LILAC: THE SECOND GENERATION LIGHTWEIGHT LOWLATENCY ANONYMOUS CHAT.

by

POBALA REVANTH RAO

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

JULY 2016

To my mother Neeraja, my father Late Jagan Mohan Rao and my brother Yashwanth Pobala who stood with me during difficulties and encouraged me to achieve this degree.

ACKNOWLEDGEMENTS

I express my sincere gratitude to my supervising professor, Dr.Matthew Wright who has been a great motivating factor and a constant source of encouragement throughout my masters' research. Without his guidance and excellent foresight this thesis would have only remained a great idea. I am sincerely thankful to Dr. Ramez Elmasri and Prof. David Levine for giving valuable suggestions and serving on my committee.

I would like to thank the administrative staff, specifically Ms.Pamela McBride and Ms. Sherri Gotcher for their valuable support and services. Special thanks to Ms.Camille Costabile in helping me through the final requirements of my thesis, and the department of computer science, University of Texas Arlington. I am grateful to Hussain Ali Mucklai for helping me in this project.

My heartfelt thanks to my family for continuous support and inspiration. I would like to express my appreciation to Jong Park, Mohsen Imani, Jees Augustine, Armon Barton, Shantan, Ashwin, Aastha, Manu Bhat and other friends in Information security lab . I also greatly appreciate my numerous other friends for their love and support.

July 26, 2016

ABSTRACT

LILAC: THE SECOND GENERATION LIGHTWEIGHT LOWLATENCY ANONYMOUS CHAT.

Pobala Revanth Rao, MS.

The University of Texas at Arlington, 2016

Supervising Professor: Matthew Wright

Instant messaging is one of the most used modes of communication and there are many instant messaging systems available online. Studies from the Electronic Frontier Foundation [1] show that there are only a few instant messengers that keep your messages safe by providing security and limited anonymity. Lilac, a **LI**ghtweight **L**ow-latency **A**nonymous **C**hat, is a secure instant messenger that provides security as well as better anonymity to users as compared to other messengers. It is a browser-based instant messaging system that uses Tor[2] like model to protect user anonymity. Compared to existing messengers, LILAC protects the users from traffic analysis by implementing cover traffic [3]. It is built on OTR (Off the Record) messaging to provide forward secrecy [4] and implements Socialist Millionaire Protocol [5] to guarantee the user authenticity. Unlike other existing instant messaging systems, it uses pseudonyms to protect user anonymity. Being a browser-based web application, it does not require any installation and it leaves no footprints to trace. It provides user to save contact details in a secure way, by an option to download the contacts in an encrypted file. This encrypted file can be used to restore the contacts later. In our experimentation with Lilac, we found the Round Trip Time (RTT) for a message is around

3.5 seconds which is great for a messenger that provides security and anonymity. Lilac is readily deployable on different and multiple servers. In this document, we provide in-depth details about the design, development, and results of LILAC.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Instant messaging involves sending messages, generally short in nature, in real-time to one or more intended recipients. Instant messaging is used for any number of purposes. There are cases where sensitive information needs to be quickly transmitted back and forth between parties, for example, communications between dispersed groups in a military setting, consultations between a patient and her doctor, or, more generally, friends and family conversations in civilian locales. In the later cases, where ease of use and speed of communication trump the security and privacy concerns of the users, instant messaging systems are used that endanger the anonymity of its users. Encrypting data on its own does not provide the privacy for users. The main purpose of encrypting the data is to protect the confidentiality of transmitted data but it leaves the identities of the communicating parties unprotected. In simple words, the third party application knows that who is talking to whom. Techniques such as traffic analysis[6] can uncover sensitive information about the participants in a distributed application. Concealing the sender and receiver's identities is an important aspect in maintaining the privacy, confidentiality, and integrity of their communications. Leaking meta-data such as the addressing, timing, and volume of traffic can, in some cases, lead to an adversary learning as much information as she would have with the content of the communication. Despite this, there are very few deployed systems that are committed to providing strong anonymity to its users in real-time chat communications.

To provide for these users' needs, we present Lilac, a lightweight, low latency, anonymous chat system. The primary objective for our design of Lilac is to leverage multi-layer

encrypted circuits to prevent an eavesdropper from being able to link communication partners. We further our defense by adding cover traffic onto the network, which mitigates the effects of timing analysis attacks in identifying chat partners.

The design of Lilac also takes into account the deployability and usability of the system. We attempt to make Lilac user-friendly and easy to use by hiding the complexity away from the user's eye. We have deployed the system at www.thelilacproject.org for public use. We have made the interface comparable to one a modern day user would expect of a chat system, as well as deploying the system as a web application that would not require the user to download additional software and install it on their system. That said Lilac doesn't leave any footprints because it does not store users messages, in fact, Lilac does not have a database of the chat history nor the users. The only information Lilac keeps is the information related to the servers in the system. In this way, we provide the ease of access and ease of use to the users that they have become accustomed to. Our main concern on this front was to design a system that when implemented, would provide the performance similar to the popular chat systems, despite the additional overhead of the security and anonymity features ingrained into our design. Our experimentation design and results are discussed in chapter 3.

CHAPTER 2

BACKGROUND

In this chapter we provide an overview of three areas of previous work [2] [7] [8] which helped shape our decisions in the design of Lilac. Chat privacy and anonymity are main benefits that Lilac aims to provide. The privacy and anonymity aspects related to a chat system such as Lilac are provided in the following subsections. First of all we will talk about the existing anonymous systems and then we will discuss the earlier design of Lilac. There have been various different attempts and approaches to the design and implementation of anonymous communication systems. The first such system was provided by Chaum [7], in which anonymity was provided by relaying messages through a series of nodes referred as *Mix Nodes*. The term refers to a server which accepts incoming connections from multiple clients and forwards them to their appropriate destinations in such a way that an eavesdropper is not able to ascertain relationships between in-bound and outbound connections. Due to the fact that a single mix node may be compromised, messages are relayed through multiple nodes to increase the strength of the anonymity.

Anonymous communication systems can be distinctly classified into two categories based on the high-latency and low-latency of communication provided. High latency architectures are designed for use in communications which does not require immediate feedback or response, such as an email. Low latency systems, however require are designed for applications which do require real-time responses, such as instant messaging. Onion Routing (OR) [2] is a technique which attempts to provide a practical solution to the implementation of low latency anonymity systems. It is designed to defend against attacks such as eavesdropping and traffic analysis. OR prevents the transport layer from determining the identities of

two communication parties. All that can be discovered is whether or not communication is taking place at one end. Now, we will discuss briefly about the chat privacy, chat anonymity, onion routing and some instant messaging systems that provide anonymity and security in detail.

2.1 Chat Privacy

Securing the privacy of user's conversations is one of the most important requirements of any anonymous communication system. The fact that an individual is using such a system hints that the contents of their conversations are sensitive in nature, and must therefore be protected. Without this protection, not only the contents of the conversation may be exposed, but the identities of the communicating parties as well. Furthermore, the addressing, timing and volume of traffic can in some cases leak as much information as its content [9].

The design of our chat system must therefor follow the lines of Chaum's mix node model [7], while adding to it a technique for obscuring the timing and volume of data flow but still allowing for real-time communication. The presence of the relays in the design makes our system openly susceptible to the Man in the Middle (MITM) attacks [5]. This could happen when the exit relay intercepts all the traffic from both communicating parties, and it replaces with the relay's own responses. The relay would be therefore be playing the role of both sides and would see the communication at both ends. To counteract this attack an added security protocol can be added called the Socialist Millionaire's Protocol (SMP) [10]. This protocol is used when there exist two parties who both have some secret value and want to determine whether their values are equal without disclosing any information about their value. In this case, the two secret values are the AES keys held by the communicating parties, concatenated with a secret pass phrase they share. The result would be different if a MITM attack had been successfully executed, but identical otherwise. This protocol will be discussed more in the chapter 3.

2.2 Chat Anonymity

There are three types of anonymity that can be provided by an anonymous communication system: sender anonymity, receiver anonymity, and unlinkability of sender and receiver. Sender anonymity refers to the protection of the identity of the user sending the data, and receiver anonymity refers to the protection of the identity of the user receiving the data. Unlinkability of sender and receiver is a property achieved when a user sending information cannot be paired with the user receiving the information, despite the fact that the sender and receiver are known to be actively communicating. An anonymous communication system must aim to provide all three of these types of anonymity.

As stated previously, anonymous communication systems can be classified into high latency and low latency systems. In both types of systems, unlinkability is achieved by routing and encrypting traffic through a series of nodes, based on the idea proposed by Chaum [7]. Intermediate nodes know only their predecessor and successor, and so long as more than one node is used, no node on its own can determine both the origin and destination of traffic passing through it. High latency systems provide measurable levels of anonymity using a one message per path model, whereby a new path is created for each message to be sent. In earlier design [11] there are only 2 relays between the client and the server. In our design we have upgraded it to three relays to provide more anonymity and security.

Systems such as Mixmaster [8] and Mixminion [12] provide re-mailer anonymity for sending and receiving e-mail messages. E-mail messages are sent to re mailer servers in fixed-sized encrypted packets, with instructions on where to send the packet to next. This architecture has proven to be an efficient one for use where delays in data transmission are not critical to the application's use, but it does not suit the needs of real-time systems such as online chat.Low latency systems, however, create connection-based paths and send data along these paths as a stream of packets for as long as the path is in use. Creating anonymous connections using onion routing [2] is the leading architecture for low latency anonymous

communication. Tor [2] is the second generation onion router. Onion routing concentrates on TCP-based real-time applications. In the Tor architecture, clients select a path through a secure tunnel and build a circuit. In the circuit, each node knows only the node immediately before it and after it. Each node shares a symmetric key with the client, which it uses to decrypt (or unpeel) the data it receives and forward it accordingly.The architecture also implement's a variant of Chaum's mixes [7] where mixes are derived by grouping a sequential set of nodes which accept messages from multiple senders, shuffle these messages and send them out in a random order to a subsequent set of mix nodes, or to the circuit's exit node. Such systems have been implemented previously, such as ISDN-mixes [13], which attempt to anonymize phone conversation traffic, and the Java Anon Proxy (JAP) [14], which attempts the same feat for web traffic. These systems have been primarily designed to provide real-time anonymous communication. However, they work in a synchronous manner which is not well suited for today's bidirectional asynchronous TCP/IP networks [15]. We focus on designing a low latency system to overcome this drawback while still enabling anonymous and private real-time chat.

2.3 Onion Routing

The Onion Router architectural approach has proven to be the most popular in regards to the design of anonymous communication systems. More specifically, the Tor Browser, an open source Internet browser used to connect to the Tor network [2], is a widely used open source application, catering to a thousands of clients per day across the globe [16]. The design of our system is based on Tor's approach, and this section will provide an overview of the Tor design.

Tor was designed to be a communication system focused on anonymity, but also low enough latency to be suitable for the interactive online applications used by most people today. The architecture of Tor consists of relays, directory servers and clients. Relays are

volunteer-operated servers which relay packets within the Tor network and eventually outside of it. Directory servers are a set of trusted servers which maintain information about the relays, such as their IP addresses, public keys, self-reported bandwidth capacities and exit policies. The clients are the users of Tor. Clients use Tor to build a circuit of relays to tunnel their data through before reaching its destination. To do this, clients must first request a signed list of relays from one of the directory servers. The client then selects some number of relays (3 by default) to create a circuit. Clients then establish unique symmetric keys with each relay using the Diffie Hellman algorithm in a telescoping key agreement procedure. The client can now create 512 byte chunks of data, called cells, using this circuit by encrypting the data with each of the symmetric keys and passing the cell to the first relay, called the guard relay. This relay uses the circuit ID attached to the cell to identify the correct key to use for decryption and the corresponding relay to forward the cell to. This process continues until the cell reaches the last relay, called the exit relay. The exit relay will decrypt the cell for the last time to discover the destination IP address of the data, and passes on the data accordingly. In this model, only the guard relay knows the identity of the sender and the exit relay knows the identity of the receiver.

The Tor design, therefore, offers an elegant solution to the problem of communicating anonymously over the Internet. As Tor has become more popular, however, its users have experienced frequent and lengthy delays [17]. Tor has expanded to cater to more user needs and this has added complexity and latency to its design. We propose a scalable anonymous system designed solely for text-based communication between users, with minimized latencies to provide users an experience similar to that which they might receive while using mainstream chat applications which provide no anonymity features.

2.4 DLP and RO-DLP

To resist timing analysis Low latency anonymous networks implement padding. One way to reverse such attack is to use Dummy traffic in a way that the overhead should be limited. Some of the well known padding algorithms are DLP [18] and RO-DLP. DLP stands for Dependent Link Padding.The output pattern of DLP is determined online depending on the input. DLP has a flexible bandwidth utilization. By using DLP the dummy traffic is minimized and the sending rate is proportional to *log(m)*. RO-DLP stands for Reduced padding overhead in DLP. Like DLP, RO-DLP schedules a cell each for outgoing circuits. RO-DLP removes the cells which are not required thus reducing the overhead. In Lilac the heart beats are generated by the following method.

- Generate a Random string R.
- Generate a Random encryption key K.
- Once you have R and K. Use AES Encryption (E) to encrypt R with K.

  Heartbeat message H = $E(R, K)$.

2.5 Instant Messengers available online

In this section we will discuss about some of the instant messengers that are available in the market and compare them with Lilac.

2.5.1 Tor Chat

Tor chat is one of the anonymous instant messaging system that uses Tor hidden services [2] as its underlying network. The main advantages of using Tor chat over Lilac is that it provides a support for file transfer which is lacking in Lilac. The disadvantages of Tor chat is that it is still vulnerable to Traffic analysis. Moreover, the Tor network is congested because of the rise in number of users and the increased usage of the network by different applications.

2.5.2 Cryptocat

Cryptocat [19] is a browser based instant messaging system, that provides security to the users. Cryptocat implements OTR and SMP like Lilac but it does not provide anonymity to the users.Lilac on the other hand provides anonymity as well as security to the users. Files can be transferred using cryptocat but this feature is not available in Lilac as of now. In future we plan to implement file transfer using DP5[20] protocol.

CHAPTER 3

LILAC'S ARCHITECTURE

In this section, we present Lilac: Lightweight low-latency anonymous chat. Our primary aim is to design and implement a system which is simultaneously secure, anonymous, easy to use, and deployable. There are four different types of components which make up Lilac. The first is the directory server, which monitors changes in the network topology, and maintains information about active relays and presence servers in the network. The directory server also hosts the Lilac website, and so delivers content to the clients. The second is the Presence Server, which maintains information about online users and is the intermediary responsible for connecting two users who wish to communicate. The third are the relays which provide anonymity to the system as per the Onion Routing model [2]. And finally, the fourth element are the clients, who are the end users of the system. We will use the rest of the section to delve into the details of the design of our system and the approaches used to reach our goals.

3.1 The Directory Server

The Directory Server is a trusted server which manages the information required for Lilac to operate as a system. That is, the Directory Server maintains information about the Lilac relays and presence servers. When one of these servers come online, they must first register themselves with the directory server and provide their IP address, public key and function (relay or presence server). The directory server is then able to compile the collection of online relays and presence servers, and update this collection as servers come online and go online. The architecture of Lilac allows for a single presence server, and so

the Directory server operates on a first come first serv basis, with the first presence server to register itself being Presence Server for the system and any following servers which register become backups in case the primary server goes down. In the case where no presence servers are available in the system, the Directory Server itself can undertake the role.

The Directory Server also performs the function of serving content to Lilac's clients. A user wishing to use Lilac will simply enter the domain name of the Directory Server into a modern web browser. The Directory Server will then serve pages which make up the Lilac user interface (UI) and the client-side program logic, as well as the details of the presence server and set of relays which are known to be online.

The implementation of Lilac as a web application; requiring no additional software requirements on the client's part (other than a modern web browser) is a step in the direction towards user-friendliness and usability. We believe this design decision will make the average user more receptive to using our system, as opposed to requiring them to download and install additional, unknown software to their personal computer. If the users do not wish to visit the website *www.thelilacproject.org* they can run the directory server on their own browser *localhost*. Lilac provides a lot of customization under the hood. Users can serve the web pages from their preferred framework (Flask, Django, Spring ..). The users can even have their own self-signed SSL certificates. The only thing they need to do is that to download the creds.json file from the website. The relay list consists of the information of the location of relays and their public keys. The structure of the relay list is as follows.

1. host = IP address of the Relay.
2. port = Port of the relay. default is 8081.
3. public key = Public key of the relay.

A sample of the relay list is shown below.


```
[{"host":"52.62.193.227",
"port":8091,
```

"publicKey":"5113daa74e909bef24e93f2b020b06f85e2ae46c3a34c54f9cfdceb77c983393"},
{"host":"54.169.78.216",
"port":8091,
"publicKey":"37bd39bce2fdb145e8e185be4774a0ebaeae376f3506b49df7b9ba216449be0a"}]

3.2 Relays & Creating a circuit

Relays are volunteer-based servers that perform the same functionality in Lilac as the relays do in Tor[2], anonymizing traffic. Relays in Lilac, unlike Tor, exclusively handle textual traffic related to chat. Because this traffic is lightweight, we do not need to be overly concerned with the bandwidth of our relays. As such, we may choose three relays (from the list provided by the Directory Server) completely randomly for our circuit. This eliminates any bias which malicious relays might attempt to exploit to get more traffic coming their way. In future, we might implement path selection algorithms in selecting the relays from the directory server like in Tor [21] [22]. Once the relays have been selected by the client, a circuit needs to be created and symmetric keys need to be established with each relay. The first step is to create a symmetric key with the guard (first) relay. Lilac implements ACE [23] protocol. which is an adaptation of the Diffie-Hellman key exchange algorithm and provides one-way key authentication, as well as operational efficiency compared with the other proposed protocols. Upon establishing a symmetric key with the guard relay, the client can use this key to privately communicate the IP address of the next relay in the circuit. The client can then establish a symmetric key with the second relay using the ACE Protocol, via the guard relay. The same process is repeated to extend the circuit to the exit (third) relay and generate a symmetric key with it and then repeated one last time for the presence server. This process is the first action taken at the client side. During this circuit build time, a loading screen is displayed on the client's web browser. Once the circuit has

been established and extended to the presence server, the client is able to use the system and is prompted to enter her username.

3.3 The Lilac Presence Server or Presence Server

This subsection will go into the details of the functionality and privacy issues related to the Presence Server. It is important to remember that the client shares a symmetric key with the Presence Server, and all communication between the client and Presence Server passes through the established circuit; therefore all traffic is private and anonymized. This subsection assumes this state to be true and focuses on the specific workings of the Presence Server and the issues related to it.

3.3.1 Functionality

The Presence Server serves a central role in establishing connections between chat partners. To do this, the Presence Server performs two functions: maintaining a list of online users and responding to requests to connect users. When a user accesses the Lilac web application, she is asked to provide a user-name to identify herself. This user-name is then used to mark the user as online for a specific period of time (more on this later). The Presence Server is then sent the user's identifying name, which it must store for future use. To begin communicating, the user must submit a request to start a chat by providing the user-name of her intended chat partner. The Presence Server receives this identifying name with the identifying name of the requester. The Presence Server then searches for the presence of the recipient in its records. If found, it forwards the request (with the requester's identifying name) to the intended recipient. To protect the recipient's privacy, no feedback is sent to the requester at this point. So if she is not able to connect to the recipient, she is not able to determine if the recipient is online, or is online and has ignored her request.

3.3.2 Epochs and Time-Stamps

The Presence Server is a critical point in the design of Lilac. This server receives information about users who are online on the system, and handles queries connecting users to each other. The Presence Server must be considered an adversary from the client's perspective and our design must, therefore, divulge as little information as possible to this server. The server must keep track of online users, and handle connection requests from users. Our main concern is the privacy and anonymity of Lilac users. When a user registers herself on the presence server to mark herself as online, she does so with her unique user-name. Allowing the Presence Server to see the user-names of all users in plain text is not an acceptable standard. Even if users hashed their user-name before registering with the presence server, the hashed user-name would remain a constant, and the presence server could still monitor the activities of a particular hashed user-name over a period of time.

To avoid this threat, the Lilac system operates on time periods known as *epochs*. The epoch would be the minimum length of time for which a user would be online and able to receive chat requests. The epoch length selected for Lilac is fifteen minutes. Another term used in Lilac is the *time-stamp* the number of epochs which have elapsed since the 1st of January 1970. Now, when a user wants to register herself as online on the Presence Server, she uses the hash of her user-name concatenated with the time-stamp to do so. When a user wants to request to start a conversation with her chat partner, she uses the hash of her chat partner's user-name concatenated with the current timestamp.

However, if a user registers herself near the end of the epoch, she may only be registered as online for an insignificant amount of time. Therefore, in the Lilac implementation, when a user registers herself on the presence server, she sends two identifying names, one for the current epoch and one for the next. The Presence Server receives both identifying names and maintains two collections of names: one for the current epoch and one for the next. When querying for a chat connection, the presence server only ever searches in the collection for

the current epoch. When the epoch elapses, the collection for the next epoch becomes the collection for the current epoch, and the collection for the next epoch is empty. Following this design, a user is online for x epoch lengths, where $1 < x < 2$. Thus, registering the user for two consecutive epochs guarantees the user will remain online for at least one epoch length. When both the epochs elapse, the user can be prompted to decide whether she wants to re-register or not.

The epoch model has therefore prevented the tracking of users by the Presence Server. Even if a user were to register with the same user-name each time she used the system, the name used to identify her by the Presence Server would change om every use, making it difficult for her activities to be tracked over an extended period of time.

3.4 Connecting to a Chat Partner

Once two users say, Alice and Bob, have registered themselves on the Presence Server and want to communicate, one of them will initiate the conversation by submitting a chat request to the Presence Server. Say Alice wants to submit this request. The request will consist of four fields: the chat recipient field, the chat requester field, the exit relay field, and the key exchange field. The chat recipient field will contain the hash of Bob's user-name concatenated with the current timestamp. The Presence Server can use this to look up the intended chat partner in its records. The chat requester field will contain the ciphertext resulting from encrypting Alice's user-name, using the hash of Bob's user-name as the symmetric key. The Presence Server will not have the necessary information to gain anything useful from this field, but Bob can decrypt it using the hash of his own user-name to discover the identity of the person trying to communicate with him. The exit relay field will contain the IP address and circuit ID of Alice's exit relay. The Presence Server will already have this information from its own connection with Alice's exit relay, but Bob will need this information in order to connect directly to Alice's circuit. Finally, the key exchange field

will contain the first message in establishing a shared symmetric key via the ACE Protocol [23].

Chat Recipient:	*H(Bob's user-name+ timestamp)*
Chat requester:	*E(Alice's user-name, H(Bob's user-name))*
Exit:	IP address of the exit node
Key Exchange:	Ace protocol's first message.

Table 3.1: Data structure of the request.

Once the request is constructed, it is sent to the Presence Server, which searches for the online presence of Bob based on the data in the chat recipient field. If Bob is online, a match will be found, and the Presence Server will forward the request to Bob via his circuit. Bob will receive the request and decrypt the requester's user-name, using the hash of his own user-name as the symmetric key. Bob will discover the request came from Alice, and will decide whether or not she wants to accept her request.

If Bob decides to accept, he will create and send a chat accept message to his exit relay. This message will consist of three fields: the chat partner field, the exit relay field and the key exchange field. The chat partner field will contain the ciphertext resulting from encrypting Bob's user-name, using the hash of Alice's user-name as the symmetric key. This field is required for Alice to verify that Bob is the one sending the message. The exit relay field will contain the IP address and circuit ID of Alice's exit relay. This information was received from the chat request and is being sent back to Bob's exit relay. The key exchange field will contain the response, specified by the ACE Protocol, for Alice to compute the shared symmetric key. This message is sent through Bob's circuit to his exit relay. Bob's exit relay will read the IP Address from the exit relay field and forward the message to Alice's exit

relay. Alice's exit relay will read the circuit ID from the message and identify the correct circuit to send the message through so that it reaches Alice.

Chat Partner:	*E(Bob's user-name, H(Alice user-name))*
Exit Relay:	IP address, circuit id
Key Exchange:	Response to the ACE protocol's first message.

Table 3.2: Data structure of the Chat Accept.

Once the request reaches Alice, her first step is to decrypt the chat partner field to ensure the message came from the expected user. Once she has confirmed this, she uses the information in the key exchange field to compute the shared symmetric key, and now she is ready to communicate with Bob. She immediately sends an acknowledgment message to Bob to confirm the establishment of the conversation. Alice's exit relay will now forward traffic to Bob's exit relay until it receives a *Disconnect* message, and vice-versa.

Alice and Bob now have four symmetric keys each, one for each of the relays in their circuit, and one they share with each other. Alice and Bob are effectively connected via a circuit consisting of 6 relays. When Alice sends a message to Bob, she encrypts it four times and passes it through her circuit. Each relay in the circuit decrypts the message before passing it on. Once the message reaches her exit relay, it is forwarded to Bob's exit relay. Traffic between Alice and Bob's relays is encrypted with their shared secret key and is therefore still secure. The message is then passed through Bob's circuit, which each relay encrypting the message before passing in on. Finally, Bob receives the message and decrypts it four times using his four keys to reveal the content of the message

3.5 Cover Traffic & Other Features

This subsection will cover several other features incorporated into the Lilac design to strengthen the security, anonymity, and usability provided by the system. One such feature is cover traffic, which involves inserting dummy messages into the system to increase the anonymity provided. In addition, the implementation of the Socialist Millionaire's Protocol provides a way to authenticate a user's chat partner as well as detecting a man in the middle (MITM) attack. The section will conclude by discussing the flexibility of the architecture in suiting the specific needs of various users.

3.5.1 Cover Traffic

Cover traffic involves inserting dummy messages into the system to increase the anonymity provided. Implementing cover traffic first requires the selection of an interval time, known as the *heartbeat interval*. This interval could be different for every node in the network. The default in Lilac is 300 milliseconds. We have experimented with different heartbeat intervals and we found that 300 ms is an appropriate interval. Once it has been selected, the method for sending outbound messages needs to be adjusted. Rather than sending outbound messages immediately, they are queued, with a separate queue being kept for each outbound connection. After the passing of a heartbeat interval (for example, after every 300 milliseconds) the node removes the first message from every queue and sends it to the respective recipient. If a queue is empty, the node generates a dummy message and sends it to along the queue's corresponding connection. Note that the term *node* here refers to clients, relays and the Presence Server. The implementation of cover traffic is useful in protected the system from statistical analysis of its traffic flow [24]. However, selecting the heartbeat interval involves a trade-off. Selecting a very short interval increases congestion in the network, whereas selecting a longer interval increases the latency experienced by the users, as

messages may be queued for this length of time at each relay. The results of operating the system under several different heartbeat intervals are discussed in the next section.

3.5.2 The Socialist Millionaire's Protocol

The Socialist Millionaire's Protocol [10] is an algorithm which is used when there exist two parties who both have some secret value, and want to determine whether their values are equal without disclosing any information about their secret. The parties partaking in the protocol can learn nothing other than whether their secret values are equal or not. SMP is included in Lilac to help prevent two possible security threats. The first is the man in the middle (MITM) attack, which involves an adversary intercepting and possibly altering traffic passing between Alice and Bob, who believe they are in direct communication with each other. The fact that every message between Alice and Bob passes through six relays makes this defense important in the Lilac design. The second threat is a user who may (maliciously) log in with the user-name Alice expects Bob to log in with. Alice will then attempt to start a conversation with this user-name expecting it to be Bob. SMP provides a way for Alice to securely authenticate her chat partner's true identity.

The SMP key which is selected to defend against these attacks is the concatenation of the symmetric key Alice shares with her chat partner and a secret passphrase entered by Alice. SMP will only be successful if Alice and Bob both have the same shared symmetric key and both enter the same secret pass phrase. The passphrase could be a secret which Alice and Bob have agreed upon online, or Alice could supply a question (which only Bob would know the answer to). Alice would provide the answer to the question as the pass phrase. Bob would attempt to answer the question sent to him and SMP would only succeed if his answer exactly matched the one entered by Alice. This way, SMP is possible without any prior arrangements, and Alice can successfully confirm whether or not she is actually talking to Bob.

For a malicious user, say, Mallory, to successfully execute an MITM attack, she would need to intercept traffic between Alice and Bob and establish different symmetric keys with both of them. Alice would falsely believe the symmetric key she shares with Mallory is shared with Bob. Similarly, Bob would falsely believe the symmetric key he shares with Mallory is shared with Alice. When Mallory receives the SMP request from Alice, she has two options. She may interfere and provide her own answer reply to the SMP request. This will cause the SMP to fail, assuming Mallory does not know the secret passphrase entered by Alice. Mallory could alternatively let the SMP take place between Alice and Bob uninterrupted. SMP will also fail in this case because Alice and Bob possess different symmetric keys, and the secret they use for SMP will be different. Therefore, Mallory has no method of forcing the SMP to evaluate to true, and Alice and Bob will know their connection is not secure.

3.5.3 Architecture Flexibility

There are numerous possible applications of an anonymous communication system such as Lilac. Its use as an open, online chat system is the main focus of this work, but the flexibility of the architecture of Lilac opens up several other possibilities. Any organization may run its own, private instance of Lilac and tune it to their specific needs. An organization may establish a Directory Server private to their own network, as well as relays and a Presence Server which reports to this private Directory Server, thus creating the organization's very own instance of Lilac. The flexibility of the design also aids in battling Internet censorship.

CHAPTER 4

EXPERIMENT DESIGN

To test for the validity of our proposed design, we designed and executed a set of experiments using a real-world Internet deployment setup. Our experiment design consisted of running Lilac with a single Directory Server and Presence Server and twelve relays. We then measured the performance of the system under the use of a thousand clients. These clients were run on ten systems (running one hundred clients each). We required our clients and relays to be geographically dispersed to more accurately simulate the real world environment. To accomplish this, we used Amazon Web Services (AWS) to run most of our relays and clients, and our Presence Server. AWS provides server instances in 9 geographic locations, namely Northern Virginia, Oregon, Northern California, Ireland, Frankfurt, Singapore, Tokyo, Sydney and Sao Paulo. These nine locations, along with our lab in Arlington, Texas, allowed us to globally distribute our system for experimentation. Table 1 summarizes the architecture.

The experiment was run several times with varying heartbeat intervals. Conducting these experiments required automating the client-side code to automatically create a circuit and register on the presence server with a random string. The code-base for the relays was not

Server Name	# of Servers	RAM (GB)	Location(s)
Directory Server	1	4	Arlington, TX
Presence Server	1	32	Frankfurt
Relay	12	16	All
Client	10	8	All

Table 4.1: Summary of Experiment Architecture

altered, but the behavior of the Presence Server was slightly changed. The Presence Server maintained a variable, called *lastRegistered*, for the last user registered. The variable was was initialized as empty. When the Presence Server received a registration request, it would check the value of *lastRegistered*. If it was empty, the Presence Server would set its value as the user-name it received in the request. If it was not empty, the Presence Server would return its value to the client making the request, and subsequently set its value to be empty. If the client received a user-name from the Presence Server, it would immediately send a request to the Presence Server to begin a chat with this user. If not, the client would wait for a chat request. To illustrate this, The Presence Server would initiate with *lastRegistered* = *null*. Client A would register with user-name x. The Presence Server would set *lastRegistered* = *x*. Client B would then register with user-name y. The Presence Server would send back x to Client B and set *lastRegistered* = *null*, returning it to its initiation state. Using this scheme we are able to continuously initiate communication channels between random clients.

To measure the performance of the system, the automated clients would first create a circuit and record the circuit build time. It would then establish a connection with a chat partner as described above. Clients would then send and receive messages continuously with their respective chat partners. A client would send a message and record the time the message was sent. When the client would receive a message, it would calculate the time elapsed since the last message was sent and record this calculated value in a log file. The client would then send another message, record the time it was sent, and continue in this loop indefinitely. The result is a log file containing the time taken to build the circuit, followed by a list of times taken for a message to be sent to a chat partner and returned to the sender. We refer to this number as the round trip time (RTT). The message-send time, which is the time it takes for a message to reach its recipient, would be half this value. We conducted the experiments as described above twice, with the heartbeat interval set at 300 and 500 milliseconds respectively. Our results are summarized below.

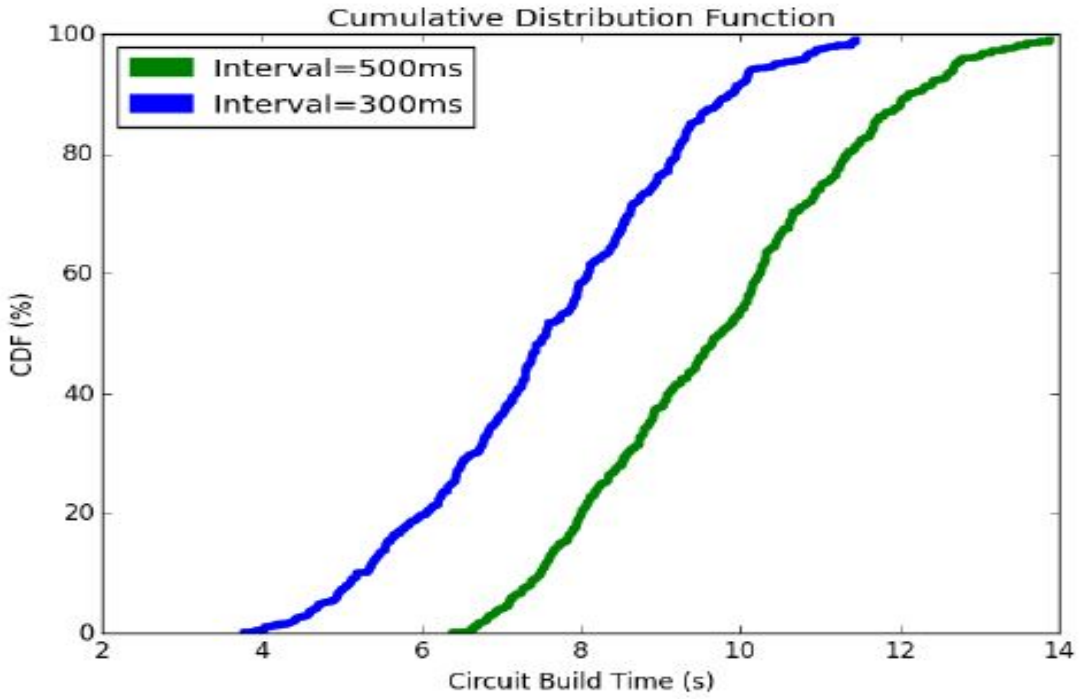

Figure 4.1: Cumulative Distribution Function for Circuit Build Times.

As shown in Figure 4.1, Circuit Build Times are kept to under 10 seconds in most cases. The delay could be seen as lengthy, but this is a one-time cost for the user which they may not even notice if they are busy multi-tasking on their computer, which most users are. The message RTT has a median value of 3 seconds for a 300 millisecond heartbeat interval setting. This means that most messages reached their recipient in one and a half seconds or less, which is more than reasonable for an instant messaging service, especially one providing the levels of privacy and security that Lilac provides.

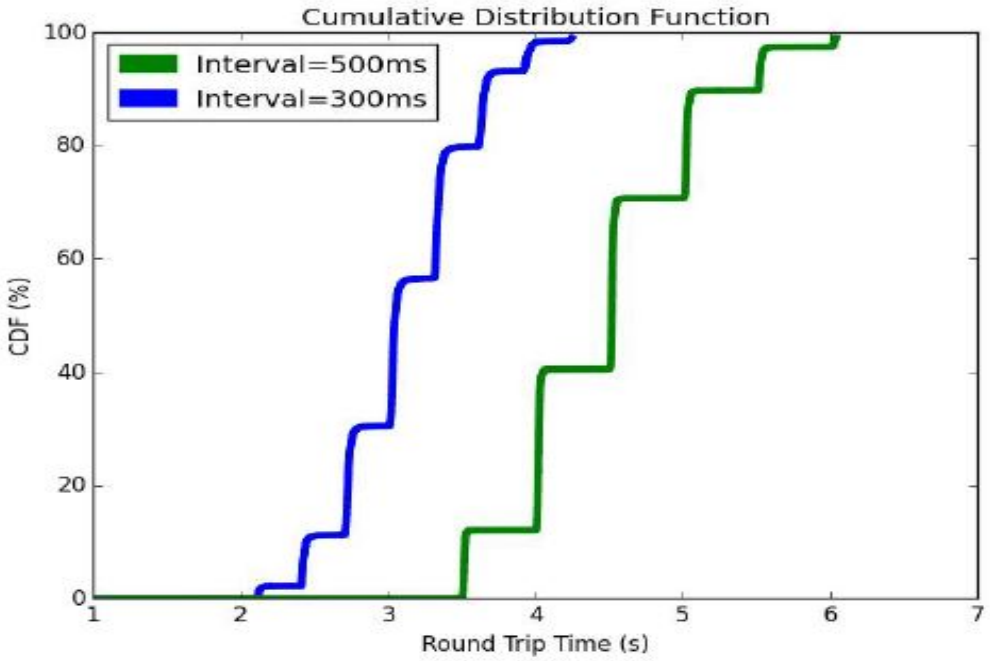

Figure 4.2: Cumulative Distribution Function for Message Round Trip Times.

4.1 Discussion of the load on Servers

So far, we have seen the results about the circuit build time and the Round trip time. In this section we will discuss how does the addition of the users effect the CPU utilization of the servers. We will discuss the load distribution on the relay server with respect to the time. It took an approximate of 4 hours to finish the experiment. We have mapped those 4 hours to the percentage of 100.

4.1.1 Relays

Below is the graph for how the CPU Utilization increased over time when clients are connected. We have mapped the experimentation time in terms of Percentages and plotted the average of CPU Utilization. Please, notice that the CPU Utilization increases with the

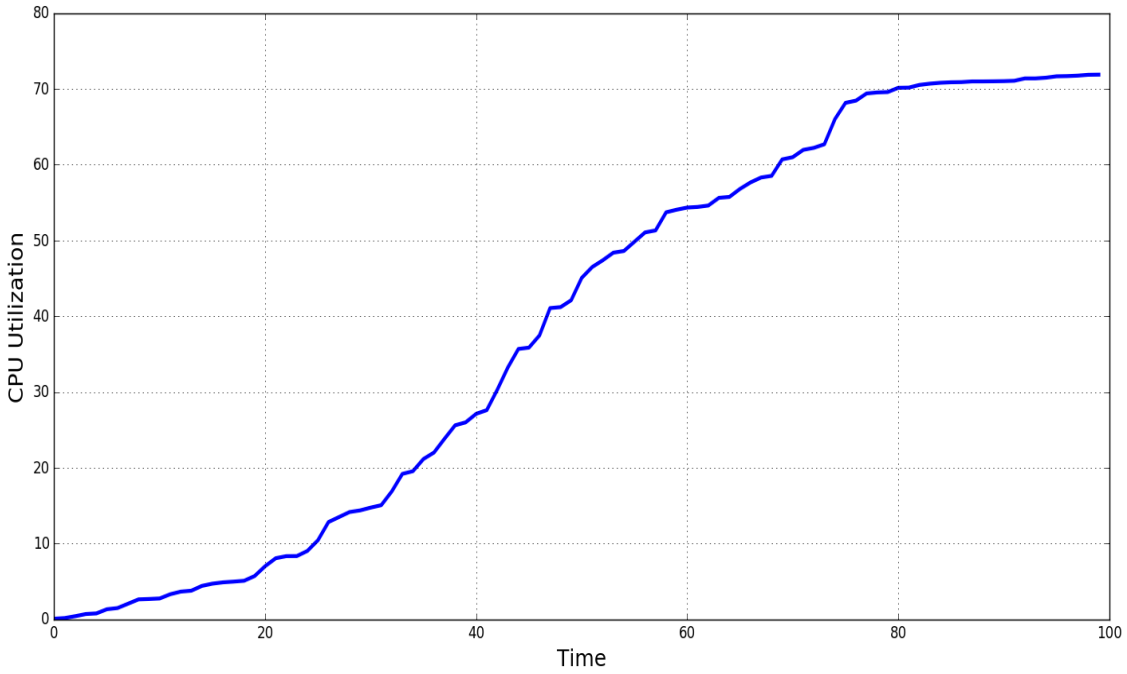

Figure 4.3: Average of Relay servers CPU utilization vs Time

increase in the time. The time is directly proportional to the number of clients connected to the server.

4.1.2 Presence Server

Here is the below graph for the presence server. We have logged the number of users registered to the presence server and recorded the load on the machine. Here is the graph for the number of users added to the presence server vs the CPU utilization. The presence server load increases with the increase in the number of users. This is because, whenever the users register to the presence server, the exit relays of the users send heartbeats to the presence server.

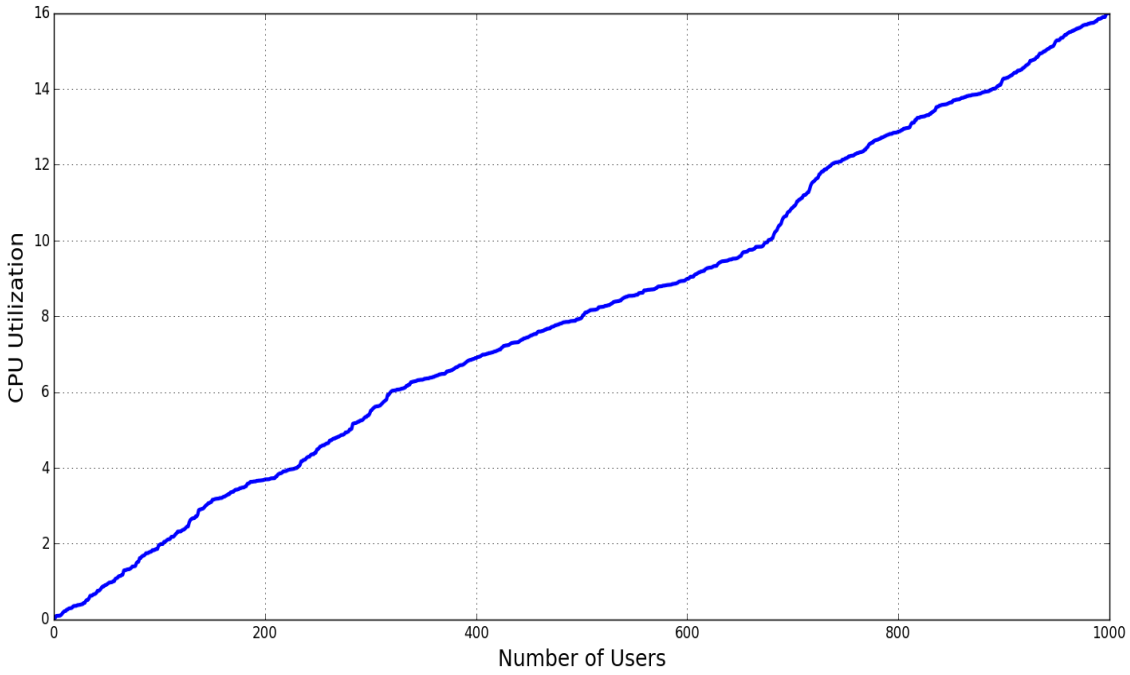

Figure 4.4: Presence Server CPU Utilization vs Number of users.

CHAPTER 5

SYSTEM DEMONSTRATION

5.1 System Implementation

In this section we will explain what software tools and programming languages are used to implement Lilac's architecture.

Lilac is written in JavaScript. We have used nodejs [25] framework to develop Lilac. These are the following important packages that are implemented in developing Lilac. Currently, Lilac is deployed on AWS [26], Heroku[27] and on PlanetLab[28]. This section also explains how to install your own private Lilac network.

1. Main programming Language - JavaScript.
2. Shell Script - To install dependencies and other configuration files.
3. SJCL - Stanford JavaScript Crypto Library.
4. Google Crypto Library.
5. Bootstrap.
6. ecc2519 - To generate private and public keys.
7. AWS - To host Lilac and relay servers
8. Heroku - To host relay servers.

AWS stands for Amazon web services. We have deployed Directory server, Presence server on AWS. We have used a couple of Load balancers for Directory Server and Presence Server in case if there is an increase in traffic. To protect Lilac from DoS or DDoS attacks we have put Lilac behind cloudflare[29]. It means all requests to directory server has to go through Cloudflare. Lilac provides another option to run directory server client from local-

host. We have hosted relay servers on Heroku containers and on planetlab. The complete map of the lilac network is shown below.

Figure 5.1: Lilac Network.

Red Markers stand for relay servers and green marker stands for presence server.

5.1.1 Deploying Lilac on AWS

Please follow the steps to create your own private Lilac network.

1. Get the source code of Lilac from *https://github.com/revanthpobala/Lilac*.
2. Go to aws console in aws.amazon.com and create 3 or more instances of ec2. Optionally, you can deploy Lilac on AWS elastic bean stalk.
3. On one ec2 instance, install the required dependencies such as nodejs, python.
4. Once everything is installed, please unzip the Lilacs' source code and run the following command *npm start* to start the server.

5.1.2 Deploying Lilac on Heroku and on other Containers

To get rid of load balancers and other infrastructure you can deploy Lilac on Heroku or other containers. Download the source code of Lilac and deploy the app on Heroku. No need to change any configuration files. Make sure that you have *Procfile* in the source code.

5.2 User Interface and System Demonstration

This section will explain about the system implementation and various features that are available in Lilac. This section will have the following items.

1. Login page.
2. Home Page.
3. SMP.
4. Contact store.
5. Send images and files which are present somewhere on the internet securely.
6. Log out

5.2.1 Login page

Login page is the first page you get to visit when you type www.thelilacproject.org in your url box.

There are multiple good things about lilac login's page. One good thing is that you do not require any registration to login. The user-name can be anything from 1 character(except space) to 256 characters. Lilac uses HTML and CSS sanitization scripts to avoid command injection or any malicious input. The second field in the login page is a Password field. If you want to store your contacts and if you want to retrieve the contacts later, you need to enter the password. If you do not enter the password, you wont be able to store your contacts. Moving on to the check box, you will find this option **Allow my friends to find me.** If

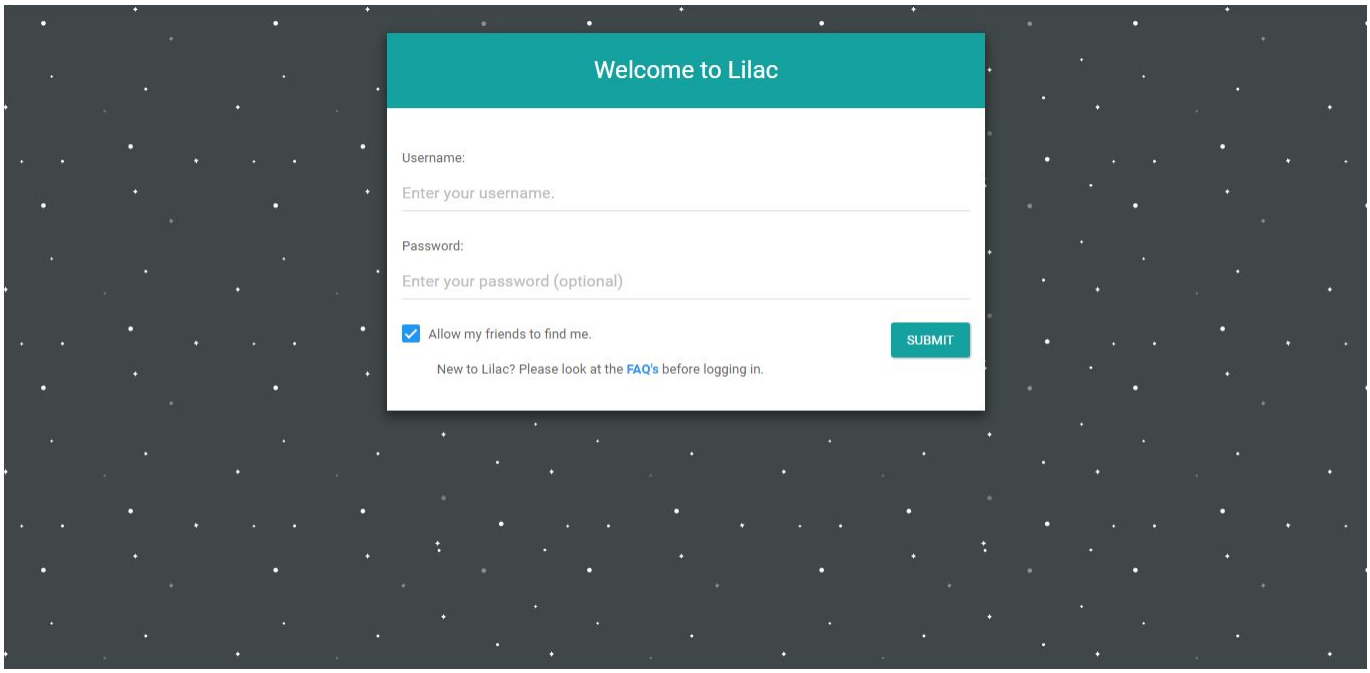

Figure 5.2: Login page.

you check the box, your user-name will be registered to the presence server. It means your friends can find you by your user-name. Instead if you want to opt out the checkbox, your user-name won't be registered to the presence server. You need to send a request to your friend, and your friend needs to send you the request. The request should be bi-directional.

5.2.2 Home Page

The Home page has a minimalist design and It contains the user-names and a menu. To avoid shoulder surfing you can hide your contacts by clicking or touching "<" button on the header.

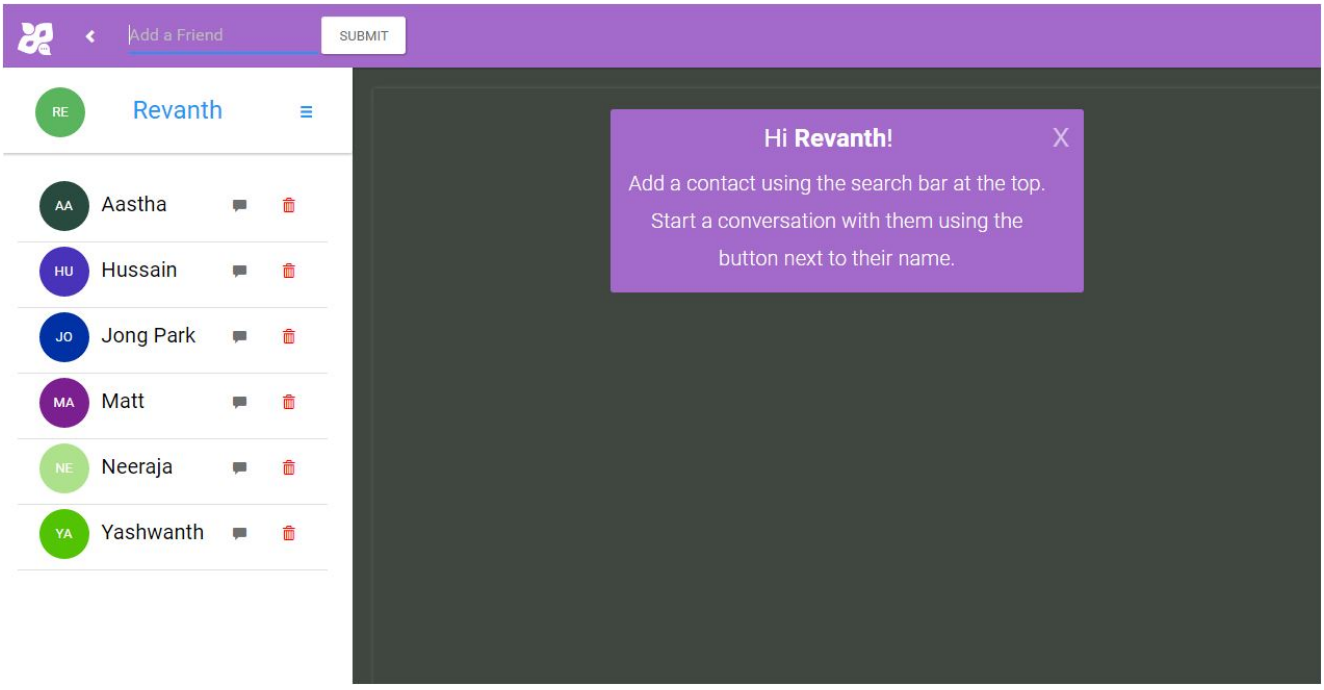

Figure 5.3: Home page.

5.2.3 Socialist Millionaire Protocol

In this sub section we will learn how to implement SMP in Lilac. On the home page you find a + button. Once you click the button you will have multiple options such as SMP, End conversation. When you click on SMP a modal appears in the following figures. You need to provide a passcode. A passcode may be any integer ranging from 1 character to 256 characters. Please, note that HTML and CSS sanitization is implemented to avoid XSS attacks. Once you enter the passcode, the other user will also get the model requesting to enter the passcode. If he enters the same passcode which you entered, then you have successfully authenticated him/her.

5.2.4 Contact Store

In Lilac the user has an option to store his/her contacts securely. Lilac, doesn't have a database, so we have used Browser's *Local Storage* to store the contacts. Automatically,

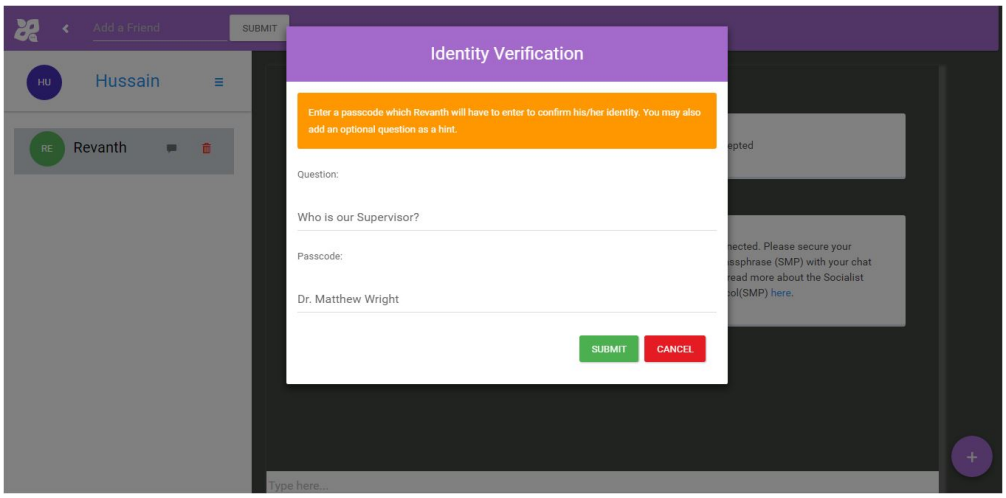

Figure 5.4: SMP initialization.

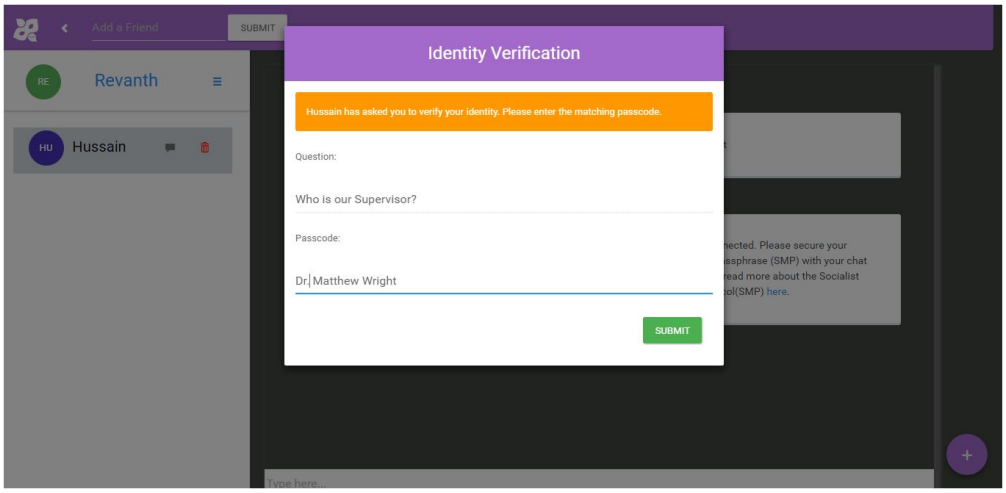

Figure 5.5: SMP Authentication.

Lilac stores the users contacts when the password is provided during login. The contacts are stored in the following format.

H - SHA 224, E- AES Encryption.

Key = *H(user-name + contact name + password)*

Value = *E(contact name + user-name+ password)*

```
"063f01244687c7caa80c00cf576955603ed1b5ce84035b0c5db36e51":
"U2FsdGVkX1/zkY9ksNH1nsXdOIYSWDzXOpm4+lcLoyo="
```

Key	Value
063f01244687c7caa80c00cf576955603ed1b5ce84035b0c5db36e51	U2FsdGVkX1/zkY9ksNH1nsXdOlYSWDzXOpm4+lcLoyo=
3ad01bcd8992a2a0e7e6e23d7b1daba5fbbe7f7ad0ca0bd9ee984d21	U2FsdGVkX1/Czm/HtCbRPEH56hettloU1EdlvQNmv/A=
58d013a38498a99cc1f87d2da5b00400022482012 4d4d8bc03e6a71a	U2FsdGVkX1/fm0lHzTuNNkPKUln6p77nsM0Oz9stZXg=
a05c061327590f3e2bb8c6802562a72222a13ed6fc2320a8495884b2	U2FsdGVkX18uAj/lf7CBYs0x4rsJ1oYdNZAA46smen8=
debug	undefined

Figure 5.6: Contact Store

5.2.5 Sending Images and other files over Lilac

Lilac currently, doesn't support sending of files or images over it's network because sending large files or images can create an overhead and congestion in the network. If the resource is available somewhere on the internet and if you want to send the information securely, you can link the content in the message. In future we are planning to have a network to deliver the files securely and anonymously by using DP5 protocol [20].

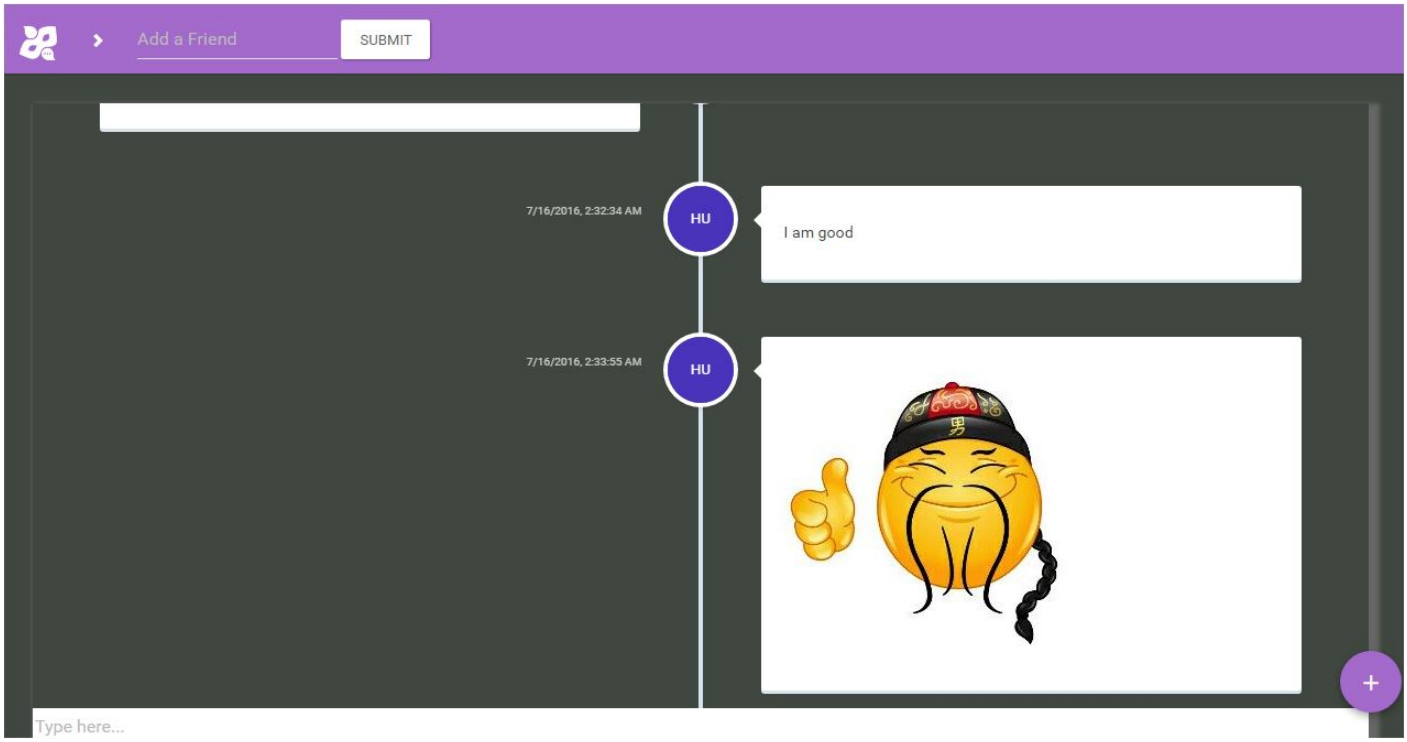

Figure 5.7: Send images or other files using Lilac.

5.2.6 Log out

A user can log out using the logout option provided in the menu. Once the user clicks the logout button, the user is logged out of the system and a new circuit is formed.

5.3 Lilac in the wild

We have kept running Lilac for 7 months continuously and asked few users to use the system. The users reported that sometimes, they have faced problems in logging in to the system or, the circuit build time is high. Here are some analytics on Lilac.

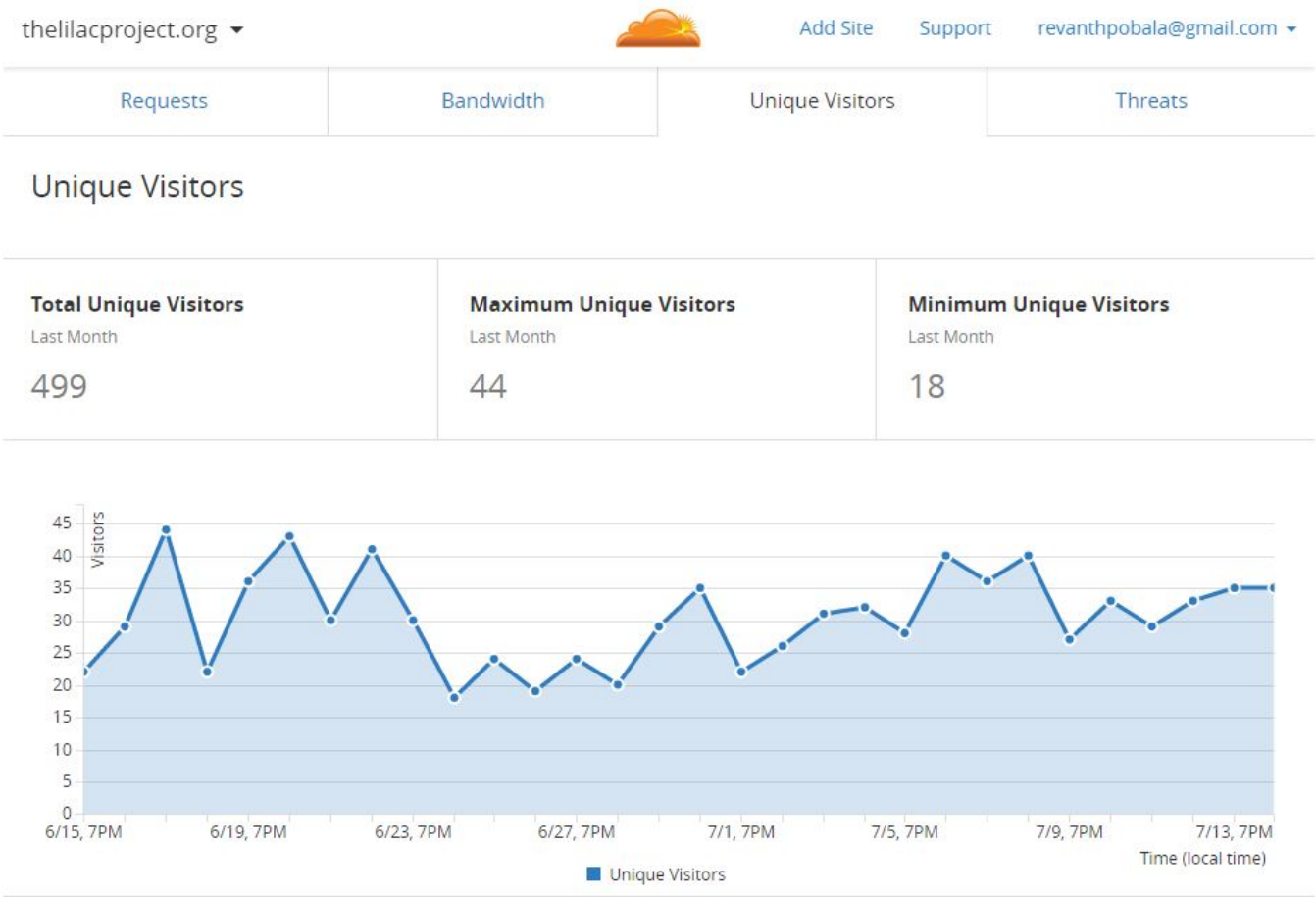

Figure 5.8: Total number of unique visitors.

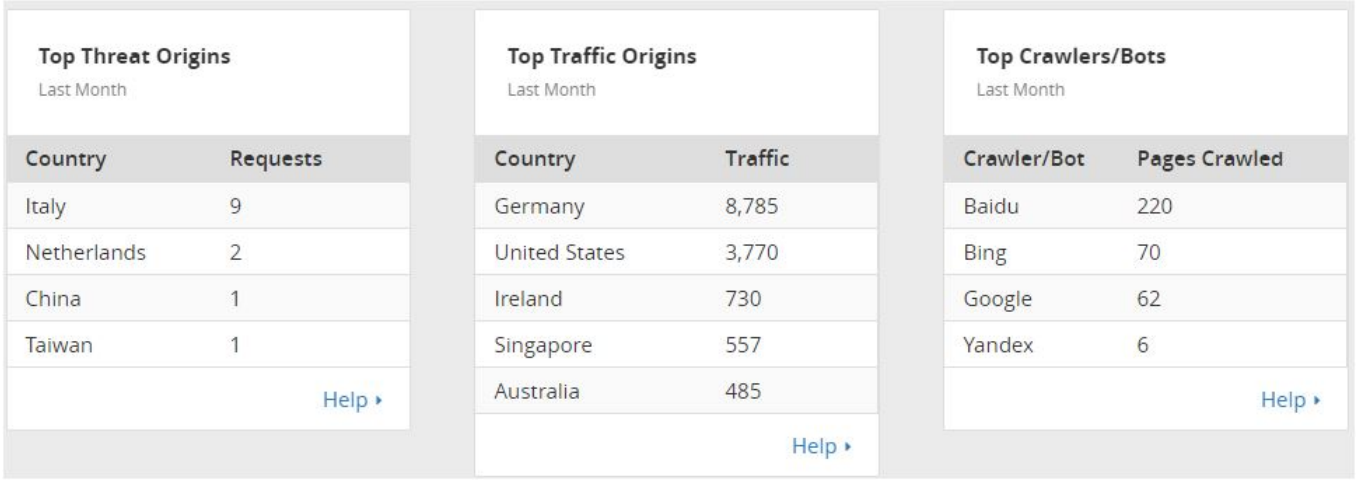

Figure 5.9: Traffic origins.

CHAPTER 6

CONCLUSION AND FUTUREWORK

This work serves a wider audience who care about security and privacy in their communications. Lilac network is distributed and has no centralized control over the network. We have deployed Lilac on AWS, Heroku and on a normal server. The study shows a reasonable trade off against the performance, security and usablilty of the system. It has been tested with bots as well as humans and the results were positive.

6.1 Conclusion

We have proposed the system for Lilac -Lightweight lowlatency anonymous chat and deployed the platform on real world internet. The system comprised of a Directory server - The server that keeps track of the network topology , Presence server - The server that brokers the connection between two users , Relay Servers- The voluntary servers, that take part in the circuit building process. The system is tested with real users and bots. The average round trip time (RTT) is around 3.5 seconds with a circuit build time of 7 seconds. The deployed system does not require any change in configuration files or settings. The system can be readily deployed on normal servers, containers and on virtual machines. Lilac is customizable and it is open sourced.

6.2 Future work

So far, we have created a network, that is basically for Lightweight communication such as instant messaging. In future, we can use the network to perform various transactions securely in the areas related to file transfer to Internet of Things (IoT). Currently, in the

design, there is only one presence server backed by few load balancers. In future we can have a protocol to have multiple presence servers. One proposal is to have a presence server based on the region. For ex: If there are 5 regions, then we can have a single presence server per region and a main presence server. When the user searches for another user, if the user is not present in the regional presence server, the regional presence server will ask for the user on the global presence server. The global presence server will query the user and make a connection when it is found. The cover traffic implemented in the system should be optimized to increase the efficiency and performance of the system. We would like to have a browser plugin and apps on iOS and Android devices in future.

APPENDIX A

Important Algorithms used in Lilac

In this appendix we will discuss what protocols, encryption algorithms and miscellaneous tools which we have used to build Lilac.

A.1 ECC25519 Curves

We have used ECC25519 curves [30] to generate private and public keys for lilac. Once the key pair is generated we used the key pair to generate session keys.

A.2 ACE protocol

To establish a secret we have implemented ACE protocol [23]. ACE protocol provides a way to establish a secret. The secret is used as a session key to encrypt the outgoing messages thus contributing to forward secrecy and OTR.

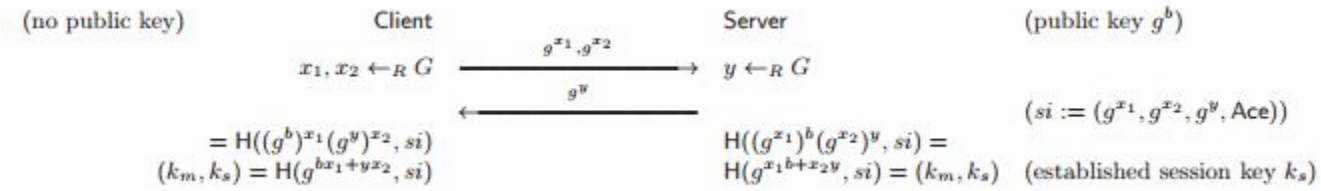

Figure A.1: ACE Protocol.

G is the exponent group.

A.3 AES -GCM

We have used AES-GCM encryption to encrypt the messages using the session key. GCM mode guarantees that the encrypted message is not tampered during the transport as it includes an iv along with the cipher text. The function is implemented from Stanford's javascript crypto library.

A.4 SMP Algorithm

Step No	Alice	Multiparty	Bob
1	Message x Random a,r, α	Public p,h	Message y Random b,y, β
2		Secure g = [h|a,b]	
3		Secure γ = [h|α, β]	
4	Test $h^b \neq 1, h^\beta \neq 1$		Test $h^a \neq 1, h^\alpha \neq 1$
5	$P_a = \gamma^r, Q_a = h^r g^x$		$P_b = \gamma^s, Q_b = h^s g^y$
6		Insecure Exchange P_a, Q_a, P_b, Q_b	
7		Secure c = $[Q_a Q_b^{-1} \| \alpha, \beta]$	
8		Test $P_a \neq P_b, Q_a \neq Q_b$	Test $P_a \neq P_b, Q_a \neq Q_b$
9	Test c = $P_a P_b^{-1}$		c = $P_a P_b^{-1}$

Table A.1: Socialist Millionaire Protocol

p = A Large prime number.
α, β, r, y, b, a are Random numbers.
In Lilac message x, y are the Hashes of the secret.
x = *H(Secret on user 1's end)* and y = *H(secret on user 2's end)*

REFERENCES

[1] EFF, 2015. [Online]. Available: https://www.eff.org/node/82654

[2] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

[3] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on tor by realistic adversaries," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 337–348.

[4] N. Borisov, I. Goldberg, and E. Brewer, "Off-the-record communication, or, why not to use pgp," in *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. ACM, 2004, pp. 77–84.

[5] A. M. Johnston and P. S. Gemmell, "Authenticated key exchange provably secure against the man-in-the-middle attack," *Journal of cryptology*, vol. 15, no. 2, pp. 139–148, 2002.

[6] S. J. Murdoch and P. Zieliński, "Sampled traffic analysis by internet-exchange-level adversaries," in *International Workshop on Privacy Enhancing Technologies*. Springer, 2007, pp. 167–183.

[7] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[8] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman, "Mixmaster protocolâĂŤversion 2," *Draft, July*, vol. 154, 2003.

[9] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Uncovering spoken phrases in encrypted voice over ip conversations," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 4, p. 35, 2010.

[10] F. Boudot, B. Schoenmakers, and J. Traore, "A fair and efficient solution to the socialist millionairesâĂŹ problem," *Discrete Applied Mathematics*, vol. 111, no. 1, pp. 23–36, 2001.

[11] A. Upadhyaya, *LILAC: ARCHITECTURE FOR ANONYMOUS LIGHT WEIGHT COMMUNICATION*, 2013 (accessed June 3, 2016).

[12] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type iii anonymous remailer protocol," in *Security and Privacy, 2003. Proceedings. 2003 Symposium on*. IEEE, 2003, pp. 2–15.

[13] A. Pfitzmann, B. Pfitzmann, and M. Waidner, "Isdn-mixes: Untraceable communication with very small bandwidth overhead," in *Kommunikation in verteilten Systemen*. Springer, 1991, pp. 451–463.

[14] O. Berthold, H. Federrath, and S. Köpsell, "Web mixes: A system for anonymous and unobservable internet access," in *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 115–129.

[15] R. Böhme, G. Danezis, C. Diaz, S. Köpsell, and A. Pfitzmann, "On the pet workshop panel âĂIJmix cascades versus peer-to-peer: is one concept superior?âĂİ," in *International Workshop on Privacy Enhancing Technologies*. Springer, 2004, pp. 243–255.

[16] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing," *Communications of the ACM*, vol. 42, no. 2, pp. 39–41, 1999.

[17] R. Dingledine and S. J. Murdoch, "Performance improvements on tor or, why tor is slow and what weâĂŹre going to do about it," *Online: http://www. torproject. org/press/presskit/2009-03-11-performance. pdf*, 2009.

[18] W. Wang, M. Motani, and V. Srinivasan, "Dependent link padding algorithms for low latency anonymity systems," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 323–332.

[19] N. Kobeissi and A. Breault, "Cryptocat: Adopting accessibility and ease of use as security properties," *arXiv preprint arXiv:1306.5156*, 2013.

[20] N. Borisov, G. Danezis, and I. Goldberg, "Dp5: A private presence service," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 4–24, 2015.

[21] M. Edman and P. Syverson, "As-awareness in tor path selection," in *Proceedings of the 16th ACM conference on Computer and communications security.* ACM, 2009, pp. 380–389.

[22] R. Snader and N. Borisov, "A tune-up for tor: Improving security and performance in the tor network." in *NDSS*, vol. 8, 2008, p. 127.

[23] M. Backes, A. Kate, and E. Mohammadi, "Ace: an efficient key-exchange protocol for onion routing," in *Proceedings of the 2012 ACM workshop on Privacy in the electronic society.* ACM, 2012, pp. 55–64.

[24] N. Mallesh and M. Wright, "Countering statistical disclosure with receiver-bound cover traffic," in *European Symposium On Research In Computer Security.* Springer, 2007, pp. 547–562.

[25] "Node js," https://nodejs.org.

[26] "Amazon web service," https://aws.amazon.org.

[27] "Heroku containers," https://www.heroku.com.

[28] "Planetlab," https://www.planetlab.org.

[29] "Cloudflare," https://www.cloudflare.com/.

[30] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography.* Springer, 2006, pp. 207–228.

BIOGRAPHICAL STATEMENT

Pobala Revanth Rao was born in India in 1992. He received his Bachelor's degree from the Nagpur University , Nagpur India in 2013. He has been a part of iSec, the Information Security Lab, from 2014. He has worked as a Security Assurance Research and Development Intern at BlackBerry Corporation.