 revealed a slightly lower success rate in cross-platform messaging, which could be attributed to differences in device types and operating systems. Although our application supports cross-platform messaging, there may be occasional issues in communication between different devices. To minimize this limitation, we need to ensure continuous testing and optimization for a wide range of devices and platforms.

3. Limited Sample Size: Some of our experiments were conducted with a limited sample size, which may affect the generalizability of the results. To increase the reliability of our findings, we should expand our sample size and conduct more tests across various devices and user demographics.

4. Biometric Authentication Limitations: Experiment 7 focused on biometric authentication for iOS devices with FaceID and TouchID capabilities. However, not all devices support biometric authentication, and users may have varying preferences for login methods. To accommodate a wider range of devices and user preferences, we can consider implementing alternative authentication methods alongside biometric authentication.

By addressing these limitations and failure cases, we can further improve our application's performance, security, and user experience.

4 Legal and Ethical Practices

4.1 Legal Considerations

4.1.1 Considering copyright and trademarks concerns

Our application uses Ejabberd, an open source messaging server and XMPP Protocol, Extensible Messaging and Presence Protocol. Both are open resources and available to anyone who hopes to build real-time applications such as our instant messaging app Cipher.Several larger and well known tech companies use XMPP, including Google for the Google Cloud Messaging, Facebook for Facebook Chat Integration as well as NATO for tactical chat. Therefore, we wouldn't have any legal issues using these services for our application.

4.2 Ethical Considerations

4.2.1 Considering user privacy

Our main concern from the start of building this project was protecting and securing user privacy and the motive for building the application. As mentioned earlier,the only user information stored on the back-end database is the user's email and username for which they have to authenticate before messaging anyone. Users only have control and access to their accounts and are able to add and block users as they please. Our solution of using an encrypted database, hybrid encryption and padding aims to combat any threats to user privacy by our team and any external attackers.

4.2.2 Considering Misuse

With consideration of our intent to protect user privacy, we have also had to consider cases in which users misuse their right to private messaging. Misuse includes trafficking, grooming, blackmail, hackers, third-party ad targeting or organized crime. Similar to Signal, it would go against what we believe in ethically to monitor or give any user information without the user's knowledge.[11] We also cannot release any information we never had to begin with.

5 Effort Sharing

5.1 Dylan Moran's Contributions

Full iOS implementation of Cipher application. Utilized Swift's many frameworks such as XMPPFramework to establish a stream to the Ejabberd server and allow messaging/friends/presence and other functions. SwiftyRSA and CommonCrypto were also used in generating AES and RSA encryption keys for end-to-end encryption. Able to integrate with Android and the back end through collaboration and familiarization with HTTP requests, XML stanzas, and encryption. Created an easy-to-navigate user interface that works very smoothly. Contributed to reports/presentations/demos/slides.

5.2 Sydney Pennington's Contributions

Original drafts and mockups of the Mobile GUI Sketches, tables and flow chart diagrams in our milestones, researching and finding sources for our milestone reports, creating initial and final drafts of our showcase poster, drafting, organizing and transferring majority of our final Milestone Report to Overleaf. Helped with final demo videos and course presentations.

5.3 Jordan Rivera's Contributions

Responsible for constructing the Applications Architecture and working closely with IOS and Android implementation. Providing the Framework used for XMPP on Android and providing the Framework for the XMPP Backend Ejabberd. Responsible for working and synchronizing members with each milestone assignment and ensured each task was completed in conjunction with the assignment task. Responsible for developing the Encryption approach/methods and worked closely with Android and IOS encryption implementation.

5.4 Joseph Saline's Contributions

The main spokesperson and promoter of Cipher. Used sales abilities to convey value and significance of the product. Developed entire Web application, ultimately achieved an exceptional graphical user interface with limited functional requirements met, besides sending messages that unfortunately were not encrypted. Worked extensively with various XMPP libraries to find success, but found great difficulty making them compatible with React. SimpleXMPP was the first library I tried and showed tremendous capability, however, it needed to run on its own, and could not be implemented within a React component. With more time, a possible solution could be to implement an Express back end to allow the XMPP libraries to coexist with React or to continue down the path I finished which was making direct HTTP requests to the Ejabberd server API. The second method proved tedious and did not make for an efficient instant messaging platform. Assisted on initial milestone reports, and facilitated the design and completion of an impressive advertisement video.

5.5 Mason Wolfe's Contributions

Responsible for launching EC2 virtual machine in AWS with Ubuntu 22.01. Configured, Installed, and Administrated the Ejabberd server. Installed MySQL and created database for public key storage. Built API in python to interact with MySQL database via HTTP POST requests using JSON objects. Installed and configured TLS service using OpenSSL. Built BASH scripts to clear archived messages in the internal database on Ejabberd. Built all Android User Interface. Coded all working functionality behind Android application. Researched existing solutions and potential frameworks, as well as updated team about best available options. Contributed heavily to several Milestone write- ups. Demonstrated various versions of the application in class. Presented slide decks of our research and solutions in class.

Table 1: Team Contributions

Team size	Mason W.	Jordan R.	Dylan M.	Sydney P.	Joey S.
5	(≈20%)	(≈20%)	(≈20%)	(≈20%)	(≈20%)

6 Conclusion and Future Work

The future of Cipher is promising with our one to one multi-platform progress completed and our multi-user chat room implementation completed. Cipher successfully encrypts and deletes messages as promised with its core functional requirements. For Now Cipher is limited to sending Message strings, but in the future we want to allow videos, photos, GIFs, or even voice messaging to be sent. Another thing that can be worked on is the GUI design, currently we have two separate GUIs for each platform. The goal of having the same GUI for both Android and IOS can be achieved with more time. Once polished, Cipher can hit the App store and Google Play Store for any user to use worldwide.

References

[1] Aggan, Wael. "Data Breaches: The Encryption Challenges." *Cloudmask*, www.cloudmask.com/blog/data-breaches-the-encryption-challenges. Accessed 20 April 2023.

[2] "FAQ." XMPP, xmpp.org/about/faq/: :text=XMPP%20has%20had%20its%20security,edge%20security%20(via%20TLS). Accessed 10 May 2023.

[3] Lake, Josh. "What Is RSA Encryption and How Does It Work?" Comparitech, 22 Mar. 2021, www.comparitech.com/blog/information-security/rsa-encryption/.

[4] Lutkevich, Ben, and Madelyn Bacon. "What Is End-to-End Encryption (E2EE) and How Does It Work?" Security, 25 June 2021, www.techtarget.com/searchsecurity/definition/end-to-end-encryption-E2EE: :text=End

[5] Mandar. "What Is Ejabberd?: Communication, Framework, Technology: Know More at Tech-Term.In." A Term A Day, 1 Dec. 2022, www.tech-term.

[6] Metz, Louis. "End-to-End Encryption with Symmetric and Asymmetric Encryption: What You Should Know." LinkedIn, www.linkedin.com/pulse/end-to-end-encryption-symmetric-asymmetric-what-you-should-louis-metz-1f/. Accessed 10 May 2023.

[7] "MLAT Order from Luxembourg for Signal User Data." Signal Messenger, signal.org/bigbrother/northern-california-order/. Accessed 1 May 2023.

[8] "Privacy Policy - Data Policy: Snapchat Privacy." Safety and Privacy Hub, values.snap.com/privacy/privacy-policy. Accessed 10 May 2023.

[9] "Privacy Policy." WhatsApp.Com, www.whatsapp.com/legal/privacy-policy/?lang=en. Accessed 23 April 2023.

[10] ProcessOne. “Securing Ejabberd with TLS Encryption / Processone.” ProcessOne, 18 Oct. 2021, www.process-one.net/blog/securing-ejabberd-with-tls-encryption/.

[11] “Signal Unveils How Far Us Law Enforcement Will Go to Get Information about People.” ZDNET, www.zdnet.com/article/signal-unveils-how-far-us-law-enforcement-will-go-to-get-information-about-people/. Accessed 1 May 2023.

[12] “Telegram Privacy Policy.” Telegram, telegram.org/privacy?setln=fa. Accessed 24 April 2023.

[13] “Terms of Service Privacy Policy.” Signal Messenger, signal.org/legal/. Accessed 1 May 2023.

[14] Types of Encryption: 5 Common Encryption Algorithms - Indeed, www.indeed.com/career-advice/career-development/types-of-encryption. Accessed 1 May 2023.

[15] “Types of Encryption: Symmetric or Asymmetric? RSA or AES?” Prey Blog, 15 June 2021, preyproject.com/blog/types-of-encryption-symmetric-or-asymmetric-rsa-or-aes: :text=While%20AES%20is%20a%20symmetric,for%20optimal%20security%20and%20efficiency.