E2E security for popular web services

By

M. Zohaib Shaheen

00000172598

Supervisor

Dr.Syed Taha Ali

Department of Electrical Engineering

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Information Technology (MS IT)

In

School of Electrical Engineering and Computer Science,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(May 2019)

Approval

It is certified that the contents and form of the thesis entitled "**E2E security for popular web services**" submitted by **M. Zohaib Shaheen** have been found satisfactory for the requirement of the degree.

Advisor: **Dr.Syed Taha Ali**

Signature: ____________________

Date: ____________________

Committee Member 1: **Dr. Asad Waqar Malik**

Signature: ____________________

Date: ____________________

Committee Member 2: **Dr. Arsalan Ahmed**

Signature: ____________________

Date: ____________________

Committee Member 3: **Ms. Haleemah Zia**

Signature: ____________________

Date: ____________________

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **M. Zohaib Shaheen**

Signature: ________________

Abstract

With the ever growing number of data breaches on servers, its now more important than ever to secure sensitive information in a way that only intended parties can decrypt/understand the message even if data itself is compromised. This is specially true for government agencies and officials, who have always remained main target for attackers.

Government Officials are however not the only intended recipients. Sharing business secrets and confidential messages is also a challenge using email as medium. We intend to create an open system which can be easily adopted by masses and can ensure complete confidentiality of data as well as user identities. Our work will provide end to end encryption for emails. On top of that, sender/receiver identity will also remain hidden from email client. Only the sender and receiver will know who sent and who received. No one else! To achieve this, we created a client side application which encrypts email on client side and forward emails in a tor like way to hide sender receiver pair. Gmail is the most famous email client with 1.2 billion users and therefore we created our application to extend Gmail functionalities to include more privacy.

Dedication

To my parents,

without whom it was almost impossible for me to accomplish.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **M. Zohaib Shaheen**

Signature: ________________

Acknowledgment

I am absolutely grateful for my thesis adviser, Dr. Syed Taha Ali for his leadership in ensuring my successful development through academic achievement and for polishing my skills. The door to his office was always open whenever I ran into a trouble spot or had a question about my research or writing.

Finally, I must express my very profound gratitude to my family members and friends for providing me with unfailing support and continuous encouragement throughout my years of life.

Table of Contents

List of Figures

List of Tables

List of Symbols

Abbreviations

PGP	Pretty Good Privacy
AES	Advanced Encryption Standard
RSA	Rivest Shamir and Adleman
MIME	Multipurpose Internet Mail Extensions
S/MIME	Secure/Multipurpose Internet Mail Extensions
CA	Certificate Authority
PGP/MIME	Pretty Good Privacy Multipurpose Internet Mail Extensions
HTML	Hypertext Markup Language
HTML5	Hypertext Markup Language Version 5
OAuth	Open Authorization
Repo	Repository
POP3	Post Office Protocol Version 3
IMAP	Internet Message Access Protocol
SMTP	Simple Mail Transfer Protocol

API	Application Programming Interface
TOR	The Onion Router
EMAIL	Electronic Mail

Chapter 1

Introduction

In this chapter, we first describe the motivation behind this work. We then move to problem statement. This is followed by objective and goal of this research. Finally, we provide a roadmap of this thesis.

1.1 Motivation

Nowadays cyber attacks are on the rise and hence privacy compromise is on the peak. Recently Hillary Clintons emails were hacked despite the fact that she was a very important person and her email/data is secured by technology experts. Still the hackers were able to penetrate through her email security. The information disclosed from these emails cost her the most important elections of USA. On the other side, government agencies are involved in breaching personal space of individuals. Government backed surveillance has been making headlines all across the world. Therefore, people need a medium through which they can communication without the fear from government or black hat hacker. In 2013, 3 billion Yahoo accounts were compromised which included information like passwords, security questions, personal details and much more. The privacy is not just for people dealing with suspicious activities but its also important corporate world. KFCs secret recipe is what keeps the empire on top of world. If their recipe could have been stolen, the empire would have sunk. Hence privacy is important everywhere. Our proposed system can prove to be really helpful for exchange of sensitive information between parties specially lawyers, journalists and government official by encrypting emails using PGP before sending and forwarding emails in a tor like network preventing email clients from knowing sender receiver identities.

1.2 Problem definition

Email is the mostly widely adopted mode of communication today but unfortunately popular email providers like Gmail, Yahoo and Outlook do not support end-to-end encryption and PGP seems to be only option for users to share sensitive information. Much work has already been done on using PGP encryption for sending emails but the usability and practicality of systems for use by general public has always been a challenge [1]. Study conducted in paper "Why Johnny Still, Still Can't Encrypt" [2] shows that only 5% of the users were able to successfully send encrypted email using Mailvelope, a popular chrome extension to integrate PGP into webmail systems. Confidante [3] improves on the Mailvelope by simplifying key management using Keybase. Keybase allows users to connect their social media identities with their public key i.e. to simply user search and also allows key management i.e. creation, storage and distribution of key. Although Keybase improves the usability of Mailvelope but it still falls short on two grounds:

- Sender and Receiver Identities are exposed to webmail services
- User has to provide access to their email using oAuth and user can

decrypt/read emails only in custom designed app, which means non-zero trust on their servers since they have complete access to users' emails.

The goal is to improve Mailvelope and Confidante by establishing zero-trust model for email exchange i.e. no access to routing server and hiding sender and receiver identities using Tor [4] like network. This will not only make email encryption more accessible but will also reduce reliance on email providers like Gmail.

1.3 Objectives and goals

This thesis is aimed at following goals:

- Emails are encrypted on client side using PGP [5] before sending email, through email client, making sure only desired recipient can read the email.

- Emails are forwarded in a tor like network hiding sender and receiver identities. This ensures that only sender knows to whom the email is sent and only recipient knows from whom he received email. Not even email client involved in forwarding can find out about sender receiver pair.

- Improve usability of existing systems by providing built-in key management.

- Ensuring mass scale adaptability by building service on top of existing mail client.

1.4 Thesis roadmap

Thesis organization is as follows: Chapter 2 provides fundamental concepts about encryption techniques, application structures, client side programming, onion routing and Tor. This chapter provides the baseline for better understanding of concepts delivered in later chapters. Chapter 3 describes the work already done in field of end to end encryption for email exchange. This chapter also compares different applications in details which are working on principle of E2E encryption. In Chapter 4, we describe the construction of our application designed to enhance security and privacy of users. Chapter 5 shows evaluation of our application. This chapter covers security evaluation, usability case study and comparison of our application with other existing solutions.

Chapter 2

Background

2.1 Types of encryptions

Two types of encryptions that are mainly used:

- Symmetric
- Asymmetric

2.1.1 Symmetric

In this technique, when the data is encrypted using key. The same key is used to decrypt it. In this type of encryption, that single key must be kept safe and both sender/receiver needs to have same key [6] [7] to send and receive encrypted messages, depicted in figure 2.1.

2.1.2 Asymmetric

In this technique, a key pair is used. For encryption and decryption, different keys are used [8]. The key pair is known as public/private keys. Public key

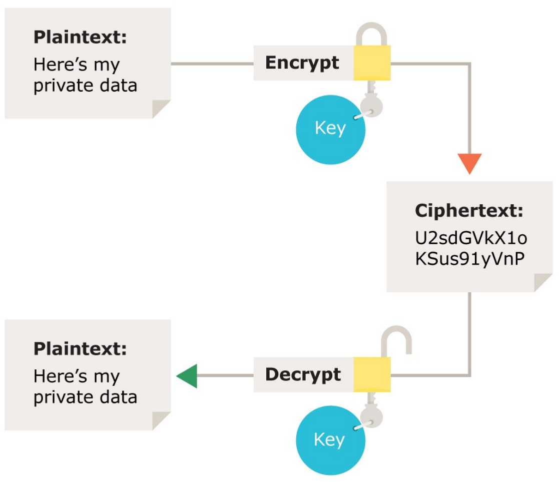


Figure 2.1: Symmetric Encryption

is used to encrypt message and in order to decrypt it, private key is used. In this case the private key must be kept hidden by owner, public key is for anyone who wants to send message to owner, shown in figure 2.2

2.2 Famous encryption algorithms

2.2.1 AES

Advanced Encryption Standard is a type of symmetric encryption [9] . AES works with AES-128, AES-192 and AES-256 i.e. 129 bit key, 192 bit key and 256 bit key. This type of encryption uses block cipher text algorithm i.e. if 128 bit key is used then for every 128 bit of message, 128 bit cypher text is generated. Similarly for 192 bit key, 192 bit of hypertext is generated for every 192 bit of message. The key bits chosen for algorithm comes at a price. If less bits are selected then process is faster but less secure as compared to longer bits. On the other hand, longer bit keys take more time in encryption making algorithm slow. As AES is a type of symmetric encryption, it means

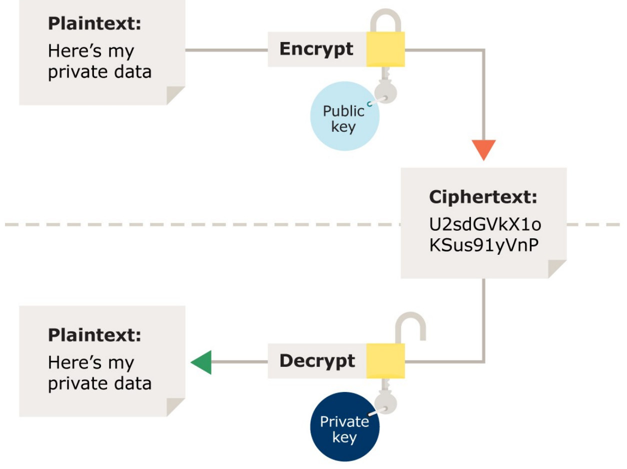


Figure 2.2: Asymmetric Encryption

only one key is used for both encryption and decryption. Hence key must be provided to recipient in order to read encrypted message and that key must never be shared with anyone else.

2.2.2 RSA

This is a type of asymmetric encryption. There are two separate keys for encryption and decryption [10]. The public key is used to encrypt message and private key is used to decrypt message. As name suggests, private key must be kept secure and never shared with anyone. In this method, sender only needs to have public key of recipient to send him encrypted message. The receiver can decrypt that message using his corresponding private key. Mostly 1024 bit and 2048 bit keys are used which make this algorithm significantly secure. In SecuriMail, OpenPGP is used which is a type of RSA cryptography.

2.3 Digital Signing

The identity of sender or recipient can be confirmed using digital signing [11] [12]. The message is signed by the private key of sender and sent away to recipient. The recipient upon receiving can verify the sender identity using corresponding public key of sender. In asymmetric encryption, if something is encrypted by using public key, only its corresponding private key can decrypt it. This process works other way around too. If something is encrypted by using private key, only its corresponding public key can decrypt it. So when the sender signs his message by using his private key, the recipient decrypts it using senders public key and upon success, he can verify sender identity. This ensures that the message is sent by actual sender.

2.4 End to end encryption

When the encryption and decryption is performed at end users and theres no central entity involved then that process is referred to as end-to-end encryption. This ensures that there is no one listening/reading except end users themselves. It ensures that even service providers cannot eavesdrop, shown in figure 2.3. S/MIME [13] [14] and PGP/MIME are most widely used for email end to end encryption. We have used OpenPGP/MIME for our thesis but we have described both below for better understanding of our choice.

2.4.1 S/MIME

This type relies on a centralized authority to decide key size used and encryption algorithm. In this type, user needs a public key and a certificate to show user trust. The certificate comprises of information about public key

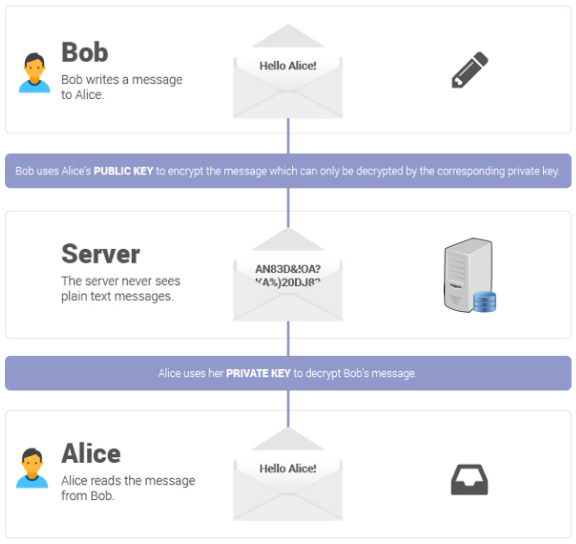

Bob
Bob writes a message to Alice.
Hello Alice!
Bob uses Alice's PUBLIC KEY to encrypt the message which can only be decrypted by the corresponding private key.
Server
The server never sees plain text messages.
AN83D&!OA?
Alice uses her PRIVATE KEY to decrypt Bob's message.
Alice
Alice reads the message from Bob.
Hello Alice!

Figure 2.3: An example of End-to-End encryption

and digital signatures signed using corresponding private key. In order to achieve this certificate, user must obtain it from certificate authority (CA).

2.4.2 OpenPGP/MIME

Pretty Good Privacy (PGP) The other heavyweight in email encryption is PGP/MIME, which is what were going to focus on in the latter part of this tutorial. You get more flexibility in how you encrypt emails, it relies on a decentralized, distributed trust model, and its fairly easy to use with web-based email clients. Its also free to get a certificate, which S/MIME is usually not. With PGP, not only can you choose how you encrypt, you can specify how well encrypted the messages you receive must be.

2.4.3 Why end to end encryption is chosen?

We have used end to end encryption in our thesis, following points describe why we opted for this.

- End to end encryptions ensures that no middle person/server can read messages [15]. If the hacker compromises the serves where your data is stored, even then they cannot decrypt that without private keys stored safely by end user. Yahoo server hack is perfect example of why we need end to end encryption.
- Privacy! The email clients can read your emails i.e Google, Yahoo can read all emails that are sent from their email accounts. End to end encryption ensures that even Gmail, Yahoo themselves cannot read your important details.

2.5 Application structure

We have developed our app using client side scripting, the reason for this choice is explained below.

2.5.1 Server side scripting

In server side programming, the program runs on server and user request is fulfilled on server and resulting data (i.e. HTML) is sent back to user after processing on server. In this type, all data is stored on server in sessions or databases. All requests are sent to server which means server can track everything user is doing.

2.5.2 Client side scripting

In client side scripting, the program runs on client side i.e. end user, demonstrated in figure 2.4. Whole process takes place on end user computer. The source code is sent from backend server to browser and user interacts with it on client side. In this type, the data is stored in local storage/html5 storage. This means that server doesnt play any role in user interactions with application. The purpose of our application is to prevent any middle node from tracking user behaviour, hence we opted for client side scripting in our application.

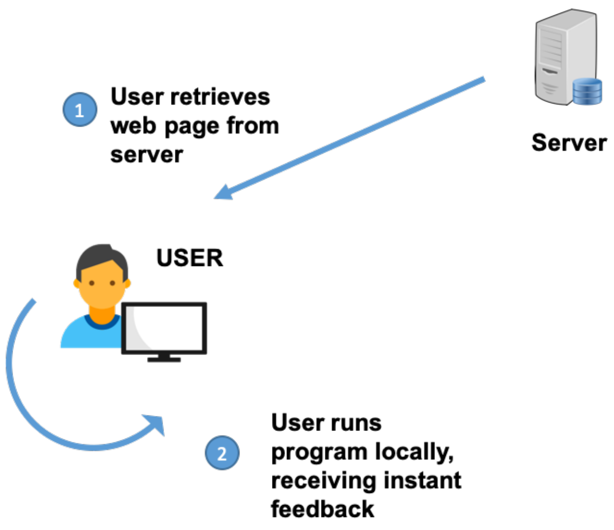

1
User retrieves web page from server
Server
USER
2
User runs program locally, receiving instant feedback

Figure 2.4: Client Side Scripting

2.5.3 Client side storages

To store data on client side, browsers provide different types of local storage APIs. Some are discussed below

Cookies

Data can be stored on client side in browser using cookies. The cookies can be set, deleted, updated using simple functions. Both old and modern browsers have support for cookies but unfortunately they have very less storage limit of 4KB. Every time request is sent to server, the cookie is attached and sent to server. This increases request time. In our case, we dont want server to read any storage items. Therefore, we didnt use cookie storage in our application.

HTML5 Local storage

This is a significant improvement over cookies [16]. The storage limit is increased to 5MB. The data is not sent to server on HTTP request. The data stored in local storage has no expiry time. It's kept in browser until deleted specifically. This is primarily used for client side scripting. We have used local storage in our application.

HTML5 Session storage

Session storage keeps in browser memory as long as browser is open. Once browser window is closed this storage is automatically deleted [17] [18]. It is also 5MB in size. Similarly, as local storage, the data is not sent to server on HTTP request.

2.6 Onion routing

In this work, the concept of onion routine is used in forwarding emails in order to hide sender and receiver identity from any middle node and Gmail.

Onion routing is used for anonymous communication in network [19]. Messages are encapsulated in layers of encryption. Hence the name taken from onion because of its layers. Without onion routine, if encrypted message is intercepted, the attacker can still know which user generated request even if he cant read that encrypted part. Thanks to onion routing, the attacker cannot even identify which user generated that request.

How it works?

In onion routing, multiple nodes take part in connection. When user generates request, the connection hops from multiple hops before finally reaching the end user/node. The end node then processes request and sends back response in same manner. Different key is used for every hop and each hop can only read what is necessary for it to forward request.

2.6.1 Tor

Tor network integrates onion routing to hide user identities from any middle nodes, service provides etc. The users Tor client gets information about all Tor nodes from central directory. Then that client chooses a random path to destination server. In order to prevent middle routing nodes from reading end user requests, the request is encapsulated in layers with different keys for encryption. Each node can only peel/decrypt its concerning layer [20]and read only necessary information which guides it to next node. If user needs to make another request to some other server, the Tor client takes a different path making it a totally random new path. Working of Tor is shown in figures 2.5 and 2.6.

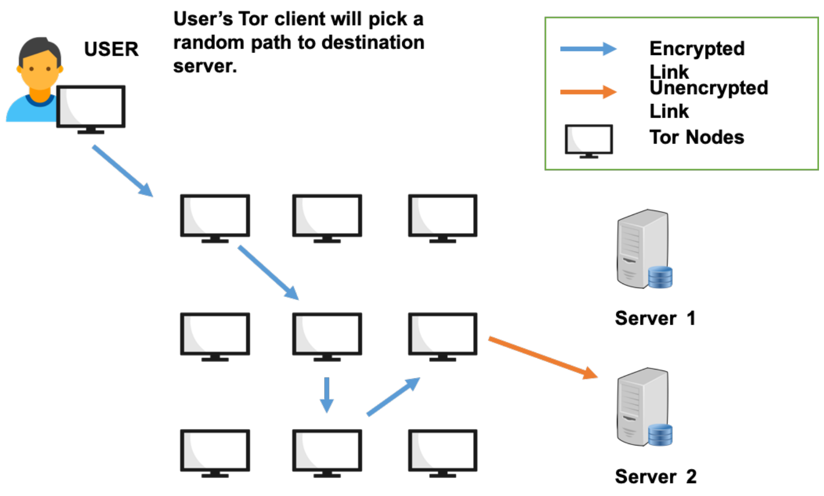

USER
User's Tor client will pick a random path to destination server.
Encrypted Link
Unencrypted Link
Tor Nodes
Server 1
Server 2

Figure 2.5: Working of Tor Step 1

Encapsulation

In human analogy, if Seth wants to message Alice. The Tor client will choose Person1, Person2 and Person3 to relay message. First the Tor client will encrypt message that only Alice can read and add extra information on top of it telling Person3 to transfer it to Alice. Then that Person3s message is further encrypted for Person2. Person2 can only read that he needs to transfer massage to Person3. Similarly, the client will add another layer, which only Person1 can decrypt and that person can only read that the message needs to be transferred to Person2.

Forwarding

When the message is transferred from Seth to Person1, Person1 decrypts its layer and finds out that it needs to transfer message to Person2. Then Person2 decrypts its layer and finds out that it needs to transfer it to Person3. Then Person3 decrypts its layer and finds address of Alice and it transfers message to Alice. So,

- Person1 only knows about Person2
- Person2 only knows about Person3
- Person3 only knows about Alice.

Problems and challenges of Tor

- Tor is susceptible to correlation attacks. If a single entity controls both entry and exit nodes for your data, it can do statistic analysis to potentially identity you i.e. your real IP address. However this scenario is only feasible in small networks with limited number of nodes.
- If an exit node is compromised or controlled by malicious entity, they can sniff the traffic and read contents over HTTP if the data is not sent over HTTPs. This can be easily prevented by always using HTTPs.

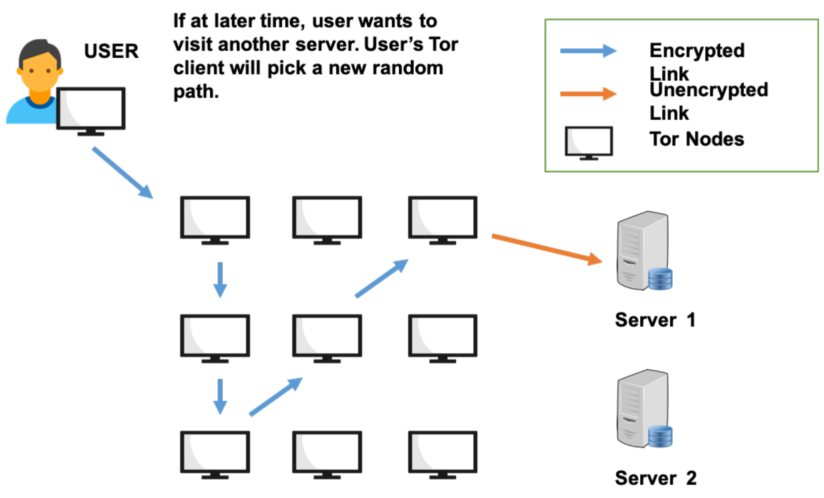

USER
If at later time, user wants to visit another server. User's Tor client will pick a new random path.
Encrypted Link
Unencrypted Link
Tor Nodes
Server 1
Server 2

Figure 2.6: Working of Tor Step 2

Chapter 3

Related Work

In this section, We will compare some of the most popular applications in use today and how they fall short in providing complete security and privacy.

3.1 Mutt Email Client

Mutt email client is a command based program capable of performing mail functions. It was initially released as early as 1995. Mutt includes support for POP3, IMAP, SMTP for sending and receiving emails. Mutt can be controlled and customised using hooks. We have mentioned Mutt here because Mutt provides support for PGP and S/MIME for encryption of emails. Mutt is used mainly by professional security companies. Its configuration is hectic and above ordinary users skills.
Every action needs to be performed using commands. Some are listed in table 3.1 so readers can have an idea about how Mutt operates. Nevertheless, Mutt is worth mentioning here because it is one of the earliest encryption tool for emails. Mutt can encrypt/decrypt emails but it doesnt hide sender/receiver identity. That means, the sender and receiver identity is not encrypted and

email client have full knowledge of who sent email and who received.

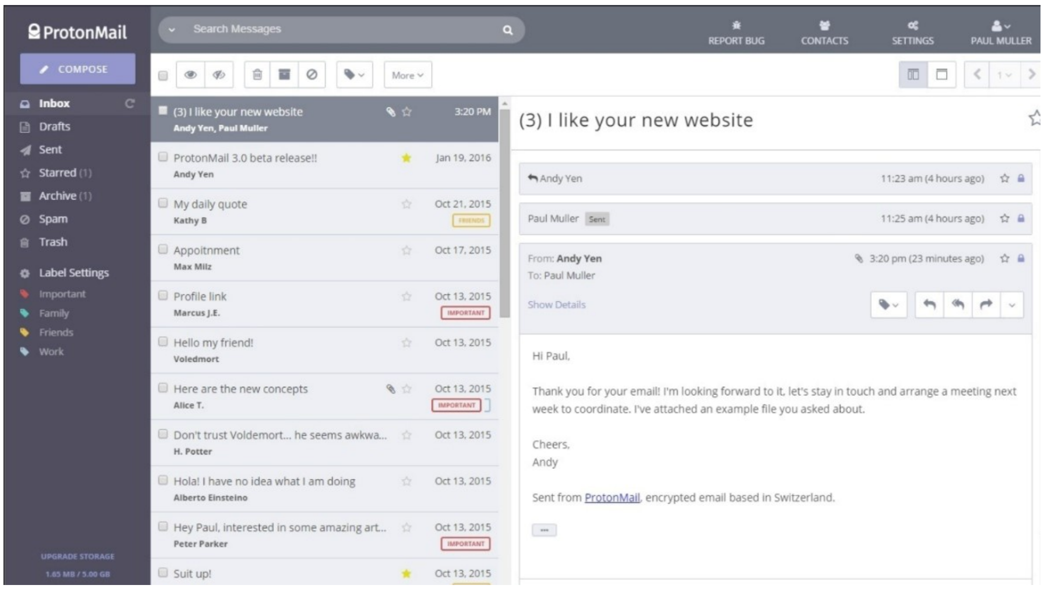
3.2 ProtonMail

It is like any regular email client e.g. Gmail, Yahoo with its custom apps for iOS, Android and web browser support. On top of email services, it provides end to end encryption using public private key pairs. The public key is stored in open directory for everyone. Any user can get public key of other user from that directory for easy communication. No need to manually share your public key with others. The private key is stored locally and its encrypted version is stored on their servers. The interface of ProtonMail is shown in figure 3.1.

The problem with ProtonMail is that its a standalone service and people have to use their email service if they want to send encrypted emails. Whereas most people prefer to use their regular email accounts for both purposes. Gmail is most famous with 1.2 billion users and people prefer to keep using Gmail instead of switching to this service. Moreover, ProtonMail doesnt hide sender and receiver identities. They claim that they do not keep log of emails sent and received but how can we researchers just trust their word.

3.3 Mailvelope

Mailvelope is one of the advanced tools for end to end encryption. It is built on top of Gmail, Yahoo, Outlook and GMX. This tool is used to encrypt emails that are sent from from famous email services instead of a separate email client like ProtonMail. Its early version was released in 2012 and after 6 years they released its first stable built in 2018. Mailvelope uses OpenPGP



(3) I like your new website	Andy Yen, Paul Muller	3:20 PM
ProtonMail 3.0 beta release!!	Andy Yen	Jan 19, 2016
My daily quote	Kathy B	Oct 21, 2015 FRIENDS
Appoitnment	Max Milz	Oct 17, 2015
Profile link	Marcus J.E.	Oct 13, 2015 IMPORTANT
Hello my friend!	Voledmort	Oct 13, 2015
Here are the new concepts	Alice T.	Oct 13, 2015 IMPORTANT
Don't trust Voldemort... he seems awkwa...	H. Potter	Oct 13, 2015
Hola! I have no idea what I am doing	Alberto Einsteino	Oct 13, 2015
Hey Paul, interested in some amazing art...	Peter Parker	Oct 13, 2015 IMPORTANT
Suit up!		Oct 13, 2015

(3) I like your new website

Andy Yen 11:23 am (4 hours ago)

Paul Muller Sent 11:25 am (4 hours ago)

From: Andy Yen 3:20 pm (23 minutes ago)
To: Paul Muller

Hi Paul,

Thank you for your email! I'm looking forward to it, let's stay in touch and arrange a meeting next week to coordinate. I've attached an example file you asked about.

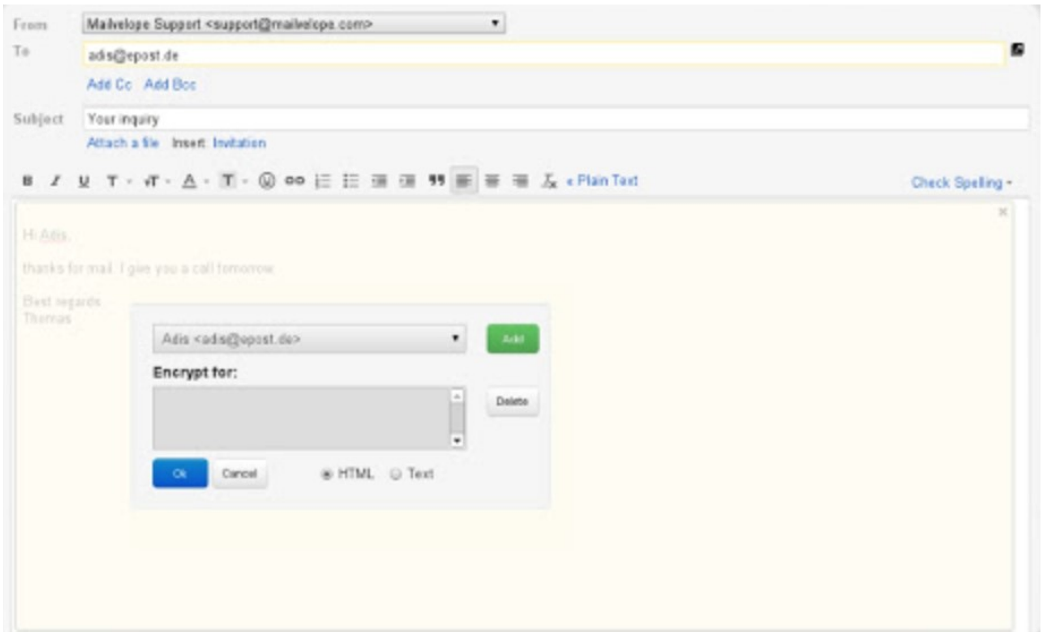
Cheers,
Andy

Sent from ProtonMail, encrypted email based in Switzerland.

Figure 3.1: ProtonMail

for encryption and decryption of emails. This application is built as an extension to web browsers. Currently they provide extensions for Google Chrome and Mozilla Firefox. Javascript is used as its core programming language. This extension functions by working directly on browsers tab where user has accessed Gmail or Outlook. It adds an extra button on compose message. If user opts this option, the user can type in this message box and the extension will encrypt that message based on recipients public key. The interface of Mailvelope is shown in figure 3.2.

At the recipients side, the encrypted email will be shown with lock image. Recipient can decrypt that with corresponding private key. The key is stored in extension memory for easy access and Mailvelope itself manages decryption process. The problem with Mailvelope is that there is no central management for keys. Users have to manually ask recipient for their public keys in order for encryption to take place. A study was published Why Johnny Still, Still Cant Encrypt [2] which evaluated the usability of this extension. During



From: Mailvelope Support <support@mailvelope.com>

To: adis@epost.de

Subject: Your inquiry

Hi Adis,

thanks for mail. I give you a call tomorrow

Best regards
Thomas

Adis <adis@epost.de>

Encrypt for:

[✓ HTML] Text

Figure 3.2: Mainvelope

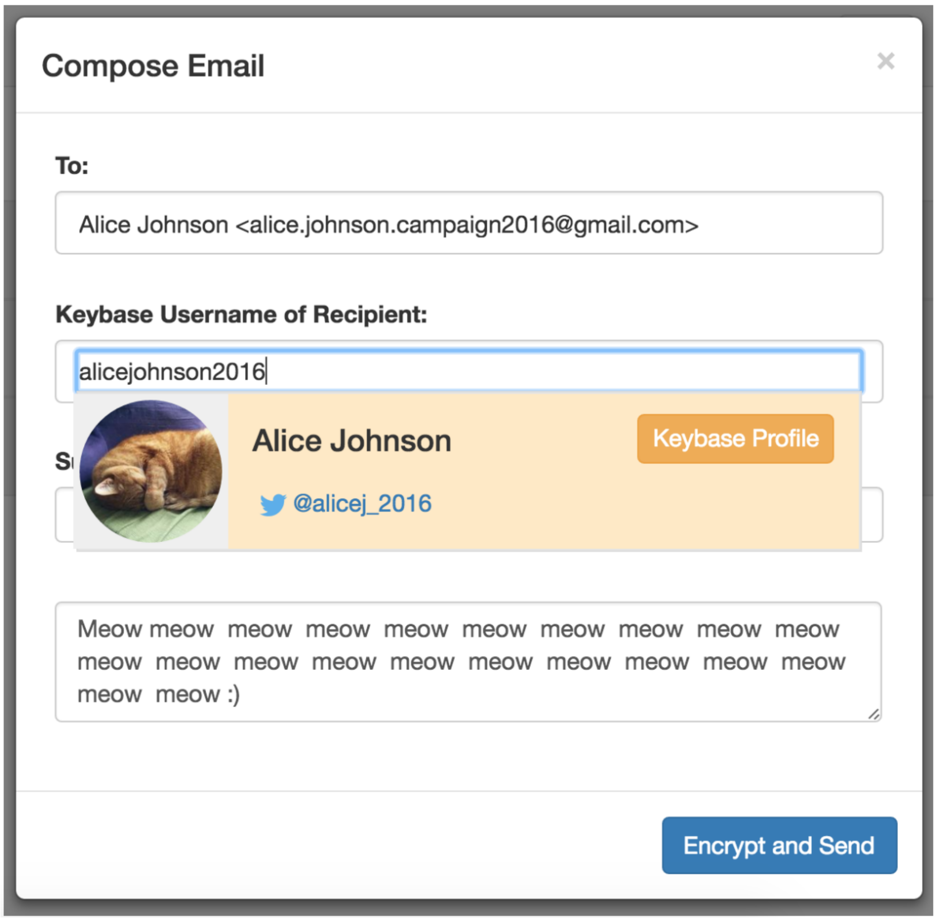
this study 20 participants were gathered to take part. They were divided in pairs to communicate. Each pair was given 1 hour to successfully send encrypted email and decrypt it at recipients side. After one hour it was found that only one pair was able to successfully communicate using Mailvelope. That is only 5 percent success rate. Later on it was found that the pair which was successful had prior knowledge of public private key cryptography. The common reason found for failure was difficulty in key generation and management. Absence of central directory for key storage and sharing. They do work email encryption for they still let client clients know about sender and receiver identities.

Table 3.1: Mutt Commands

Key	**Function**	**Description**
∧A or <Home>	<bol>	move to the start of the line
∧B or <Left>	<backward-char>	move back one char
Esc B	<backward-word>	move back one word
∧D or <Delete >	<delete -char>	delete the char under the cursor
∧E or <End >	<eol>	move to the end of the line
∧F or <Right >	<forward-char>	move forward one char
Esc F	<forward-word>	move forward one word
<Tab >	<complete>	complete filename, alias, or label
∧T	<complete-query>	complete address with query
∧K	<kill-eol>	delete to the end of the line
Esc d	<kill-eow>	delete to the end of the word
∧W	<kill-word>	kill the word in front of the cursor
∧U	<kill-line>	delete entire line
∧V	<quote-char>	quote the next typed key
<Up >	<history-up>	recall previous string from history
<Down >	<history-down>	recall next string from history
∧R	<history-search>	use current input to search history
<Backspace >	<backspace>	kill the char in front of the cursor
Esc u	<upcase-word>	convert word to upper case
Esc l	<downcase-word>	convert word to lower case
Esc c	<capitalize-word>	capitalize the word

3.4 Confidante

It uses OpenPGP to encrypt emails inside custom designed application before sending email using APIs. Confidante doesn't store keys on its servers. It Leverages Keybase for management and storage of public and private keys. Private keys after encryption are stored in Keybase. So users have to create accounts on Confidante as well as Keybase to use its encryption service. Keybase further allows users to connect accounts with their social media profiles. This way they establish more user trust regarding keys. This is shown in figure 3.3.



Compose Email

To:

Alice Johnson <alice.johnson.campaign2016@gmail.com>

Keybase Username of Recipient:

alicejohnson2016

Alice Johnson

@alicej_2016

Meow meow meow meow meow meow meow meow meow meow meow meow meow meow meow meow meow meow meow meow meow meow :)

Figure 3.3: Confidante

Confidante's web application stores authentication tokens of users provided by email clients e.g Gmail OAuth token. Hence they can access user's email account. Though they mention, they cant read encrypted emails but still if they can read all unencrypted ones we send, it questions the integrity of total privacy.

As it doesnt provide meta data protection hence identity of sender and receiver is not protected. It relies on Keybase for key management. This means it adds another third party service which has knowledge of users keys.

Chapter 4

Methodology

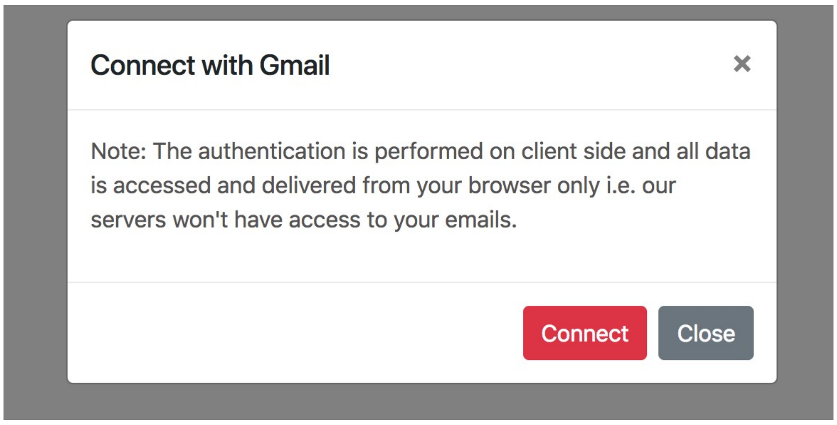
4.1 Graphical User Interface: Welcome to SecuriMail

This section is further divided into following three parts:

- Connection
- Key setup
- Email Client

4.1.1 Connection

User can send encrypted emails and hide sender and receiver identities through our application SecuriMail. The user will have to pair his Gmail account with SecuriMail. User will be prompted with connect screen. This is shown in figure 4.1. Once the user tries to connect, he will be shown message regarding privacy, Note: The authentication is performed on client side and all data is accessed and delivered from your browser only i.e. our servers won't have access to your emails. Once the user tries to connect, Gmail will show their standard oAuth modal. User can login from his desired account. No password is shared with us. Gmail only provide access to user account on clients browser. No token is stored.



Connect with Gmail

Note: The authentication is performed on client side and all data is accessed and delivered from your browser only i.e. our servers won't have access to your emails.

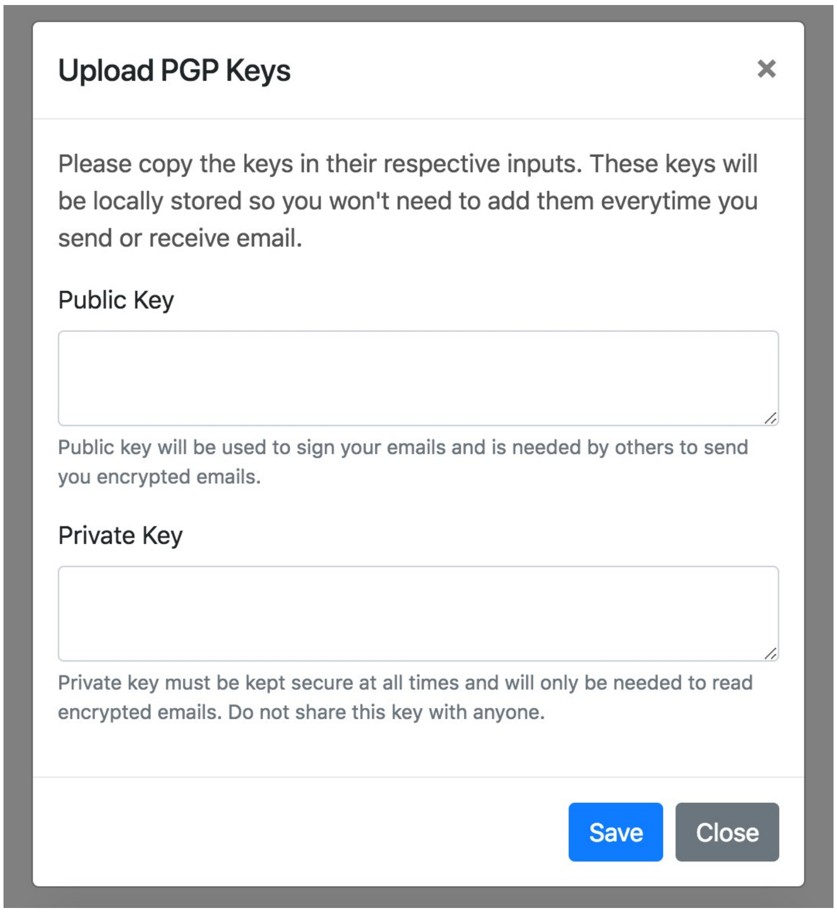
Figure 4.1: Connections

4.1.2 Key Setup

Once user has successfully logged in using his Gmail account on our application SecuriMail. He will be asked to manage keys. These are the keys which will be used for encryption and decryption purposes. User can either upload his old keys or can generate new keys using our application.

Uploading Keys

If user opts to add his old keys, he will be prompted with 2 input fields, Public Key and Private Key. These keys will be locally stored so user won't need to add them every time for encryption and decryption. Public key will be used to sign user emails and is needed by others to send him encrypted emails. Private key will be used to read encrypted emails. Private key is confidential. It will be stored in local storage of users browser and never on server.


Upload PGP Keys

Please copy the keys in their respective inputs. These keys will be locally stored so you won't need to add them everytime you send or receive email.

Public Key

Public key will be used to sign your emails and is needed by others to send you encrypted emails.

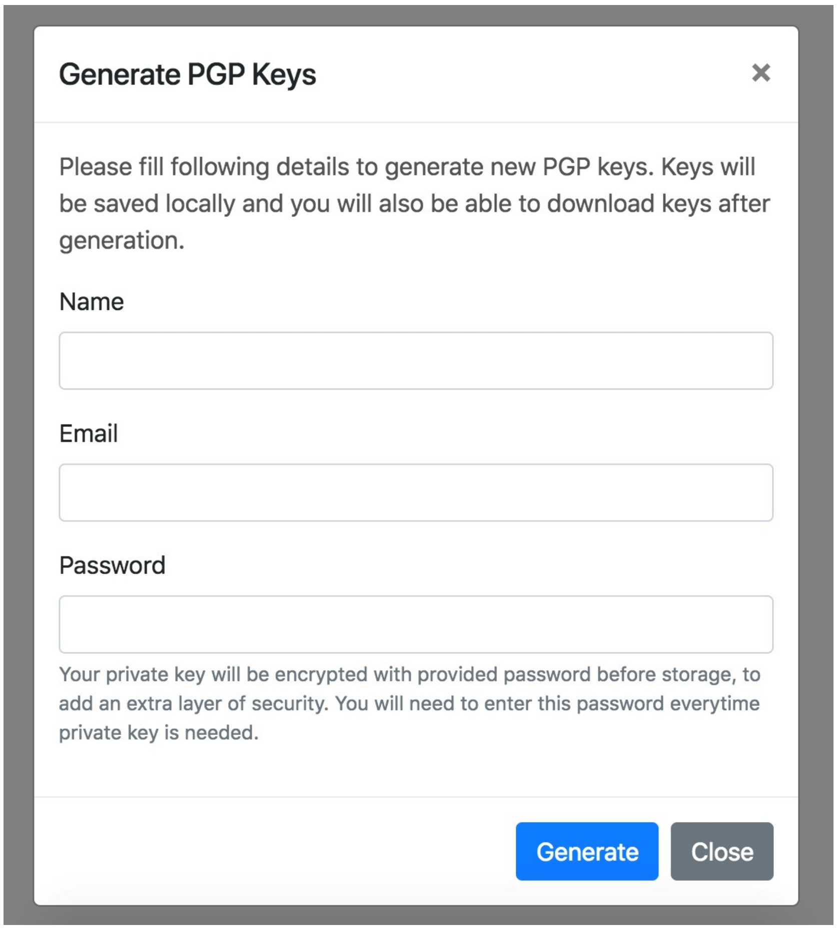
Private Key

Private key must be kept secure at all times and will only be needed to read encrypted emails. Do not share this key with anyone.

Figure 4.2: Uploading PGP keys

Generating Keys

User can generate new keys from application. User will be required to add name, email and password to generate keys. Private key will be encrypted with provided password before storage, to add an extra layer of security. User will need to enter his password every time private key is needed. Keys generated are standard PGP keys.


Generate PGP Keys

Please fill following details to generate new PGP keys. Keys will be saved locally and you will also be able to download keys after generation.

Name

Email

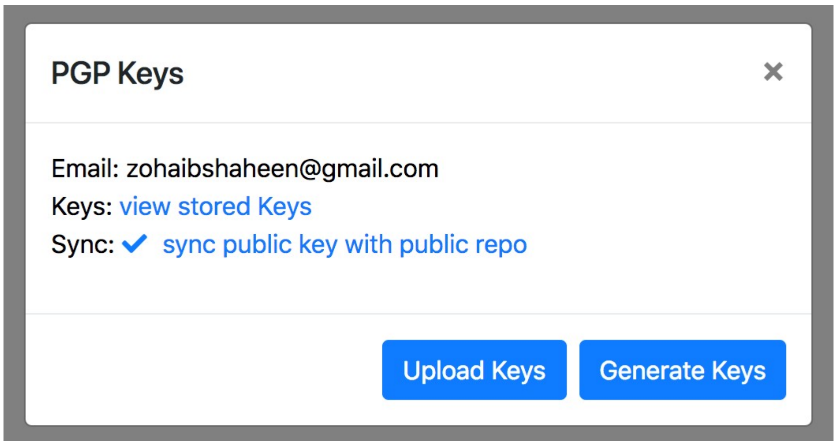
Password

Your private key will be encrypted with provided password before storage, to add an extra layer of security. You will need to enter this password everytime private key is needed.

Figure 4.3: Generating PGP keys

Key Sharing

Once keys have been generated or uploaded, they are stored in browsers local storage. But the purpose of public key is that everyone knows about that and can send particular user encrypted email using that users public key. Therefore, for easy sharing of keys, we have made a public directory for storage of emails and their public keys. So any user using our service can send encrypted email to other user on our application without the hassle of adding public key manually. Sharing our public key is as simple as pressing a single button.The user can click ’sync public key with public repo button and SecuriMail will automatically upload users email and public key in our directory. As simple as that!


PGP Keys

Email: zohaibshaheen@gmail.com

Keys: view stored Keys

Sync: ✓ sync public key with public repo

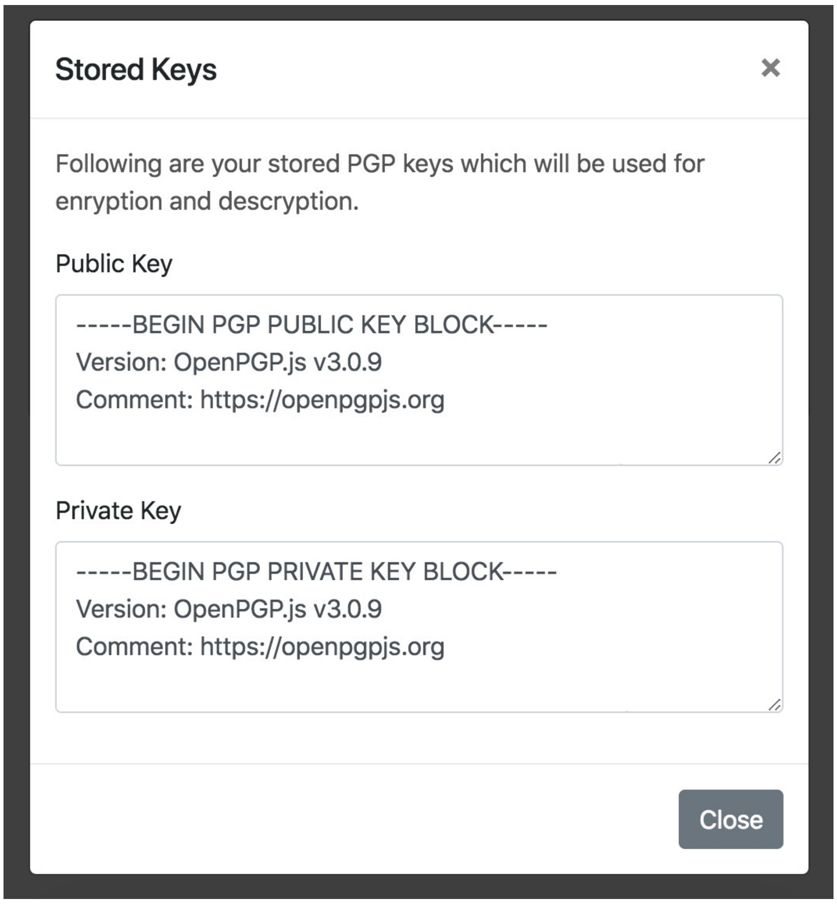
Figure 4.4: PGP keys

View Stored Keys

If user doesnt want to share his public key in our directory. He can access his stored public and private keys at any time. It is of importance to note that the private key shown here is encrypted with user selected password. Hence user can only copy encrypted version of his private key. This is to add an extra layer of security in case someone gets physical access of users browser.

4.1.3 Email Client

This section will include details about accessing inbox, sending encrypted emails and decrypting received emails. This is the main backbone of application. It is kept as simple as possible and as close to Gmails interface as possible. Emails are shown in list view. The list displays senders name, subject of email and time received. Once user clicks emails subject, detail


Stored Keys

Following are your stored PGP keys which will be used for enryption and descryption.

Public Key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: OpenPGP.js v3.0.9
Comment: https://openpgpjs.org
```

Private Key

```
-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: OpenPGP.js v3.0.9
Comment: https://openpgpjs.org
```

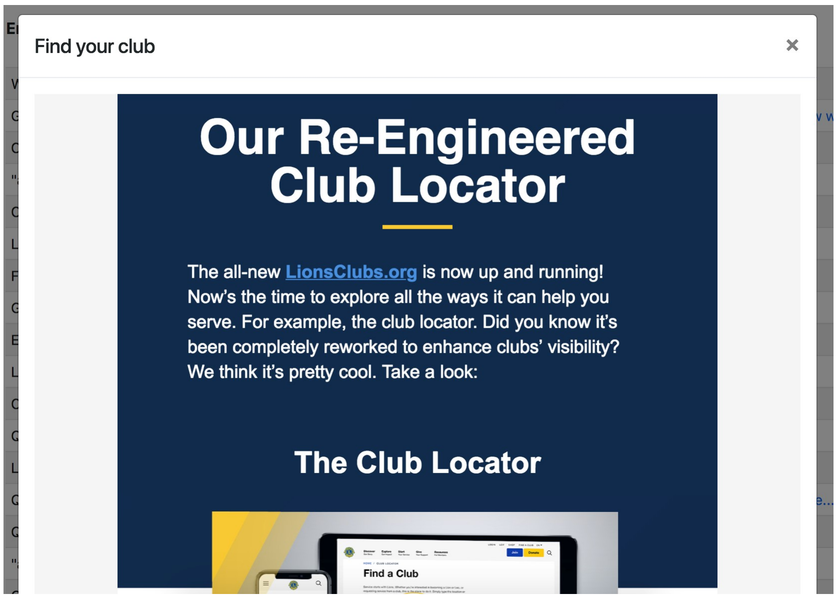
Figure 4.5: Stored keys

modal box is shown with full message body as shown in figure **??**. In case of encrypted message, the user will be prompted to decrypt it. In order to decrypt it, user will have to add his password with which his private key is encrypted. The private key is stored in browsers local storage. SecuriMail automatically uses this key and decrypts it with password provided by user. Then that decrypted key is used.

If encrypted email is accessed directly from Gmail, it will show meaningless data as shown below.

—-BEGIN PGP MESSAGE——
Version: OpenPGP.js v3.0.9
Comment:https://openpgpjs.org
wcBMA5XrGTpqGe40AQf/eSTXBsSDsrHXSV+pIWRYu0hVhlaE0/J4Bqnso4eq
AyRKIMd3SV+eszGUnIwFNexpdBrdansi+rdS5OQ/uFt/28ftCzuLh7AyGEh+
uSnvx9fVaepGnhIY3+tDJ85sMt8zgVv6KQG9eXduaLjIly2Ref7F79J9isTm
IRBpjRQJrdb5ar1z8bc2CE1baJ2XR/agkDtgVUfx5kHw1eA1KkAlmRVhQpgF
nu5eFve1TKKMFQndxKX/Od9U2fXDo1Gl1qHLEMjd26JaACFOPs6vE9htB0k+
lX/FdOLsZCsOYrHJKXd4t69At6MzHHspvzDRvU7ZFQfZssl147lcS36FBA0/
4NLB/wGylchBsLhWy3gufPMrlN4QOVX6RcvvRScE3RcUXaq5So8V/yb6H9c/
wSTHeELt+4T3/SdVeGH8/Kt6cd8Xf03PypxH6eTrKgQ92c2dJRCmiogQQZU7
wowkzjHDWzi7Pe+EJSqQj1wdwZUzYzpw6+Ox2z9EOMKtFcMBpxS+0ynr/Oip
9GRhF5AaB5QGdfzXsolWCesAVStUyx8NFJIaJ1VWXTiN41mIh61rLCI4e1zp
oSVlciHzwom14Repsap+8K68mMSUlZnVAfwDuZ3026/9YMr1HWYIQVX/+vg+
3eq5olKakf7wracjN137CaprzeGug4uWNFXmlIaqnuFryOyzId5M42M6uheO
gycMMLB9U0Zwjbtp5hBj/HHZLx7GnrxvV9qdktD/Bl9JetM282rK7vk6eGeA
mh8dyzPO49G7oAcTunhxxm+oryoCXEvNAqfhkqhXWrQY/5UFLsJ2qeCxi6W8
WXYVrmfotZoFFo7ZRiYUJg6NFjAt3gHDF0ZwieXElmf6ZQq/1G8DchdwBpak
J4BmfoTjOYf5yMzsQ9dyWQxXd2rCKPgkzHVHCTeOS9cJ/d2Dp5Rda5UPDVL5
ITav3oHLN+EY9ADj6XGNYxRknaug32ZX6fZUScHn///Am1AdQKIyZs4xh5Xs
S0883EYFt2wBP/1/mj5BeJi4xz++uGu25IkzUvLvvKGNbm5FKmQxMQOLyl7R
UMMPbJXdXumf3J3RS4evAW7aEJsl1xyAFXNfOkq4KKTeAhvmeMfvSFZ74W71
i074tC3GeS6jgaVdji2tVO4F6JYXs5YJUa9tTi1DIfgU/ZFPhpMjJSe4izwT
FwUPnB4aGiIPyjHiGBB9oArKeZd64tuezr0efa+upJqzWWw464l1gWY7/Fr/
IEs3BlEX50Veo7GUe8yCsRwSl/67CWloR1uvMh0rrTY=
=sblq
——END PGP MESSAGE——

Only SecuriMail can access information from that encrypted email. Once that email is opened in SecuriMail and it is decrypted. The application will even retain images and email formatting as shown in 4.6.



Find your club

Our Re-Engineered Club Locator

The all-new LionsClubs.org is now up and running! Now's the time to explore all the ways it can help you serve. For example, the club locator. Did you know it's been completely reworked to enhance clubs' visibility? We think it's pretty cool. Take a look:

The Club Locator

Figure 4.6: Fully Formatted Decrypted Email

4.2 Core Development

This section will explain the actual programming behind the application and it is divided into six algorithms for better understanding of readers.

4.2.1 function handleAuthClick():

Algorithm 4.1 is used to call gmail to start authentication of user. clientId and apiKey are taken from gmail for OAuthentication in our application. The scopes define which permissions are needed from users. In our case, we re-

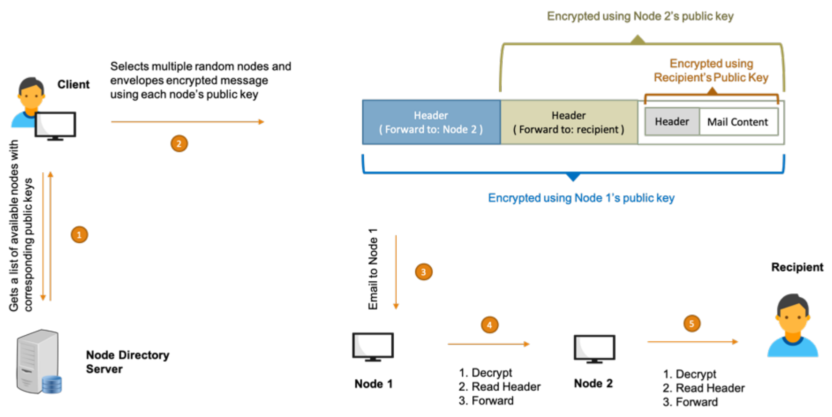

Encrypted using Node 2's public key
Encrypted using Recipient's Public Key
Client
Selects multiple random nodes and envelopes encrypted message using each node's public key
Header (Forward to: Node 2)
Header (Forward to: recipient)
Header
Mail Content
Encrypted using Node 1's public key
Gets a list of available nodes with corresponding public keys
Email to Node 1
Recipient
Node Directory Server
Node 1
1. Decrypt 2. Read Header 3. Forward
Node 2
1. Decrypt 2. Read Header 3. Forward

Figure 4.7: Email Forwarding Using Onion Routing

Algorithm 4.1 function handleAuthClick():

```
 1: Set clientId
 2: Set apiKey
 3: Define scopes
 4: Ask read permission
 5: Ask write permission
 6: Contact Gmail for authentication
 7: if authentication is successful then
 8:     Proceed to next algorithm
 9: else
10:     prompt error message
11:     Goto step 1
12: end if
```

quested permission for reading users email and sending email on his behalf on client side. Once gmail processes the request, handleAuthResult(authResult) function is called to send back result of authentication.

Algorithm 4.2 function handleAuthResult(authResult):

```
1: Check user authentication
2: Store user email
3: if public and private keys exist then
4:     Display Inbox
5: else
6:     Upload keys
7:     Goto Step 5
8: end if
```

4.2.2 function handleAuthResult(authResult):

Algorithm 4.2 checks if user is authenticated. If user is successfully logged in. His email is stored in local storage. Then the function checks if user has already created public and private keys. If these keys are not found in local storage, the user is prompted to generate or upload keys before accessing inbox. Once keys are stored, the algorithm will show inbox to user.

4.2.3 function openpgp.generateKey();

Algorithm 4.3 is used to generate public and private key pair for user using OpenPGPs javascript library. The user is required to enter passphrase and email. The email is validated with standard email format. Password is validated and only stronger passwords are allowed. Then algorithm generates public private key pair. It is of importance to keep in mind that the private key generated is encrypted and user is required to enter passphrase to decrypt his private key every time he wishes to decrypt his emails. Generated keys are stored in local storage of browser for convenience. These keys can be downloaded in text format for exporting.

Algorithm 4.3 function openpgp.generateKey();

```
1: Input email
2: Input passphrase/password
3: Validate email and password
4: if valid then
5:     Load OpenPGPs javascript library
6:     Generate public private key pair
7:     Encrypt private key with passphrase/password
8: else
9:     Goto Step 4
10: end if
11: Store Public Key
12: Store encrypted private key
```

Algorithm 4.4 function jQuery.getJSON()

```
1: Contact central directory
2: Browser email and public key pair
3: Retrieve email and public key pair
4: Store in variable
```

4.2.4 function jQuery.getJSON()

Algorithm 4.4 is used to get public key and email pair of all users from our central database. This helps user to automatically find public key of recipient. These public key and email pairs are also used to aid in creating onion route which will be discussed in details in encrypt function.

Algorithm 4.5 function encrypt();

```
1: Input Email
2: Input Subject
3: Input Message
4: Validate inputs
5: if valid then
6:     Get public key of email inputted in step 2 using algorithm 4.4
7:     Load OpenPGP javascript library
8:     Assign recipient value from step 2
9:     Encrypt message using public key taken in step 6
10:    while random do
11:        Get random email and public key from algorithm 4.4
12:        Add previous email details on top of message
13:        Encrypt message using public key taken in step 11
14:        Replace recipient with email taken from step 11
15:    end while
16:    Send email to new recipient
17: else
18:    Goto Step 1
19: end if
```

4.2.5 function encrypt();

Algorithm 4.5 is of utmost important in our thesis. It is used for encryption of email based on OpenPGP javascript library. When user wanted to send encrypted email, the application executes 4.5. It checks if all required fields are filled i.e. recipient, subject and message. Once it passes that check, the recipents public key is taken from central database and message is encrypted using that public key, so that only required recipient can read the email using his private key. Now the message is encrypted and ready to be sent away. But, we want to also hide sender / receiver identity. For this purpose, we take inspiration from Onion routing. The function grabs 3-5 other user email and key pairs and that encrypted message is further encrypted in layers using these public keys.

In depth Explanation

First our application will encrypt message that only actual Recipent can read and add extra information on top of it telling Email3 to transfer it to Recipent. Then that User3s message is further encrypted for User 2. User 2 can only read that he needs to transfer massage to User3. Similarly, the client will add another layer, which only User1 can decrypt and that user can only read that the message needs to be transferred to User2.

Encryption takes place on client side. Server cannot read any users message or onion route. The emails which are selected to be part of onion route which aid in hiding sender / recipient identity are chosen on client side and no record is kept anywhere.

Forwarding

When the message is transferred from Sender to Recipent, User1 decrypts its layer and finds out that it needs to transfer message to User2. Then User2 decrypts its layer and finds out that it needs to transfer it to User3. Then User3 decrypts its layer and finds address of Recipent and it transfers message to Recipent. Then the Recipient decrypts its message sent from actual Sender.

Algorithm 4.6 function decrypt();

```
1: Get encrypted private key from local storage
2: Prompt user to enter passphrase/password
3: Load OpenPGPs javascript library
4: Validate password
5: if valid then
6:     Decrypt encrypted private key using password
7: else
8:     Show error
9:     Goto step 2
10: end if
11: Validate private key with email message
12: if valid then
13:     Decrypt email
14:     if decrypted email have header details to further forward email then
15:         forward to recipient from decrypted email
16:         show thank you message for being part of onion route
17:     else
18:         show decrypted email in application
19:     end if
20: else
21:     Show error
22: end if
```

4.2.6 function decrypt();

Algorithm 4.6 is used in decrypting emails body. The message is decrypted using OpenPGPs javascript library on client side. The user is prompted to enter passphrase/password needed to decrypt the private key. This decrypted private key is then used to decrypt message. If this private key is corresponding to the public key which is used to encrypt message, the message is decrypted correctly. Otherwise the user is shown error. If decrypted email have header details to further forward email, forward it to recipient found after decrypting email. Show current user thank you message for being part of onion route. Otherwise, if decrypted email had no extra header details that means this email is intended for current user. Show current user the decrypted email in SecuriMail. Decryption takes place only on client side. Server cannot read any users message either encrypted or unencrypted.

Chapter 5

Evaluation

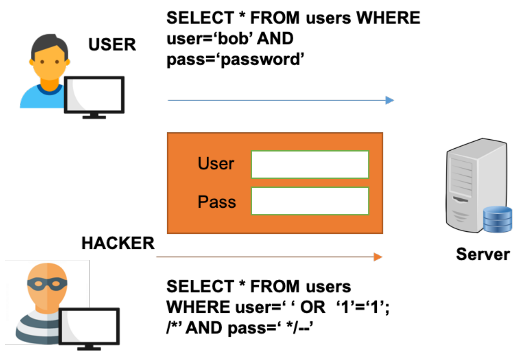
In this chapter, we first evaluate the security of SecuriMail. We describe possible attacks on web applications which is followed by detailing about how proposed SecuriMail is equipped with counter measures to prevent them. Then we give a brief discussion of a usability case study conducted on SecuriMail. In the last, we compare SecuriMail with it's well-known counterparts and describe how proposed solution is better than its counterparts.

5.1 Security Evaluation

This section evaluates the security provided by our application against advanced attacks. First, it explains the possible attacks and then it explains the counter measures taken by SecuriMail.

5.1.1 SQL Injection Attack

In this type of attack, the attacker tries to get access of database by injecting custom SQL queries. If successfully, this is the most dangerous type of attack. It has been depicted in Figure 5.1. In SecuriMail, it means the attacker can get access of email and public key directory and change corresponding emails and their public keys. However, even if successful the attacker cannot get private key of any user because the application stores private key only in end users local system and it is further encrypted by passphtase,making it totally safe. In order to prevent SQL injection, SecuriMail is equipped with web application firewall (WAF) which ensures that no malicious query can be passed to the server.



```
SELECT * FROM users WHERE
user=‘bob’ AND
pass=‘password’
```

```
SELECT * FROM users
WHERE user=‘ ‘ OR  ‘1’=‘1’;
/*’ AND pass=‘ */--’
```

Figure 5.1: Depicting SQL injection attack

5.1.2 Cross-Site Scripting (XSS) Attack

Cross site scripting happens when an attacker can inject a malicious script into applications front end, demonstrated in Figure 5.2. This often happens because of poor site security. There are two types of XSS attacks; persistent and reflected. In persistent attack, the attacker stores some malicious script and whenever that page is opened, that malicious script is executed. In reflected attack, the attack constructs URL of application in such a way that whenever a user visits that URL, the malicious script is executed. Once any form of attack is successfully, the attacker can steal cookies and session information data of users. In order to prevent this type of attack, SecuriMail

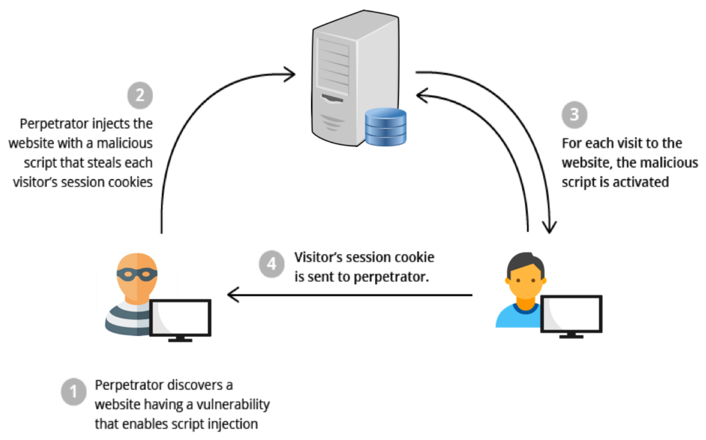

2
Perpetrator injects the website with a malicious script that steals each visitor's session cookies
3
For each visit to the website, the malicious script is activated
4
Visitor's session cookie is sent to perpetrator.
1
Perpetrator discovers a website having a vulnerability that enables script injection

Figure 5.2: Demonstrating possible XSS attack

is following guidelines of OWASP XSS security. If user tries to push tags and scripts through input, he is blocked. Every input is validated and sanitized before displaying its result on front end.

5.1.3 Brute Force Attack

In this type of attack, the attacker tries to log in on actual users behalf by using brute force. The attacker users some script which tries to input all possible combinations of username and password to pass through. In our case, SecuriMail doesnt allow login inside application. User needs to log into their Gmail account on gmail.com. Hence, in this regard Gmail is ensuring security of users accounts.

5.1.4 Eavesdropping Attack

In this attack, the attack eavesdrops on client which is communicating with server and tries to read their communication. This is demonstrated in figure 5.3. In our case, the attacker trying to read emails sent and received by user.

Fortunately, SecuriMail is built keeping in mind the privacy and security of end users. All emails are encrypted on client side using recipients public keys. Hence the attacker cannot access email messages. Similarly, the emails are decrypted at client side using private key stored only inside clients browser, which is further encrypted using passphrase. Hence communication is totally secure and safe!

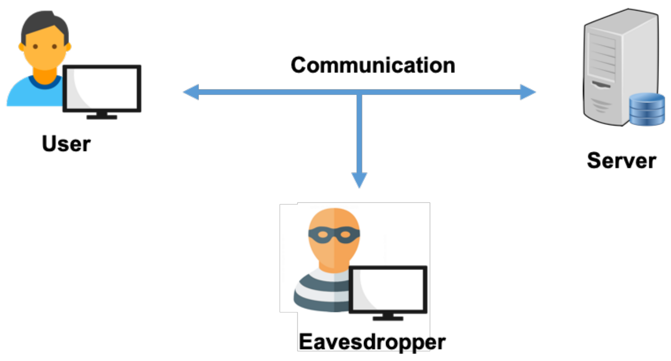


Figure 5.3: Eavesdropping Attack

5.2 Usability Case Study

This is a case study conducted to test usability of SecuriMail by general public. The study parameters are kept as close to a previous case study Why Johnny Still, Still Cant Encrypt [2]. That study was conducted to test usability of Mailvelop. Mailvelop is a browser extension that works on top of Gmail to send and receive encrypted emails. Surprisingly, in that study there was only 5 percent success rate. Therefore, we conducted this study on SecuriMail to compare results with previous study of Mailvelope.

5.2.1 Study Setup

The study was conducted on 12 May, 2019. A total of 10 participants were used. They were formed in pairs of 2. Each pair was given task to communicate successfully with each other. Each pair was given one hour to complete the task. In order to maintain standardization, both users in pair were required to have Gmail accounts. The participants were given instructions about what they have to perform.

Table 5.1: Educational Backgrounds Of Participants

	Participant 1	Participant 2
Pair 1	Under Graduation Student	Under Graduation Student
Pair 2	Under Graduation Student	Under Graduation Student
Pair 3	Under Graduation Student	Under Graduation Student
Pair 4	Post Graduation Student	Post Graduation Student
Pair 5	Post Graduation Student	Post Graduation Student

5.2.2 Demographics

All participants were invited from NUST university. There were 8 male participants and 2 female participants. Educational backgrounds of participants are shown in table 5.1.

5.2.3 Results

At the end of study, it was found that 4 pairs were successful in communicating with each other. This means 80 percent success rate as compared to 5 percent success rate of Mailvelope. The successful pairs found it simple and straight forward to use. The only pair which was not successful, found key creation to be difficult. They further explained, they were trying to send email to recipient without creating account of recipient on SecuriMail which means the public key of recipient was not available and sender couldnt locate recipients account beforehand. Time taken and results of study according to each pair is shown in table 5.2.

Table 5.2: Study Results

	Time Taken	Success
Pair 1	20 mins	Yes
Pair 2	40 mins	No
Pair 3	13 mins	Yes
Pair 4	12 mins	Yes
Pair 5	10 mins	Yes

5.3 SecuriMail Vs Other E2E Applications

All E2E applications including ProtonMail, Confidante, Mailvelope, to mention a few, do encryption; so does our SecuriMail. But the question here arises is then how it is better than all well-known E2E applications? Fortunately, the answer is interesting. It is worth mentioning that SecuriMail is an E2E application, first of its kind, which does use Onion routing to hide the identities of sender/ receiver pair. Also, it outperforms other E2E applications in other features are well.

All other E2E applications, as depicted in Table 5.3, merely provide a couple of features which limits their use. For instance, though ProtonMail provides built-in key directory, however, it is not open source. Moreover, Mailvelope which is open source though, does not provide other features. Therefore, our SecuriMail thus provides many features which are demonstrated in the Table 5.3.

Table 5.3: A Brief Comparison of E2E Applications Against Different Characteristics

E2E Applications	**Server Less Model**	**Open Source**	**Mobile Compatibility**	**Random Header Size**	**Reciever Identity Hiding**	**Sender Identity Hiding**	**Onion Routing**	**Built-in Public Key Directory**
Mutt Email Client	No	Yes	No	No	No	No	No	No
ProtonMail	No	No	Yes	No	No	No	No	Yes
Mailvelope	Yes	Yes	No	No	No	No	No	No
Confidante	No	Yes	Yes	No	No	No	No	No
SecureiMail	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

5.4 Compromised Parameters

Our designed system compromises on the following aspects:

- As in our system, an Email needs to be hopped from one user to the other before it reaches the receiver. Thus, time delay in the system can be incurred. However, this is how onion routeing works. We do incur time delay but sender and receiver identity is masked.

- Message size is also increased in our case. Size of the message depends how many hops Email will traverse in the network; because every hop adds a layer on message received. This could increase size of the message significantly when number of hops is large. We argue that in future, this direction needs to be explored to minimize overhead.

Chapter 6

Conclusion and future work

The research shows that it is not only possible to improve usability of existing solutions, with same level of security, but to also ensure privacy of the user identities without compromising on either performance or adaptability.

6.1 Usability

In a test study conducted with small number of test users, applications interface was reported to be intuitive and mostly self-explanatory. Breaking down of key generation into steps and providing tips at each step to make it simpler, was specially appreciated by all users. Test users didnt experience any crashes or unexpected behaviors throughout the study. It was however necessary to brief on what is OpenPGP in essence and how we are employing it on client side, in order to establish user trust.

6.2 Security

Our application employs NIST recommended key length of 2048 bits, which has been deemed sufficient for security requirements till 2030. Our choice of key length provides equal and in few cases better security than other available solutions in market today without impacting performance.

6.3 Adaptability

Adaptability is usually a decisive factor in mass adoption of any application. The goal was to make it feasible for application to be adapted for various platforms like Android, iOS and Linux. With the above goal in mind, the application was built entirely in client-side JavaScript. This not only allows access on any platform through browser but also makes it extremely easy to create a port for any platform in the form of native application. Unlike other solutions available in market, we have built this tool on top of largest email provider in the world i.e. Gmail which has more than 1.2 billion monthly active users. This incentivizes users by allowing them to send secure mails without changing their email account.

6.4 Performance

OpenPGP is unique in a way that it combines best features of both public key cryptography and symmetric encryption. It works by compressing data and then encrypts it using one- time symmetric key which is in-turn encrypted using public-key cryptography and sent with encrypted email. This greatly impacts which key length we can use for our public / private key pair. Our choice of 2048 bit key was based on the fact that key length will only affect

the key
generation, which is one-time process and wont affect encryption or decryption of email at any stage thus providing same performance as other solutions in market.

6.5 Contributions

This research has been carried out in three systematically chosen steps.

- We have analyzed existing solutions and found usability and privacy issues associated with each. Based on these issues, functional and non-functional requirements for new system have been laid down.
- We have proposed a new system using available technologies, which can improve on usability of existing solutions and add privacy by using Tor network.
- We have created a fully functionating cross-platform application, which can be used by anyone with Gmail account.

The summary of our contributions is as below:

- We have done an in-depth analysis of existing solutions in market and found usability and privacy concerns in each. In studies highlighted in previous sections, it has been shown that usability of existing applications has been a hinderance in mass adaptation of openPGP by users. This ranges from un-intuitive interfaces to lack of key management and user guides built into applications today. We have also highlighted that none of the solutions available provide privacy or user identity hiding, which is a crucial feature now giving privacy concerns of users all across the world. Based on the shortcomings of existing solutions, requirements for new system has been laid down.

- Our second contribution is in the form of new architecture which provides user identity hiding by employing Tor in email routing. This is a novel approach towards email privacy and has been proved effective in providing anonymity and privacy in other forms of data communication before. Apart from identity hiding, architecture also proposes an improved usability model by integrating key management. Architecture relies on Zero Trust model i.e. there is no server involvement and all steps including key generation, encryption, decryption and routing are done on client-side. The architecture can be easily adapted for other platforms including Android, iOS and Linux and is not limited to Gmail only.

- Our most important contribution is in the form of a cross-platform application built using client-side JavaScript. It provides an intuitive and simple interface to connect Gmail and encrypt / decrypt emails on the go. Application has built-in key management and public repository as proposed in the architecture and handles email routing using Tor with minimal user interaction.

- The biggest accomplishment of application is accessibility on any device using browser and its ability to handle key generation, encryption, decryption and routing on client-side with zero sever involvement i.e. Zero Trust model. This zero-trust model is crucial nowadays given that large scale data breaches have been reported by multinational organizations and our data cannot be considered secure with any server now.

6.6 Future Research Directions:

Like any research, this research has certain limitations and based on these limitations we are prosing future directions to improve usability and performance of system.

- Application can already be accessed on any platform using browser but adaption of the solution in the form of native applications for platforms like Android and iOS will greatly improve its adoption and usability. Features like push notifications and background sync can push users to use these apps as their daily drivers.
- Current implementation of Tor doesnt take into account if the intermediary nodes are online or not, which can add delays to email routing. This can be countered by using web sockets to broadcast status of each node to server, which can then publish online nodes only when applications requests node list for routing.
- Architecture is not limited to just Gmail and can be adapted for other email providers like Outlook or Yahoo, which provide APIs. However, it is absolutely necessary that APIs are accessible on client-side to main Zero trust model proposed in the research.

Bibliography

[1] A. Whitten and J. Tygar, “Why johnny can’t encrypt: A usability case study of pgp 5.0,” 1999.

[2] S. Ruoti, J. Andersen, D. Zappala, and K. Seamons, “Why johnny still, still can’t encrypt: Evaluating the usability of a modern pgp client,” 10 2015.

[3] A. A. Lerner, E. Zeng, and F. Roesner, “Confidante: Usable encrypted email a case study with lawyers and journalists,” 2017.

[4] D. Goldschlag, M. Reed, and P. Syverson, “Onion routing for anonymous and private internet connections,” 1999.

[5] S. L. Garfinkel and D. Margrave, “How to make secure email easier to use,” 2005.

[6] D. Salama, H. Abd elkader, and M. M. Hadhoud, “Performance evaluation of symmetric encryption algorithms,” *Communications of the IBIMA*, vol. 10, 01 2009.

[7] P. Henry and Hui Luo, “Off-the-record email system,” vol. 2, pp. 869–877 vol.2, April 2001.

[8] M. Agrawal and P. Mishra, "A comparative survey on symmetric key encryption techniques," 2012.

[9] D. P. Mahajan and A. Sachdeva, "A study of encryption algorithms aes, des and rsa for security," *Global Journal of Computer Science and Technology*, 2013.

[10] Xin Zhou and Xiaofei Tang, "Research and implementation of rsa algorithm for encryption and decryption," vol. 2, pp. 1118–1121, 2011.

[11] W. Dai, T. P. Parker, H. Jin, and S. Xu, "Enhancing data trustworthiness via assured digital signing," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 6, pp. 838–851, 2012.

[12] S. Halevi and H. Krawczyk, "Strengthening digital signatures via randomized hashing," 2006.

[13] F. Zibran and Minhaz, "Cryptographic security for emails: A focus on s/mime," 05 2019.

[14] B. Ramsdell and S. Turner, "Secure/multipurpose internet mail extensions (s/mime) version 3.2 message specification," 2010.

[15] D. D. Clark and M. S. Blumenthal, "The end-to-end argument and application design: The role of trust."

[16] W. West and S. M. Pulimood, "Analysis of privacy and security in html5 web storage," *Journal of Computing Sciences in Colleges*, 2013.

[17] S. Matsumoto and K. Sakurai, "Acquisition of evidence of web storage in html5 web browsers from memory image," pp. 148–155, 2014.

[18] S. Z. Naseem and F. Majeed, “Extending html5 local storage to save more data; efficiently and in more structured way,” pp. 337–340, 2013.

[19] J. Re and J. Wu, “Survey on anonymous communications in computer networks,” *Computer Communications*, pp. 420–431, 2013.

[20] M. Backes, I. Goldberg, A. Kate, and E. Mohammadi, “Provably secure and practical onion routing,” pp. 369–385, 2012.