Zephyr: Low Overhead Anonymous Private Messaging

Friedrich Doku[1]

Abstract. Private messaging over internet-related services is difficult to implement. Regular end-to-end encrypted messaging systems are prone to man-in-the-middle attacks and only hide messages but not the identity of their users. For example, WhatsApp [15] offers a strong privacy guarantee but does not hide much metadata because it uses end-to-end encryption. Other messaging systems such as Vuvuzela [6] that do hide the identity of their users consume a lot of computing resources, making them expensive to deploy.
Zephyr is an anonymous messaging system that protects the privacy of message contents and message metadata. Users that use Zephyr do not reveal who they are talking to or the contents of their messages. The goal of Zephyr is to decrease the amount of information being sent by the user and hide as much metadata as possible while being low on system resources.

Keywords: Anonymity · Communication · Privacy.

1 Introduction

Most private messaging systems today rely on an end-to-end encryption scheme and do not do a good job at hiding metadata. Metadata is all an adversary needs to figure out your identity and other personal information. As a result of electronic surveillance of private communication systems, many private messaging systems will be end-to-end encrypted or offer some kind of security. The security that they offer is no more than a basic system sprinkled with "encryption dust". For example, a simple communication channel with public-key encryption. Encryption alone is not enough to guarantee a secure system. A truly secure system is designed for security.

Other private messaging systems such as Tor are a step forward in ensuring privacy but are still not reliable. A user of Tor can have their identity compromised by the use of traffic analysis [11]. Tor relies on a large number of users to ensure privacy. Cover traffic can be used to hide the user's identity but is very expensive and only offers limited protection [12]. Messaging systems that offer a "self destruct" feature that destroys messages for both users, such as Telegram still don't ensure much privacy because the data can be recovered with the use of computer forensic tools.

This paper presents Zephyr, a system that provides private messaging and hides as much metadata as possible without sacrificing performance. Zephyr does not rely on a large number of users to ensure privacy: only two users are needed

to ensure an adversary will not be able to figure out the identity of the users. The system's security is not dependant on the number of users. Furthermore, Zephyr uses cryptographic algorithms that enable it to quickly encrypt large amounts of messages without high overhead hence making it possible to run Zephyr's server components on low-performance personal computers and smartphones.

Zephyr uses a mixnet that hides metadata by routing messages through multiple proxy servers (mixers). A mixer in Zephyr's mixnet can be run on a variety of computing devices from a high-performance server in the cloud to a low-performance Raspberry Pi. Each user in Zephyr chooses random mixers and downloads all of her selected mixers' public-keys. Once the keys are downloaded the user generates a random permutation and adds layers of encryption in the specified order. Select mixers decrypt a layer of encryption and shuffle their messages using random permutations only known to themselves and send their messages to the next mixer in that order. The mixer that removes that last layer of encryption from a message sends it to a public mailbox where users will find and download their messages.

Zephyr's drawback is that to protect itself from traffic analysis it has to make some performance sacrifices. A powerful adversary with access to internet service providers can monitor the message output paths of each mixer. Zephyr solves this problem by spreading its nodes over large geographical areas because it is hard for an adversary to gain access to overseas internet service providers. However, doing so increases the overall latency of Zephyr, making it unsuitable for instant messaging.

Nevertheless, Zephyr still performs better than traditional end-to-end encrypted messaging systems in terms of security. Zephyr's latency may be much higher than "secure" instant messaging systems but it provides a greater security guarantee.

All in all, we make the following contributions:

- Analyze the design of Zephyr, a private messaging system that keeps its users anonymous while being low on computing resources.
- Evaluate an implementation of Zephyr running on a laptop with a Raspberry Pi as a mixer node to showcase performance on a low-performance computer.

2 Related Work

End-to-end encryption messaging systems. Recent work has shown that secure messaging systems can provide end-to-end encryption while being low on computing resources. However, these systems such as WhatsApp [15] and Matrix [1] do not protect the identities of their users; they only keep the contents of their messages secure. In contrast, Zephyr can keep its users' identities anonymous while protecting the contents of their messages at the same time.

Systems that provide anonymity. Anonymous communication has been studied for years and previous works should not be ignored. For example, David Chaum's works on mix networks in the late 1970s [3]. Chaum's mix network

requires every mixer in its network to processes the data, which increases the latency and overhead of each mixer, unlike Zephyr where users must select only a few mixers to lower overhead. Chaum's mix networks also do not provide any protection from major traffic analysis attacks. Chaum's best-case assumes that the adversary will only be able to monitor traffic to and from the mixnet. If any traffic is monitored inside of the mixnet, then the strategies of the mixnet are known. In contrast, Zephyr assumes that all but one communication link in the mixnet is being monitored.

Tor's routing policy makes it prone to traffic analysis [14]. More specifically, the predecessor attack [16]. This is because Tor has a higher preference for selecting nodes with higher bandwidth. Although this decreases the overall latency of the system, it puts a user's identity in danger by making them more vulnerable to traffic analysis. To defend against this and related attacks, Zephyr only allows users to use a small set of randomly selected nodes.

Atom [8] requires messages to be routed through hundreds of servers, which causes the system to have high latency. Unlike, Zephyr where users are required to route their messages through a few mixer servers. Atom also makes use of expensive cryptographic primitives, which makes it unsuitable to run on low-performance computers. Atom makes use of a modified ElGamal mixnet. Unlike Zephyr's mixnet, Atom's does not allow its users to know which mixers they are using. Thus, users will be unable to verify the transmission of their messages.

Riffle [9] is a low latency anonymous messaging system that verifies the integrity of its mixers using zero-knowledge proofs [10]. This method makes it possible for other parties to verify that the mixer did not tamper with their messages. However, computing zero-knowledge proofs as well as verifying them can be computationally expensive. This makes Riffle unable to run mixers on low-performance computers with limited memory and CPU power.

Vuvuzela [6] can achieve a strong degree of privacy from a global passive attacker by using differential privacy techniques to hide the communication patterns between its users. However, Vuvuzela exploits a dialing protocol that consumes a lot of bandwidth. This protocol makes users download conversation invitations as well as random noise, which is indistinguishable from real user requests. This random noise is needed to protect against traffic analysis attacks. Before users can communicate they must use the dialing protocol each time they want to start a new conversation with another user, which means they will have to consume excess bandwidth each round.

Vuvuzela adds noise to observable variables to protect the identities of its users. However, noise is expensive and can consume significant amounts of bandwidth. Unlike Vuvuzela, Zephyr does not rely on expensive differential privacy techniques. Zephyr achieves a similar degree of privacy while consuming less bandwidth.

3 Threat Model and Goals

Zephyr aims to keep its users anonymous and protect the contents of their messages while being low on computing resources. In this section, we describe Zephyr's threat model and goals.

3.1 Threat Model

Zephyr's deployment consists of a decentralized network of hundreds to thousands of servers controlled by different individuals and organizations or just a single individual or single organization with some of these servers overseas. Every user in Zephyr will agree on a set of participating servers.

We assume that some adversary has taken control of every internet service provider (ISP) in one of the countries Zephyr is being used in. All of Zephyr's traffic is being monitored by that adversary and that adversary controls all but one mixer. The controlled mixers may change the order messages are sent and substitute an adversary's public-key with the mixers original public-key.

All cryptographic assumptions are standard (the adversary will not be able to break cryptographic primitives). We assume that Zephyr's mixer public-keys are known to all users and the adversary.

Zephyr is not a fault-tolerant system. It does not protect from denial-of-service attacks or server component failures, more specifically, Byzantine server faults. However, it can recover from minor server failures. Since we do not cover availability attacks on Zephyr, this is left out of scope.

Cryptographic Primitives Zephyr exploits two cryptographic primitives which are described here.

Identity Based Encryption. Zephyr relies on the Boneh-Franklin Identity-Based encryption scheme to generate public and private-key pairs for its users [2], which consists of the following algorithms:

- $(mk, params) \leftarrow \mathsf{Setup}(k)$. Generates system parameters $params$ and the master-key mk from a security parameter k.
- $(d) \leftarrow \mathsf{Extract}(params, mk, ID)$. Generates the user's private-key d given the $params$, mk, and the user's id ID.
- $(C) \leftarrow \mathsf{Encrypt}(params, ID, M)$. Encrypts a message M from $params$ and ID to create a ciphertext C
- $(M) \leftarrow \mathsf{Decrypt}(params, C, d)$. Decrypts C from d, $params$, and mk to generate M.

XSalsa20 Authenticated Encryption. Zephyr also exploits XSalsa20: a stream cipher based on Salsa20 that uses a 192-bit nonce instead of a 64-bit one found in its predecessor. XSalsa20 is used in Zephyr because of its growing popularity due to its speed and immunity to timing attacks. Each mixnet server and user in Zephyr exploits this encryption scheme.

- $(C) \leftarrow \mathsf{AEncrypt}(n, M, pk)$. Encrypts a message M from a public-key pk and a cryptographic nonce n to create a ciphertext C
- $(M) \leftarrow \mathsf{ADecrypt}(n, pk, C, sk)$. Decrypts C from the recipients pk, private-key sk, and n.

3.2 Goals

Zephyr has three primary goals.

Correctness. Informally, Zephyr is correct if every honest user is able to successfully send her message to her intended recipient and the intended recipient is able to receive her message after each round of Zephyr's protocol.

Anonymity. Similar to previous work [6,8], Zephyr achieves the goal of providing anonymity to its users if an adversary that controls a majority of Zephyr's servers is unable to determine the identity of a user and who the user is communicating with by no more than random guessing.

Low Overhead. Zephyr achieves the goal of having low overhead if its server components are able to run with acceptable performance using very few computing resources, more specifically, CPU time, bandwidth, and memory. Zephyr should be able to support its users when running with several low-cost computing devices with at least 2GB of RAM and a CPU clock speed of at least 1.5GHz.

4 Overview

Figure 1 presents an overview of Zephyr. At a high level, Zephyr consists of five different entities: private-key generators (PKGs), users, mixers, information-nodes, and mailbox servers. For the sake of simplicity, the information-nodes are not included in Figure 1. Every user in Zephyr has a mailbox associated with them; users will share mailboxes.

To set up the network, each node in Zephyr will download a configuration file from a trusted source that contains the IP addresses of all participating nodes and the length of Zephyr's round. Once the list has been downloaded, the nodes will begin their initial operations, such as generating public/private key pairs and establishing connections with other nodes. Nodes will check if every other node is ready before moving on.

Zephyr's model contains a central authority that coordinates the actions of the system. The coordinator starts new rounds. The length of each round in minutes T is determined by a parameter in the configuration file. During each round, the coordinator will allow T minutes for users to exchange messages. At the end of the round users will be stopped from sending messages until the next round begins. If the coordinator fails a mixer in the system will become a

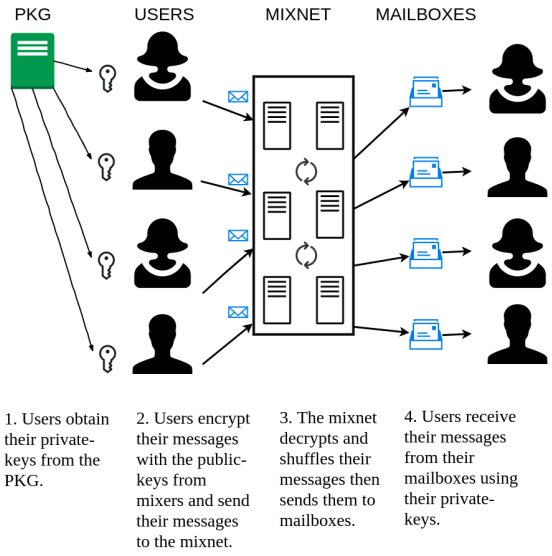

Fig. 1. An overview of Zephyr's protocol.

coordinator. Once the round is over and the original coordinator recovers, the mixer and the coordinator will switch back to their original roles.

All processes in Zephyr will take place in multiple rounds managed by the coordinator. In each round, users will generate the private-keys from the PKG, select a mailbox address that is computed as a function of the recipient's email ID, and select a number of mixers at random. Users will add the mailbox address to the end of the message encrypt it for the recipient and encrypt again for each mixer is has selected, forming multiple layers of encryption.

After all the users have sent their messages to the mix network, each mixer will decrypt and shuffle their messages. The final mixer to decrypt the message will send the message to its public mailbox. Finally, at the end of the round users will download all the messages from their public mailboxes and attempt to decrypt each one, storing the messages that are successfully decrypted.

Zephyr's anonymity mostly depends on how each user selects their mixer nodes due to traffic analysis attacks. In order to combat this, Zephyr requires its users to follow a specific algorithm to select mixers. We will take a look at how Zephyr does this in Section 5.5.

5 Design

We will now explain the details of Zephyr and how it protects against a passive adversary that is monitoring all of its traffic in one of its participating countries.

5.1 Private Key Generators

Traditional key distribution servers leak data about who a user is communicating with based on the public-key the user receives from the server. In contrast, Zephyr uses PKGs, which can compute the public-key of each user using IBE. All a user needs to know is her recipient's email address, which she can then use to generate her recipient's public-key.

Each user in Zephyr receives her private-key from the PKG, however, she must verify her identity first before receiving her private-key. In order to verify her identity, the PKG will send a unique code to the user's email. The user must send the code back to the PKG to verify her identity.

5.2 Mixnet

Zephyr exploits a mixnet to protect the contents of messages. In this section, we will describe its processes.

Communication In order to have reliable communication between mixers in the network, Zephyr makes use of a lightweight in-memory distributed hash table (DHT). Mixers use the DHT in order to send status updates to other mixers. Each message sent to the hash table will automatically be broadcasted to every node in the network, thus, increasing the reliability of the mixers. Zephyr's mixers tell other mixers they are ready to begin the round by sending a put request to the key "ready". Since Zephyr's DHT supports keys with multiple values, every mixer will download all the values under the key "ready". Once the number of values returned is equal to the number of mixers in the network, the protocol will start. The DHT Zephyr exploits is very lightweight on system resources, making it suitable for low-performance computing devices. To send and receive messages from other mixers Zephyr uses remote procedure calls.

Since Zephyr's mixers are lightweight, they are suitable to run on cell phones, laptops, and other low-performance computing devices. However, most of these devices are behind a NAT, which makes them unable to communicate with other mixers behind a NAT. In order for Zephyr to exploit these devices, Zephyr traverses the NAT by using the Interactive Connectivity Establishment protocol [13].

Processing Messages After all the mixers are set up, each mixer generates a public/private key pair for authenticated encryption and puts their public-key in the DHT. Users will later download these public-keys and use them to encrypt their messages with multiple layers of encryption.

Once mixers start receiving messages, they will decrypt each message, shuffle them, and send them to the next mixer. Messages are shuffled using the Fisher-Yates shuffle [7]. This shuffling algorithm was chosen because it effectively creates a random permutation from a finite sequence very quickly. Each permutation generated is equally likely.

Algorithm 1 Mixer protocol

A mixer takes a set of ciphertexts C as input.

1. **Decrypt**: Each mixer s receives ciphertexts $(C_1, \ldots, C\text{n})$ from the previous mixer. In order, s decrypts the ciphertexts with its private-key.
   If s is the last mixer in the selected route, then it sends the ciphertext C to the recipient's public mailbox. Otherwise, it sends them to the next mixer.
2. **Shuffle**: s shuffles randomly $(C_1, \ldots, C\text{n})$, and sends each C to its next mixer.

5.3 Information-nodes

Zephyr uses information-nodes to store the public-keys of mixers and make them available to users. Information-nodes store the mixers' public-keys using a DHT. Once all the mixers in the network are ready one mixer will be selected at random to send all the public-keys to an information-node. That information-node will then put it in its DHT to share the mixers' public-keys with every other information-node.

The decision to include information-nodes in Zephyr's design is a trade-off between bandwidth and CPU usage. Information-nodes provide key distribution for mixer public-keys, but if mixers used IBE instead of authenticated encryption, information-nodes would not be needed. All that would be needed to obtain a mixer's public-key is a mixer's unique ID, which could be its IP address. However, IBE uses more CPU resources in comparison to authenticated encryption. So, to save CPU resources Zephyr exploits information-nodes that consume 32 bytes of bandwidth for each mixer public-key.

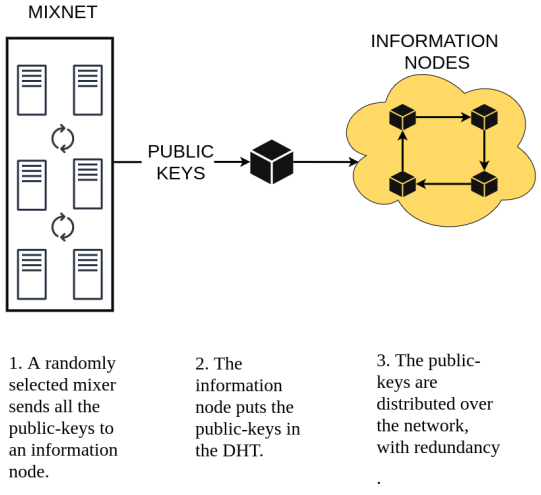

Fig. 2. Overview of information-node setup.

5.4 Mailboxes

Mailboxes are the final destination for user messages. A user's mailbox is determined by the length of characters in the user's email address modulo the number of mailboxes. At the end of each round, the user will download every ciphertext from the mailbox and attempt to decrypt each one, storing the successfully decrypted messages.

Zephyr's mailboxes are SQL databases. The database will have a specific column for each mailbox and users will download their whole column. Mailboxes can also have their SQL databases stored in memory, depending on the amount of RAM of the system, to increase read and write speeds. In certain circumstances, Zephyr might store the database in memory, for example, there are a few clients but a lot of available memory.

5.5 Users

We will now describe how users operate in Zephyr.

Obtaining Keys Figure 3 shows the pseudocode for the Zephry user library. Before a user can create messages she must obtain her private-key from a PKG. To obtain the private-key the user will use initialize() to authenticate herself by sending back a unique code from the PKG, which can be found in her email. Once the user has been authenticated, she will then obtain every mixer public-key from the mixnet by sending a request to an information-node. Users will be required to do this every round of Zephyr's protocol since the mixer keys and PKG keys will be regenerated.

Selecting Mixers To send messages users must select mixers. How these mixers are selected is very important. If they are not selected with proper care, the identity of a user may be revealed through traffic analysis attacks. To make sure users select mixers that will protect their identity, users will have to select mixers from two groups. One group will contain mixers that are hosted overseas and another group will contain mixers hosted in the user's country. Users will generate two random permutations to shuffle the group of mixers from its country and the group of mixers overseas. After shuffling the mixers, the user will select one overseas mixer at random. Only one overseas mixer is selected because that is all that is needed to protect a user's identity, furthermore, selecting one overseas mixer helps to decrease latency. Mixers will be selected with every call to send_message().

Sending Messages After all the mixers have been selected, the user will then prepare to send a message. First, the user will encrypt the plain text message with the recipient's public-key pk_r and append the recipient's mailbox address to the end of the ciphertext. Next, the user will form layers of encryption by encrypting the ciphertext with the public-key of the last mixer pk_{mn}, adding

the address a_n of the last mixer and encrypting that ciphertext with the public-key pk_{mn-1} of the mixer before it, adding the address of the second to last mixer a_{n-1} and encrypting it with the public-key of third to last mixer pk_{mn-3} and adding the address of that mixer and so on.

The resulting ciphertext will look something like this:

$$a_{m1}, pk_{m1}(a_{m2}, pk_{m2}(mailbox, pk_r(message)))$$

Finally, the user will send her message to the mixnet.

```
void initialize(string email,
                string filepath){
    byte_string_t user_pk;
    params_t params;
    // Load the configuration file
    auto vec = get_config_info(filepath);
    // Authentication with PKG
    auto x = get_keys_from_pkg(vec[2][0],
    to_string(8080), email);
    // Store PKG parameters and private-key
    string key_serial_tmp = move(x[0]);
    string param_serial_tmp = move(x[1]);
    deserialize(key_serial_tmp, user_pk);
    deserialize(param_serial_tmp, params);
 }

string encrypt(string msg, params_t params){
    string ciphertext = pkg_encrypt(email,
    params, msg);
    return ciphertext;
}

void send_message(string ciphertext){
    auto mixer_keys = get_mixer_keys();
    auto to_send =
    create_ciphertext(mixer_keys,ciphertext);
    attach_to_mixer(to_send);
}

void recieve_message(){
    auto mailbox_addr = get_mailbox(email);
    auto msgs = download_from(mailbox_addr);
    store(msgs);
}
```

Fig. 3. Pseudocode for user operations

Receiving Messages To receive messages a user will download all of her messages from her public mailbox by using recieve_message(). Once they are downloaded, she will attempt to decrypt each one and save the messages she has decrypted successfully.

6 Implementation and applications

To evaluate Zephyr, we implemented it in approximately 4,000 lines of C++ code. The source code will be available at https://github.com/MutexUnlocked/congenial-zephyr. We used the libsodium library [4] for authenticated encryption and Stanford's IBE Secure Email library [5] for identity-based encryption. To communicate servers used gRPC and TCP. We will now highlight two situations where Zephyr would be very useful.

Communication for Multinational corporations Multinational corporations have offices in overseas countries, which means that a lot of their network communications will take place behind different internet service providers. These conditions enable Zephyr to be a full-strength because access to overseas servers is not an issue since a multinational corporation can run Zephyr with its own servers. Users will send a message selecting one mixer from any of the overseas countries and selecting the rest from mixers hosted in their country. Essentially, the corporation will be able to rely on its own network infrastructure to secure its own messages.

Decentralized Messaging Zephyr can be deployed as a peer-to-peer messaging system. Using this deployment option, no central authority will have control over Zephyr. Furthermore, Zephyr will be able to leverage underutilized computing resources from all over the internet as long as the owners of the computing resources agree to let Zephyr exploit them. Zephyr's design provides incentives for people to add computing resources when it's decentralized. For example, to protect itself from traffic analysis Zephyr must have overseas nodes. If two parties from overseas countries using Zephyr combine their computing resources, they will be better off than using Zephyr in just their own country. Parties can also reduce message download times by using each other's mailbox resources; the more mailboxes the less bandwidth a user will have to use to download messages.

There are two differences between Zephyr's implementation and Zephyr's design described in this paper. First, instead of downloading a list of nodes, each node already has a list stored on its disk. Second, each node was not behind a NAT so there was no need to traverse it.

7 Evaluation

Our evaluation quantitatively answers the following questions:

- Can mixers be run on low-performance computers with acceptable performance?
- Can Zephyr run with acceptable performance overall?

7.1 Experimental Setup

To answer the above questions, we ran a series of experiments on a laptop and Raspberry Pi. The laptop that was used to run Zephyr had an Intel 8250U CPU with 4 cores, 12 GB of RAM, and 100 Mbs of network bandwidth. The laptop runs Linux version 5.0.9. The Raspberry Pi that was used was a Raspberry Pi 4 Model B with a Cortex-A72 (ARM v8) 64-bit SoC with 4 cores, 2 GB of RAM, and 100 Mbs of network bandwidth. The Raspberry Pi runs Linux Version 4.19.

We use the following parameters across our experiments. Zephyr consisted of 5 mixers, 1 mailbox, 2 information-nodes, and 1 PKG each corresponding to one docker container. One mixer is hosted in London and the other 4 mixers are distributed evenly on the laptop and the Raspberry Pi. Users are required to use all 5 mixers in their selection. To ensure that clients would operate without interruptions, clients were simulated by running them in parallel in a separate docker container. Each user's message has a fixed size of 28 bytes. The PKG and information-nodes also run on the laptop in separate docker containers.

7.2 User performance

Deploying Zephyr requires the individual(s)/organization(s) using it to decide the ratio of users to mailboxes. This consideration is a trade-off between user bandwidth and latency: for a fixed number of users, more mailboxes decreases user bandwidth but requires more computing resources (the rest of this section quantifies this trade-off). We expect that Zephyr will be used in a setting that does not require low latency messaging, so a large number of mailboxes will not be needed. In this setting, Zephyr is running with very few computing resources.

Latency. The crucial parameters that affect latency are the number of mailboxes and the number of users in Zephyr. The more users there are the more bandwidth each user will have to use to download her mailbox. This is because the number of users is evenly distributed among all mailboxes.

CPU. We measured the CPU usage for users decrypting messages from their mailboxes; this is where most of a user's CPU resources are utilized. The number of ciphertexts that can be decrypted per second per core is given for both the laptop and Raspberry Pi in the table below.

Table 1. User decryption performance

Machine	Decryptions per second per core
Raspberry Pi 4 Model B	76
Intel 8250u Laptop	531

Using a single CPU core, it will take the Raspberry Pi 2.6 seconds to decrypt a mailbox of 200 messages and a laptop with an Intel 8250U CPU 0.3 seconds.

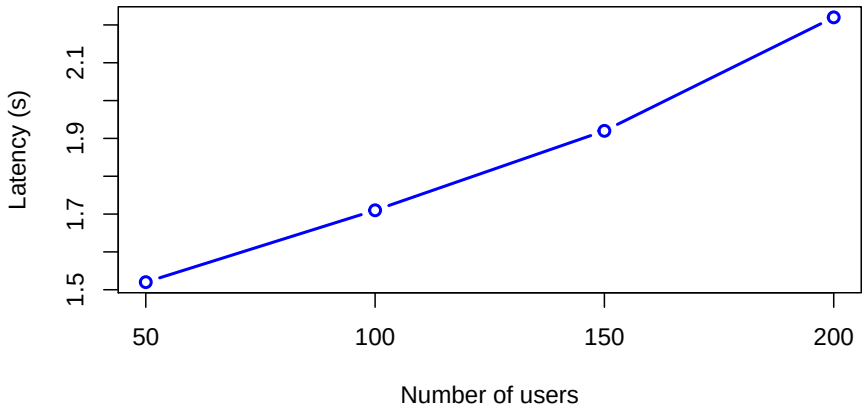

Fig. 4. User latency based on the number of users in Zephyr.

Bandwidth. Figure 5 shows the total user-side bandwidth requirement as a function of the number of users in Zephyr. The bandwidth is mostly spent on downloading messages from mailboxes. In Figure 5, with two hundred users, the mailbox contained 200 messages from users; each user sent one message.

The number of mailboxes in Zephyr will have to increase as the number of users increase to maintain acceptable performance. This is not a problem because Zephyr can easily be decentralized: any server can join the network as long as it is included in Zephyr's configuration file, which can be possible if the individual(s)/organization(s) using Zephyr agree. This allows the number of computing resources to increase.

In our experiment, Zephyr was still able to achieve acceptable performance with limited computing resources. Furthermore, the bandwidth usage for 200 users was still under 2000 KB (2 MB); It would only take 1 microsecond for a user using a 4G phone to download her messages.

7.3 Server performance

To evaluate whether Zephyr can achieve acceptable performance with only a few computing resources, we measured each variable that has significant effects on Zephyr's overall performance. Specifically, we measured the time it takes to decrypt messages encrypted with IBE or authenticated encryption, the memory usage of mixers, and the time it takes for an information-node to process a user's request. we also measured the request processing time of the PKG that generates the users' private-keys.

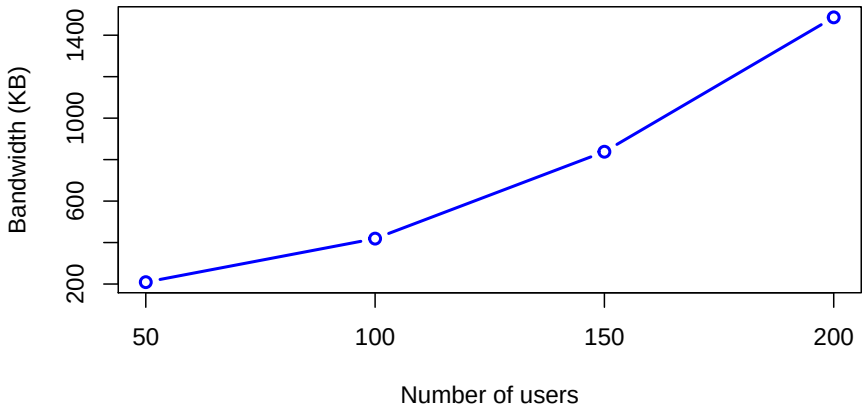

Fig. 5. User bandwidth usage based on the number of users in Zephyr.

7.4 Mixer performance

Memory. With only 5 mixers each used less than 10 MiB of RAM to process messages. The amount of RAM needed is not directly proportional to the number of users in the network. This is good for Zephyr because Zephyr's mixers should be able to run on devices with few computing resources, such as very few gigabytes of RAM. The Raspberry Pi running as a mixer would still have plenty of available memory left for other processes in the background.

CPU. We measured the CPU usage for decrypting messages. This is where most of a mixer's CPU resources are utilized. Decryption was only allowed to one core in this experiment, although it could have been easily parallelized with more cores. The number of ciphertexts that can be decrypted per second per core is given for both the laptop and Raspberry Pi in the table below.

Table 2. Mixer decryption performance

Machine	Decryptions per second per core
Raspberry Pi 4 Model B	393
Intel 8250u Laptop	1000

Compared to the IBE algorithm it takes the authenticated encryption algorithm a lot less time to encrypt and decrypt messages. Using a single CPU core, it will only take the Raspberry Pi running as a mixer a tenth of a second to

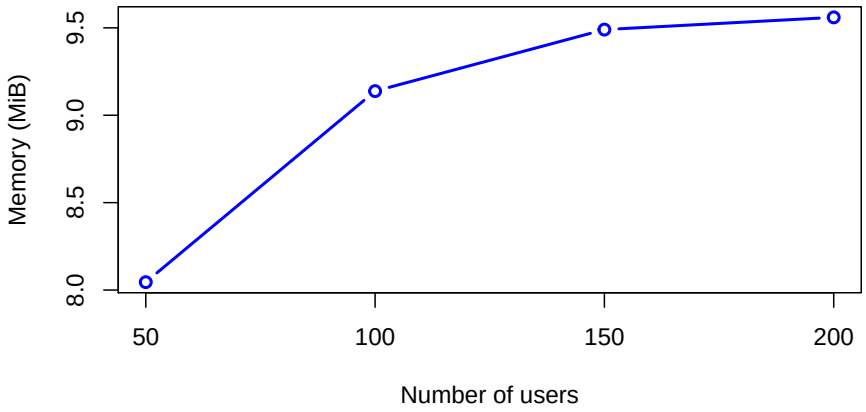

Fig. 6. Memory usage per mixer based on users.

decrypt 200 messages and a laptop with an Intel 8250U CPU running as a mixer 9 milliseconds.

7.5 PKG performance

A PKG has to process every user's request for her private-key every round, which has a dramatic effect on the user's latency. A laptop running with an Intel 8250u CPU can handle 24 user requests per second.

7.6 Information-node performance

Zephyr's information-nodes can send every mixer public-key to users incredibly fast. With only 5 mixers and 200 users, the average user was able to receive all the mixer public-keys in only 0.17 seconds using 160 bytes of bandwidth.

8 Conclusion

Zephyr makes private messaging over internet-related services easier to implement and does not rely on regular end-to-end encryption messaging systems, which are prone to man-in-the-middle attacks. Zephyr hides users' identities by using private-key generators, information-nodes, mailboxes, and a mixnet. Users that use Zephyr do not reveal who they are talking to or the contents of their messages.

9 Discussion and Limitations

Mailbox Decryption For users to find their messages from mailboxes faster, Zephyr needs to be deployed with more mailboxes as the number of users grows to decrease the number of messages in each mailbox. This is because IBE is too slow for decrypting messages; a faster algorithm would not require the number of mailboxes per group of users to grow as quickly. However, IBE cannot be removed from Zephyr because that would mean Zephyr would have to rely on key distribution techniques that would potentially leak a user's identity. To address this, Zephyr's future design could include mailboxes for various sizes of messages. Smaller messages go to one mailbox and larger messages go to another mailbox. Users that receive larger messages would consume more bandwidth and CPU resources without placing the same burden on users that only want to receive smaller messages.

Scalability Scalability is not a major design consideration in Zephyr. However, Zephyr can be horizontally scaled. Horizontally scaling Zephyr is not very expensive. For example, Zephyr can be horizontally scaled by adding multiple Raspberry Pi nodes, which cost no more than $40. Furthermore, Zephyr has built-in NAT traversal for mixers, which makes it very easy to decentralize. Zephyr can use computing resources from all over the world as long as the individual(s)/corporation(s) using Zephyr agree to include them.

References

1. Matrix specification, https://matrix.org/docs/spec/
2. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. vol. 32, pp. 213–229 (08 2001). https://doi.org/10.1007/3-540-44647-8$_1$3
3. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–90 (Feb 1981). https://doi.org/10.1145/358549.358563, https://doi.org/10.1145/358549.358563
4. Denis, F.: The sodium cryptography library (Jun 2013), https://download.libsodium.org/doc/
5. Group, S.U.A.C.: Ibe secure e-mail (Apr 2002), https://crypto.stanford.edu/ibe/
6. van den Hooff, J., Lazar, D., Zaharia, M., Zeldovich, N.: Vuvuzela: Scalable private messaging resistant to traffic analysis. In: Proceedings of the 25th Symposium on Operating Systems Principles. p. 137–152. SOSP '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2815400.2815417, https://doi.org/10.1145/2815400.2815417
7. Knuth, D.E.: The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms. Addison-Wesley Longman Publishing Co., Inc., USA (1997)
8. Kwon, A., Corrigan-Gibbs, H., Devadas, S., Ford, B.: Atom: Horizontally scaling strong anonymity. In: Proceedings of the 26th Symposium on Operating Systems Principles. p. 406–422. SOSP '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3132747.3132755, https://doi.org/10.1145/3132747.3132755
9. Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle: An efficient communication system with strong anonymity. PoPETs **2016**, 115–134 (2015)

10. Li, F., McMillin, B.: Chapter two - a survey on zero-knowledge proofs. Advances in Computers, vol. 94, pp. 25 – 69. Elsevier (2014). https://doi.org/https://doi.org/10.1016/B978-0-12-800161-5.00002-5, http://www.sciencedirect.com/science/article/pii/B9780128001615000025
11. Manils, P., Abdelberi, C., Blond, S., Kaafar, M.A., Castelluccia, C., Legout, A., Dabbous, W.: Compromising tor anonymity exploiting p2p information leakage (07 2010)
12. Mathewson, N., Dingledine, R.: Practical traffic analysis: Extending and resisting statistical disclosure (06 2004). https://doi.org/10.1007/11423409$_2$
13. Rosenberg, J.: Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245 (Apr 2010). https://doi.org/10.17487/RFC5245, https://rfc-editor.org/rfc/rfc5245.txt
14. Shirazi, F., Simeonovski, M., Asghar, M.R., Backes, M., Diaz, C.: A survey on routing in anonymous communication protocols. ACM Comput. Surv. **51**(3) (Jun 2018). https://doi.org/10.1145/3182658, https://doi.org/10.1145/3182658
15. Systems, O.W.: Open whisper systems partners with whatsapp to provide end-to-end encryption (Nov 2014), https://signal.org/blog/whatsapp/
16. Wright, M.K., Adler, M., Levine, B.N., Shields, C.: The predecessor attack: An analysis of a threat to anonymous communications systems. ACM Trans. Inf. Syst. Secur. **7**(4), 489–522 (Nov 2004). https://doi.org/10.1145/1042031.1042032, https://doi.org/10.1145/1042031.1042032