Scalable Private Messaging Resistant to Traffic Analysis

Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich
MIT CSAIL

Abstract

Private messaging over the Internet has proven challenging to implement, because even if message data is encrypted, it is difficult to hide metadata about *who* is communicating, in the face of traffic analysis. Systems that offer strong privacy guarantees, such as Dissent [32], scale to only several thousand clients, because they use techniques with superlinear cost in the number of clients (*e.g.,* each client broadcasts their message to all other clients). On the other hand, scalable systems, such as Tor, do not protect against traffic analysis, making them ineffective in an era of pervasive network monitoring.

Vuvuzela is a new scalable messaging system that offers strong privacy guarantees, hiding both message data and metadata. Vuvuzela is secure against adversaries that observe and tamper with all network traffic, and that control all nodes except for one server. Vuvuzela's key insight is to minimize the number of variables observable by the attacker, and to use differential privacy techniques to add noise to all observed variables in a way that provably hides information about which users are communicating. Vuvuzela has a linear cost in the number of clients, and experiments show that it can achieve a throughput of 30,000 messages per second for 2 million users with a 1-minute latency on commodity servers.

1 Introduction

Many users would like their communications over the Internet to be private, and for some, such as reporters, lawyers, or whistleblowers, privacy is of paramount concern. Encryption software can hide the *content* of messages, but adversaries can still learn a lot from *metadata*—which users are communicating, at what times they communicate, and so on—by observing message headers or performing traffic analysis. For example, if Bob repeatedly emails a therapist, an adversary might reasonably infer that he is a patient, or if a reporter is communicating with a government employee, that employee might come under suspicion. Recently, officials at the NSA have even stated that "if you have enough metadata you don't really need content" [29: ¶7] and that "we kill people based on metadata" [20]. This suggests that protecting metadata in communication is critical to achieving privacy.

Unfortunately, state-of-the-art private messaging systems are unable to protect metadata for large numbers of users. Existing work falls into two broad categories. On the one hand are systems that provide strong, provable privacy guarantees, such as Dissent [32] and Riposte [10]. Although these systems can protect metadata, they either rely on broadcasting all messages to all users, or use computationally expensive cryptographic constructions such as Private Information Retrieval (PIR) to trade off computation for bandwidth [30]. As a result, these systems have scaled to just 5,000 users [32] or hundreds of messages per second [10].

On the other hand, scalable systems like Tor [14] and mixnets [7] provide little protection against powerful adversaries that can observe and tamper with network traffic. These systems require a large number of users to provide any degree of privacy, so as to increase the anonymity set for each user, but even then are susceptible to traffic analysis. Adding cover traffic to try to obscure which pairs of users are communicating has been shown to be expensive and to yield only limited protection against a passive observer over time [12, 23], while adversaries that can actively disrupt traffic (*e.g.,* add delays) gain even more information [1].

This paper presents Vuvuzela, a system that provides scalable private point-to-point text messaging. Vuvuzela ensures that no adversary will learn which pairs of users are communicating, as long as just one out of N servers is not compromised, even for users who continue to use Vuvuzela for years.[1] Vuvuzela uses only simple, fast cryptographic primitives, and, using commodity servers, can scale to millions of users and tens of thousands of messages per second. At the same time, Vuvuzela can provide guarantees at a small scale, without the need for a large anonymity set: even if just two users are using the system, an adversary will not be able to tell whether the two users are talking to each other.

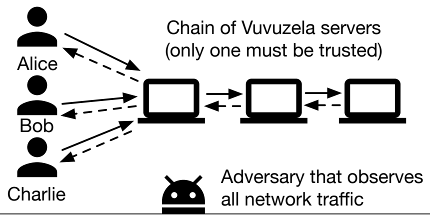

Figure 1: Vuvuzela's overall architecture. The Vuvuzela network consists of a chain of servers, at least one which is assumed to be trustworthy.

Vuvuzela works by routing user messages through a chain of servers, as shown in Figure 1, where each of the servers adds cover traffic to mask the communication patterns of users. Unlike prior systems, Vuvuzela's design enables cover traffic to scale to millions of users, and allows us to prove strong

[1] Vuvuzela cannot hide the fact that a user is connected to Vuvuzela's network, but we expect that users will simply run the Vuvuzela client in the background at all times to avoid revealing the timing of their conversations.

guarantees about the level of privacy provided by cover traffic. We achieve this using two key techniques.

First, Vuvuzela's protocols are carefully structured to reveal only a small, well-defined set of observable variables to an adversary. For instance, Vuvuzela's communication protocol, used for sending user messages, exposes just two variables: the total number of users engaged in a conversation, and the total number of users not engaged in a conversation. It does *not* reveal the users in either group. This is significantly smaller than the number of variables exposed by previous systems, and enables Vuvuzela to focus on minimizing the useful information that an adversary can learn from these variables.

Second, Vuvuzela adopts ideas from differential privacy [16] to state precise privacy guarantees, and to bound information leakage over time by adding noise to the observable variables with cover traffic. Vuvuzela ensures that any observation that an adversary can perform will be "almost independent" of whether some user is active or not,[2] which means that the adversary cannot learn who, if anyone, a user is talking to. Somewhat counter-intuitively, results from differential privacy show that the amount of cover traffic needed is constant—independent of the number of users or messages—and we find that the amount is manageable in practice. Adding noise to achieve differential privacy is tractable for the small number of variables exposed by Vuvuzela, but it was not feasible for prior systems that expose many distinct variables.

Vuvuzela's design applies these techniques to build a complete messaging system that uses two protocols: an efficient point-to-point *conversation* protocol, for exchanging messages between users that have agreed to communicate, and a more expensive *dialing* protocol for starting conversations. In terms of fault tolerance, Vuvuzela is resilient to misbehaving clients, but an adversarial server can block all communications (which is unavoidable in our threat model of an adversary that can tamper with all network traffic).

To evaluate Vuvuzela, we implemented a prototype in Go on several commodity servers (36-core VMs on EC2). Like many prior systems, Vuvuzela operates in *rounds* to avoid leaking fine-grained timing information. Results show that Vuvuzela can support 2 million users with each round incurring a latency of 1 minute. In this configuration, Vuvuzela lets each user exchange 500,000 messages over time, with any set of partners, without risking discovery. The cover traffic needed for this level of security is equivalent to only about 300,000 active users, which is less than a sixth of the real user traffic, and is easy to handle in practice. Finally, we show that sharding each Vuvuzela server across multiple machines achieves near-linear speedups, enabling Vuvuzela to scale to even larger workloads.

In summary, the contributions of this paper are:

[2]More precisely, the probability of an observation when the user is active is at most $e^{\epsilon}\times$ the probability of the same observation when the user is inactive plus δ, for some small ϵ and δ.

- A new approach to hiding metadata in messaging systems, by minimizing the number of observable variables and applying differential privacy techniques.
- The design of Vuvuzela, the first private messaging system that can hide metadata while scaling to 2 million conversing users (which is about 100× higher than prior systems).
- An evaluation of Vuvuzela's scalability and security on commodity servers.

2 Motivation and Goals

Vuvuzela's goal is to provide point-to-point messaging between users in a way that is *private* in the face of a strong adversary that can observe and tamper with the entire network and all but one of Vuvuzela's servers. That is, an adversary should not be able to distinguish between a scenario where two particular users are communicating, and a scenario where they are not, even after interfering with the network over many rounds. This section lays out the motivation behind Vuvuzela's design, and Vuvuzela's threat model and security properties.

2.1 Motivation

To get a better sense of why privacy is hard to provide under Vuvuzela's strong adversary model, consider a system with only *one* server, which is fully trusted. Even in this system, achieving privacy is nontrivial. For example, suppose that each round, the server first collects messages from all clients that wish to send a message, and then sends each message to its recipient. Although the adversary cannot immediately tell which sender was responsible for a given recipient's message, she can still discover relevant information. For instance, if the adversary suspects that Alice and Bob are communicating, she can temporarily block network traffic from Alice, and see whether Bob stops receiving messages. Or, in our model of a strong adversary, she can block traffic from *all* clients except for Alice and Bob, and see whether any messages got exchanged when just Alice and Bob were connected.

As discussed in §1, previous systems handle this problem using mechanisms that limit scaling, such as broadcasting all messages to all users [10, 32] or cryptographic schemes like PIR [30]. As a result, these systems are limited to a few thousand users or a few hundred messages per second.

In fact, due to the attacks above, any system that reveals *some* information about the number of messages exchanged is vulnerable to our adversary over many rounds, because she can use attacks like blocking one of Alice and Bob and seeing how that changes the number of exchanged messages. Furthermore, in Vuvuzela, unlike in the simpler setting above, we must protect not only against a network adversary but all but one of the servers being compromised. Our security goals take this into account to protect each user over many rounds.

2.2 Security goals

Informally, the security definition we want is the following: for each user Alice, the adversary *should not be able to distinguish*

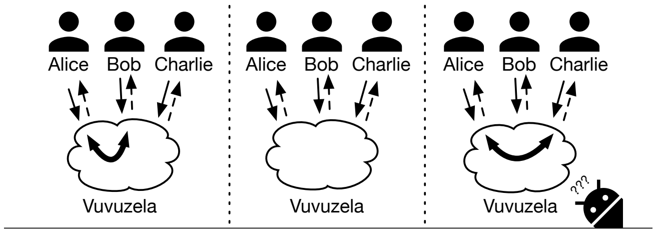

Figure 2: Vuvuzela's security goal. An observer capable of seeing all network traffic must not be able to distinguish between various possible worlds. In one world, Alice is communicating through Vuvuzela with Bob. In another, she is connected to the system but not exchanging messages with other users. In a third, she is communicating with Charlie. Vuvuzela gives Alice differential privacy: any event observed by the adversary has roughly equal probability in all worlds.

between any of Alice's possible communication patterns, even after Alice exchanges messages in a large number of rounds.

We make this definition precise using differential privacy [15]. Differential privacy says that for any observation X that the adversary might make of the system, the probability of observing X should be similar regardless of Alice's communication pattern, as shown in Figure 2.

Formally, differential privacy is defined as follows:

Definition 1. *A randomized algorithm M is (ε, δ)-differentially private if, for all sets of outcomes S, and all adjacent inputs x and y,* $\Pr[M(x) \in S] \leq e^{\varepsilon} \Pr[M(y) \in S] + \delta$.

In our case, the inputs x and y are the user actions for every round (*i.e.,* which users are communicating in each of the rounds), and we consider two inputs *adjacent* if they differ in only the message exchanges by one user, Alice, in up to k rounds. The function M represents the observations made by the adversary after she performs whatever manipulation she wishes of the system. k, ε, and δ will be set to grant a high degree of protection even if Alice uses Vuvuzela to exchange hundreds of thousands of messages.

Note that this definition covers *all* information that an adversary might learn about Alice's communications: not only whether she's talking to a specific user Bob, but whether she's communicating at all. As long as Alice exchanges fewer than k messages, the adversary cannot distinguish her from a user that exchanges *no* messages and is just connected to the system. This formalization subsumes most other privacy guarantees that Alice might want in practice: distinguishing whether she is talking to Bob or Charlie, distinguishing whether she has ever talked to a specific 100-person group, etc.

2.3 Threat model

Vuvuzela's design assumes an adversary that controls all but one of the Vuvuzela servers (users need not know which one), controls an arbitrary number of clients, and can monitor, block, delay, or inject traffic on any network link. Two users, Alice and Bob, communicating through Vuvuzela should have their communication protected if their two clients, and any one server, are uncompromised. Since users will communicate over multiple rounds, we assume that the adversary may also monitor and interfere with them over multiple rounds.

Our cryptographic assumptions are standard. We assume secure public and symmetric key encryption, key-exchange mechanisms, signature schemes and hash functions. We also assume that the Vuvuzela servers' public keys are known to all users, and that two users who wish to communicate know each other's public keys. Separate mechanisms are needed to let users discover each other's keys, but we consider these orthogonal to the private communication problem in this paper, which is difficult even with these assumptions.

3 Conversation Protocol

As described in §1, Vuvuzela uses two main protocols. The first protocol, called the *conversation* protocol, allows a pair of users to exchange messages, assuming that they both agreed to communicate with one another. The second protocol, called *dialing*, enables one user request a conversation with another.

This section describes Vuvuzela's conversation protocol, starting from a simple strawman design and evolving it to the complete protocol. We specifically focus on how the protocol achieves strong privacy guarantees by following the two steps outlined in §1: minimizing the number of variables visible to an adversary and adding cover traffic to hide any information that can be obtained from the few remaining variables.

3.1 Overview

Vuvuzela consists of a chain of *servers* to which *clients* connect to communicate. The conversation protocol proceeds in *rounds*. In each round, each client may exchange a message with another client, or may send and receive fake messages that hide the fact that it is inactive (these are distinct from Vuvuzela's cover traffic, which will be described in §3.4). The first server in Vuvuzela's chain is responsible for coordinating the round, by announcing the start of a round to clients and waiting a fixed amount of time for clients to submit messages.

Every Vuvuzela client can independently decide to join and leave at any round. However, we suggest that users run the Vuvuzela client whenever their computer is connected to the Internet. This minimizes the amount of information disclosed by the fact that the Vuvuzela client is active on a user's machine.

Conversations in Vuvuzela happen through the concept of *mailboxes*. The last server in the Vuvuzela chain maintains a set of mailboxes, and clients send requests through the chain to access these mailboxes, with communicating clients agreeing to use the same mailboxes. As we explain, the concept of mailboxes greatly reduces the number of observable variables, by making most mailboxes indistinguishable to an adversary.

Finally, the Vuvuzela servers each perform some actions to hide client activity: each server shuffles the operations it receives into a random order to unlink them from clients, and each server adds cover traffic to hide *how many* messages are

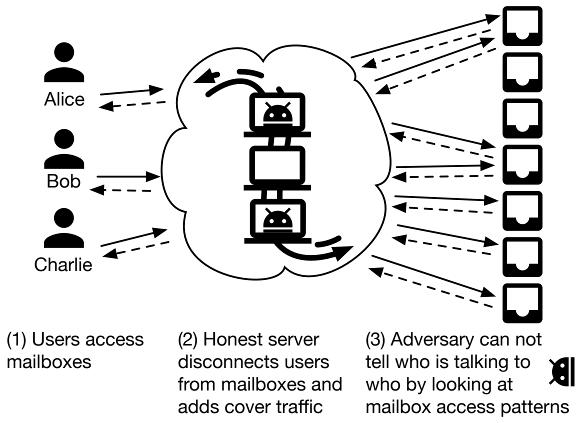
Alice
Bob
Charlie
(1) Users access mailboxes
(2) Honest server disconnects users from mailboxes and adds cover traffic
(3) Adversary can not tell who is talking to who by looking at mailbox access patterns

Figure 3: Overview of Vuvuzela's conversation protocol.

actually exchanged each round. Figure 3 shows a sketch of the whole conversation protocol.

In the rest of this section, we explain the protocol in detail by starting with a simple strawman design that does *not* provide security, then identifying all the variables revealed to an attacker, and finally describing how to hide them.

3.2 A simple strawman protocol

To control the number of observable variables, Vuvuzela does not let users communicate directly, but instead uses a mailbox design, so that Vuvuzela servers never have to initiate connections back to a user. Figure 4 illustrates a strawman mailbox-based design. The way two users communicate in this protocol is reminiscent of a "dead drop." Users pick some mailbox as a meeting spot, and perform an *exchange* operation on that mailbox. The exchange operation places a new message in a mailbox, and retrieves whatever message was placed there by another user. If two users perform an exchange on the same mailbox, they receive each others' messages. Thus, if Alice and Bob want to communicate, each of them exchanges the message they want to send (if any) with the mailbox, and each will receive the other's message as a result.

The protocol supports conversations between many pairs of users, where each pair has a shared secret. The shared secret is used to encrypt messages, and to determine what mailbox ID to use. The protocol takes place in rounds, and in each round, each user can send and receive one message, by exchanging a message through a single mailbox. To hold conversations, users participate in many rounds. Mailboxes are not assigned to users, but since mailboxes are named by long pseudo-random IDs, it is exceedingly unlikely that two users access the same mailbox unless they share the same key.

Users exchange messages by asking the Vuvuzela servers to perform the exchange for them. At the beginning of the round, each user decides what message to send, then sends a request to exchange that message with the correct mailbox. After receiving all requests for a round, the servers execute them, and send back the results to the users. This is how two users, such as Alice and Bob in Figure 4, can communicate.

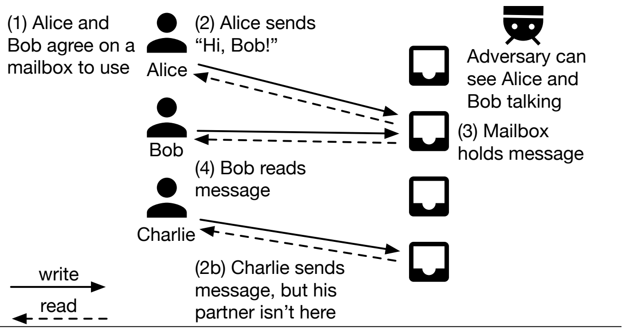
(1) Alice and Bob agree on a mailbox to use
Alice
(2) Alice sends "Hi, Bob!"
Adversary can see Alice and Bob talking
Bob
(3) Mailbox holds message
(4) Bob reads message
Charlie
write
read
(2b) Charlie sends message, but his partner isn't here

Figure 4: Strawman conversation protocol that does not hide information about which users accessed a given mailbox.

If a user requested an exchange with a mailbox that no other user interacted with, Vuvuzela sends back an empty message.

While it forms the basis of our protocol, this strawman design allows an adversary to observe three sets of variables:

1. Which users participated in the protocol each round.
2. Which users accessed each mailbox (letting the adversary link users to one another).
3. How many messages were successfully exchanged each round (how many mailboxes were accessed by two users).

3.3 Minimizing observable variables

Through careful design of Vuvuzela's protocol, we show that we can hide all but the last variable in the strawman design, and for the last variable, we will obscure it with enough cover traffic to provide a high degree of privacy.

Hiding which users are active. To eliminate the variable of which users are participating in each round, all users always perform an exchange, even if they have no partner. For example, in Figure 4, Charlie performs an exchange with a random mailbox. To make sure that all exchanges look the same to an observer, the last Vuvuzela server returns an empty message when only one request is received for a mailbox, and all messages are padded to the same length and encrypted using an indistinguishable encryption scheme.

Randomizing mailbox IDs. If Alice and Bob were to always use the same mailbox ID, then an observer might correlate the fact that Alice and Bob are online with the fact that a particular mailbox is active. To ensure that the adversary cannot learn anything from the identity of the mailbox accessed in any given round, Vuvuzela clients choose a new mailbox to access for every round, by generating a pseudo-random mailbox ID based on their shared secret. Two users generate this shared secret based on their public keys and the round number.

By using a cryptographically secure pseudo-random number generator, Vuvuzela ensures that an adversary cannot learn any information from the mailbox IDs being accessed at every round, and cannot correlate the mailbox IDs across rounds.

Unlinking users from operations. To eliminate the observable connection between the sender of a message, the mailbox

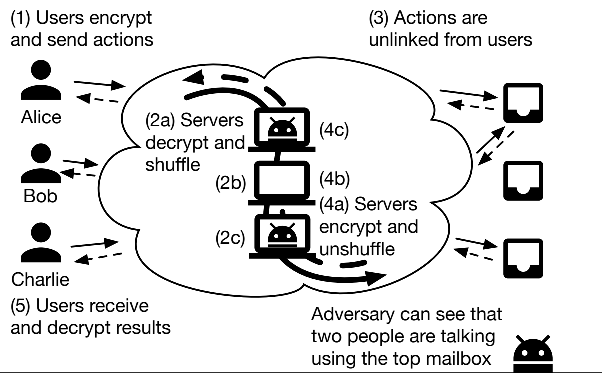
(1) Users encrypt and send actions
(3) Actions are unlinked from users
Alice
Bob
Charlie
(2a) Servers decrypt and shuffle
(2b)
(2c)
(4c)
(4b)
(4a) Servers encrypt and unshuffle
(5) Users receive and decrypt results
Adversary can see that two people are talking using the top mailbox

Figure 5: Vuvuzela's conversation protocol with exchange shuffling.

that the message is placed in, and the eventual recipient of the message, Vuvuzela employs a mixnet-based design, as illustrated in Figure 5. The goal of this mixing phase is to make sure that no observer can determine which user initiated which mailbox exchange. The servers form a chain, and take turns shuffling the exchange operations. After passing through all servers, the server at the end of the chain is responsible for performing the exchanges and computing their results. Then, the results are sent back through the chain, mixed in reverse, and finally delivered back to the original user.

To make sure that exchange operations get properly mixed, each user encrypts their operation with the public key of each server. If there are three servers, with public keys pk_1, pk_2, and pk_3, then a user encrypts their operation o to $E_{\text{pk}_1}(E_{\text{pk}_2}(E_{\text{pk}_3}(o)))$.[3] Within each server's layer of encryption, the user also includes a temporary key for that server to use to encrypt the user's result on the way back, as shown in Algorithm 1. Each server waits for all of the round's operations to come in, decrypts its layer of encryption, and shuffles all the operations with a random permutation.

This provides a strong degree of security. Vuvuzela assumes that at least one mixing server is honest. In that case, even if all other mixing servers are compromised, the adversary cannot correlate the operations going in to the honest server and coming out of that server, as the adversary is assumed not to have the server's private key. Consequently, a user cannot be linked to their mailbox operations after mixing.

After processing the exchanges, the results get passed back through the chain in reverse. Each server encrypts each result with the temporary key that was left for it on the way in, applies the shuffling permutation in reverse, and sends it back to the previous server in the chain, or the original user for the first server in the chain (Algorithm 2). Again, Vuvuzela's assumption of at least one honest server prevents an adversary from linking any result to the corresponding mailbox exchange.

3.4 Hiding the number of messages exchanged

At first glance, it might seem as if the protocol in Figure 5 provides strong privacy guarantees. Every round, each user picks a random mailbox, sends an indistinguishable operation, and the protocol ensures the adversary cannot tie the user to the mailbox that was accessed. How is this protocol insecure?

The problem lies in the one remaining observable variable: the histogram of mailbox access counts. While the mailboxes and their contents are all indistinguishable from our adversary, some mailboxes still look different from the perspective of an observer. For example, in Figure 5, one mailbox is accessed twice in a round, while another mailbox is accessed just once. As we described in §2.1, this can provide valuable information by exposing the number of messages exchanged.

One attack possible in Figure 5 involves the adversary controlling, for example, the first and third Vuvuzela servers. Suppose the adversary wants to know whether Alice and Bob are communicating. To figure this out, she collects operations from all users at the first server, but then throws away all operations except those from Alice and Bob. Then, she passes these operations to the second server. The second server will mix the operations and send them to the third server. If the adversary controls the third server, she can now figure out whether Alice and Bob are talking! Specifically, if Alice and Bob are communicating, the third server will see two exchanges for the same mailbox; otherwise, it will not.

An observer that might not be willing to perform such invasive attacks could still learn a lot from mailbox access patterns. For example, the observer can simply wait for Alice to go offline, and look at the difference in mailbox access patterns between two rounds. If suddenly the number of mailboxes with two exchanges decreases, the adversary can infer that Alice was probably talking to someone in the previous round.

These attacks demonstrate that even a small amount of observable information can be valuable. Luckily, we can completely describe the variables observable to an adversary with two numbers: the number of mailboxes that had one exchange operation, and the number of mailboxes that had two exchange operations.[4] Beyond their number of accesses, the mailboxes are cryptographically indistinguishable.

To hide these last two variables, each server generates *cover traffic* messages that look like accesses to random mailboxes. The server draws two samples, n_1 and n_2, from the distribution $N \sim \max(0, \lceil \text{Laplace}(\mu, b) \rceil)$, and adds n_1 messages that each access a mailbox once, and n_2 pairs of messages that access the same mailbox. It shuffles these messages along with the real ones before passing them to the next server, and removes them when results come back. §5 will show why this distribution of noise is sufficient, and explain how to set μ and b.

Figure 6 shows the complete protocol with cover traffic. Looking at the mailboxes in this figure, it is no longer possible to tell exactly how many users are talking and how many are not. While the cover traffic added hides the exact number of

[3]Of course, o's contents is also encrypted with the recipient's key.

[4]It is possible for a mailbox to get more than two exchanges, if users misconfigure client software, or if an adversary issues many exchanges for a mailbox. However, for mailboxes used by two uncompromised users, this cannot happen: their mailbox number is derived from a shared secret not known to anyone else. Thus, we focus on mailbox access patterns of uncompromised users; the adversary already knows the accesses by compromised users.

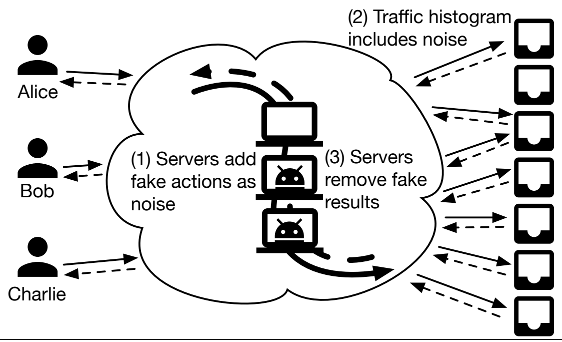
(2) Traffic histogram includes noise
Alice
Bob
Charlie
(1) Servers add fake actions as noise
(3) Servers remove fake results

Figure 6: The complete Vuvuzela conversation protocol. In addition to mixing operations, Vuvuzela servers also add fake operations to hide the number of mailboxes accessed. On the right, it is no longer possible to tell how many users are currently talking, and how many are not.

mailboxes of each type, it still provides the adversary with a rough idea of how many people are talking. This is in line with our security goal. We do not care if the adversary knows roughly how many people are talking; the only thing that matters to us is whether an adversary can tell whether a single individual is talking or not, and that is completely hidden in Vuvuzela's conversation protocol. Specifically, an adversary will no longer learn anything by throwing away all operations except those from Alice and Bob, and an adversary will also no longer learn anything when Alice goes offline, as the cover traffic hides those valuable small changes in the access counts.

Summary. Assuming that at least one server is honest, there is no other information accessible to the adversary. All messages have a constant length, so each user's bandwidth usage is constant. After reaching the honest server, all users' operations are unlinked from the sending user. All users' operations are indistinguishable from random, except for their mailbox numbers. Mailbox numbers are random, but an adversary does know what mailbox each message accesses. Each mailbox will be accessed by either one or two users, and so the total number of mailboxes with each access pattern is all an adversary learns, and we show in §5 that our cover traffic hides access patterns in a differentially private way.

4 Dialing Protocol

Vuvuzela's conversation protocol is useful for pairs of users that have agreed to talk to each other. However, because each user can exchange messages with only one other user per round, users also need a way to start conversations. This is handled by Vuvuzela's dialing protocol. We expect that users run this protocol each time they want to start a conversation with a partner, and possibly many times with the same partner.

In principle, one might want to combine the communication and dialing protocols in order to improve privacy, by making communication and dialing messages indistinguishable from one another. However, as we will show in the rest of this section, dialing has significantly different latency and message size requirements, which led to our decision to expose two distinct protocols in Vuvuzela.

Algorithm 1 Conversation round: client

```
Consider round $r$, when user Alice with public key $pk_{\text{alice}}$ and secret key $sk_{\text{alice}}$ wants to send message $m$ to user Bob with public key $pk_{\text{bob}}$. Each server $i$, ranging from 1 to $n$, has a public key $pk^i_{\text{server}}$.
1. Compute mailbox and encrypt message: Using Diffie-Hellman, compute a shared secret $s_{n+1} = \text{DH}(sk_{\text{alice}}, pk_{\text{bob}})$. The mailbox will be $b = H(s_{n+1}, r)$. Encrypt and pad the message $m$ using nonce $r$ and secret key $s$ to get $e_{n+1} = (b, \text{Enc}(s_{n+1}, m))$.
2. Onion wrap the message and send it to the servers: Alice encrypts the message for each server in the chain. Encryption happens in reverse, from server $n$ to server 1, as server 1 will be the first to decrypt the message. For each server $i$, generate a temporary keypair $(pk_i, sk_i)$. Then, re-encrypt $e_{i+1}$ with $s_i = \text{DH}(sk_i, pk^i_{\text{server}})$ to get $e_i = (pk_i, \text{Enc}(s_i, e_{i+1}))$.
3. Receive the result from the servers and unwrap it: After sending $e_1$ to server 1, Alice gets back $e'_1$. To compute the message $m'$ sent by Bob, Alice decrypts the layers: $e'_{i+1} = \text{Dec}(s_i, e'_i)$. After decrypting $e'_{n+1}$ and unpadding the message, Alice can read the message Bob sent her, if any.
```

4.1 Overview

In Vuvuzela's dialing protocol, a user can send an invitation to talk to another user identified by a long-term public key. The invitation itself consists of the sender's public key. Then, the two users can derive a shared secret from their keys using Diffie-Hellman and use the conversation protocol to chat. The challenge Vuvuzela's dialing protocol addresses is, once again, to reveal as few variables to an observer as possible, and to add the right amount of noise to those variables.

Recall that Vuvuzela's conversation protocol from §3 reveals only the number of conversations that are currently ongoing. That protocol relies on the fact that users have a shared secret they can use to derive random-looking mailbox numbers, and as a result, the identity of a mailbox in each round is meaningless to the adversary.

For dialing, we cannot rely on secret mailbox IDs, because users do not know which other users may wish to dial them. Instead, the dialing protocol uses a number of large *invitation mailboxes*, as shown in Figure 7. Each such mailbox receives all invitations for a fixed set of public keys; with m invitation mailboxes, public key pk's invitations are stored in mailbox $H(pk)$ mod m, where H is a standard cryptographic hash function. Each user downloads all invitations from her mailbox (including fake invitations added as cover traffic) and decrypts them to find invitations meant for her. If she wishes to accept a conversation with a sender, she simply starts the conversation protocol based on that sender's public key.[5]

We wish to hide two sets of variables from an adversary:

1. Given a sender, what mailbox did she add an invitation to?
2. Given a mailbox, how many invitations are in it (since the adversary can link recipients to mailbox IDs)?

[5] To tie the sender's public key to an identity, the Vuvuzela client software can use either manually entered out-of-band verified public keys, or a local copy of a public database of keys such as a PGP key server.

Algorithm 2 Conversation round: server

```
1.  Collect and decrypt messages: Server $i$ receives many messages
    of the form $e_i = (pk_i, \mathrm{Enc}(s_i, e_{i+1}))$, either from clients or from
    the previous server in the chain. The server first computes all the
    shared secrets $s_i = \mathrm{DH}(sk^i_{\text{server}}, pk_i)$, and then decrypts $e_{i+1}$.
2.  Generate cover traffic: The server samples a random $n_1$ and $n_2$
    from Laplace$(\mu, b)$ capped at 0. Then, the server generates $n_1$
    individual accesses to random mailboxes with random messages,
    and $n_2$ pairs of accesses. These fake messages are added to the
    pool of messages for this round.
3a. Shuffle the messages and sent them to the next server (servers
    $i < n$): The server computes a permutation $\pi$ for this round, and
    permutes all the messages, and sends them to the next server.
    After the next server returns the results, the server unshuffles
    them by applying the inverse permutation $\pi^{-1}$.
3b. Process all messages and mailboxes (server $i = n$): The last
    server in matches up all the accesses to each mailbox. For each
    pair of accesses to the same mailbox, the server exchanges the
    contents of the messages and returns those.
4   Encrypt results and return them: Each resulting message $e'_{i+1}$
    gets encrypted to $e'_i = \mathrm{Enc}(s_i, e'_{i+1})$, and all messages get returned.
```

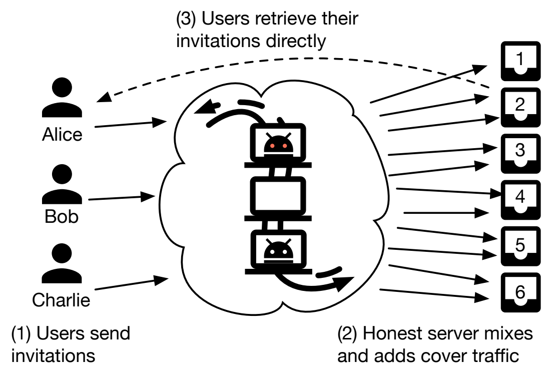
(3) Users retrieve their invitations directly
Alice
Bob
Charlie
1
2
3
4
5
6
(1) Users send invitations
(2) Honest server mixes and adds cover traffic

Figure 7: Overview of Vuvuzela's dialing protocol.

As in the conversation protocol, we hide the first set of variables using a mixnet, and the second set of variables by adding cover traffic. Even though there are potentially many mailboxes to add noise to, we show that the noise required is manageable because invitations are much smaller and less frequent than conversation messages. By carefully choosing the number of mailboxes, we can also make the total noise proportional to the number of actual invitations. Each client needs to download one mailbox worth of noise.

4.2 Unlinking senders from invitations

Like conversations, dialing in Vuvuzela take place in rounds. Each dialing round takes several minutes, which translates into the latency of starting a new conversation.

For each round, the Vuvuzela servers fix a number m of invitation mailboxes to create (we describe how to best set m later on). Then, each client chooses an invitation mailbox (e.g., the mailbox of a friend that she wants to communicate with), and sends an invitation to that mailbox. If a client does not want to start a conversation in a given round, she writes into a special no-op mailbox that is not used by any recipient. Each invitation message consists of the sender's public key, a nonce, and a MAC, all encrypted with the recipient's public key. An invitation for a recipient with public key pk is placed in mailbox $H(pk) \bmod m$. Invitations are also onion-encrypted and mixed, so that they are disconnected from their sender.

4.3 Hiding the number of invitations per mailbox

To prevent traffic analysis, each Vuvuzela server also adds a random number of fake invitations to each mailbox. Servers have to add noise to each mailbox because an adversary can otherwise figure out whether people are talking to Alice, based on the number of invitations in mailbox $H(pk_{\text{alice}}) \bmod m$.

Like in conversations, each server adds noise drawn from a truncated Laplace distribution $N \sim \max(0, \lceil \mathrm{Laplace}(\mu, b) \rceil)$ to each invitation mailbox. The parameters μ and b can be different from conversations, as we shall discuss in §5.

4.4 Tuning the number of mailboxes

In the dialing protocol, the amount of noise that needs to be added to each invitation mailbox turns out to be constant, based only on the desired security parameters and not on the number of users. However, the *number* of invitation mailboxes, m, should be set based on a trade-off between level of noise and amount of data that will be downloaded by clients.

For example, to minimize total noise, the servers could use only one invitation mailbox; however, that would require each client to download *all invitations*, which would be a highly expensive. At the other extreme, using 10,000 mailboxes would decrease the amount of work per client but increase the noise processed by the servers by 10,000×.

To strike a balance between these factors, we propose setting m so that each mailbox has roughly equal amounts of real invitations and noise. In particular, suppose that the average noise per mailbox required by our security parameters is μ, and that there are n users, of which a fraction f send real invitations each round. Then we can set $m = nf/\mu$. This ensures that each mailbox has roughly μ messages of real invitations and μ of noise, and that the overall processing load on the servers is only 2× the load of the real messages.

Note that setting m is only an optimization: even if the system uses too many or too few mailboxes, each user is fully protected by the level of noise, μ, added to her mailbox.

5 Analysis

In this section, we analyze how much noise must be added in Vuvuzela to achieve a given level of differential privacy. We start by analyzing one round of the conversation protocol, then expand to multiple rounds and to dialing. We also summarize the noise required for realistic security parameters.

5.1 One round of conversations

Recall from §2.2 that our goal is to provide (ε, δ)-differential privacy, wherein every observation that can be made by our adversary is nearly equally likely if we change the actions of one user. Given the mixnet design of Vuvuzela, the only variables the adversary can see are the set of users connected to the system, the number of mailboxes that are accessed twice, and the number of mailboxes that are accessed once.

If the adversary has not compromised the last server in the Vuvuzela chain, she cannot see these mailbox access statistics, so the protocol is perfectly private in that she will see indistinguishable outcomes (every client receiving an encrypted string) regardless of the communication pattern between users. If the adversary has compromised the last server, then at least one of the noise-adding servers before it is uncompromised. We show that in this case, the protocol can be made (ε, δ)-differentially private with the right parameters for noise.

Let us call the number of mailboxes that are accessed just once m_1 and the number of mailboxes that are accessed twice m_2. We now analyze how changing the communication pattern of one user, Alice, for one round affects m_1 and m_2 for that round. If Alice was not talking to anyone and starts to, m_1 will fall by 1 and m_2 will grow by 1. If Alice only changes her communication partner, m_1 and m_2 will both stay the same. If Alice was talking to a partner and stops, or if the adversary knocks her partner offline, m_2 will fall by 1 and m_1 will grow by 2 or 1. If an adversary knocks Alice offline, either m_1 will fall by 1 (if Alice was not communicating) or m_2 will fall by 1 and m_1 will grow by 1 (if she was). Finally, if an adversary knocks both Alice and her partner offline, m_2 falls by 1. In all cases, m_1 and m_2 change by at most 2.

We now show that the noise added to m_1 and m_2 in §3.4, which was generated with a Laplace(μ, b) distribution capped below at 0, provides differential privacy.

Theorem 1. Consider the algorithm $M(x)$ that adds noise $N \sim \max(0, \lceil \text{Laplace}(\mu, b) \rceil)$ to a value $x \geq 0$. Then M is (ε, δ)-differentially private with respect to changes of up to 2 in $|x|$, for parameters $\varepsilon = \frac{2}{b}$ and $\delta = \frac{1}{2}\exp\left(\frac{2-\mu}{b}\right)$.

Proof. Given in Appendix A. □

Finally, because the adversary can observe both m_1 and m_2, we need to make each of them $(\varepsilon/2, \delta/2)$-differentially private to get an overall level of privacy of (ε, δ). This gives:

Theorem 2. Adding noise $N \sim \max(0, \lceil \text{Laplace}(\mu, b) \rceil)$ to each of m_1 and m_2 is (ε, δ)-differentially private with respect to the actions of each user, for parameters

$$\varepsilon = \frac{4}{b} \quad \text{and} \quad \delta = \exp\left(\frac{2-\mu}{b}\right).$$

5.2 Multiple rounds of conversations

Adversaries will try to learn information about users across multiple rounds of communication, possibly perturbing the system each round (*e.g.,* knocking Alice offline) based on observations in earlier ones. This scenario is known as adaptive composition in the differential privacy literature. Fortunately, differential privacy provides a bound on ε and δ after k rounds of composition, with the appealing property that the noise needed for a given ε and δ grows only with $\sqrt{k}$.

k	ε'	δ'		Mean	Std.dev.
100,000	ln 3	10^{-4}	→	118,000	8,000
500,000	*(same)*	*(same)*	→	282,000	17,000
1,000,000	*(same)*	*(same)*	→	408,000	24,000
500,000	ln 2	10^{-4}	→	438,000	27,000
(same)	0.2	*(same)*	→	1,490,000	91,000
500,000	ln 3	10^{-3}	→	225,000	15,000
(same)	*(same)*	10^{-5}	→	338,000	19,000

Table 1: Mean μ and standard deviation $\sqrt{2}b$ of the Laplace noise that must be added to conversation mailbox counts to achieve a given level of differential privacy after k rounds. All rows use $d = 0.4\delta'$ in Theorem 3.

Theorem 3. Consider the algorithm M that adds noise $N \sim \max(0, \lceil \text{Laplace}(\mu, b) \rceil)$ to each of m_1 and m_2 over k Vuvuzela rounds. Then M is (ε', δ')-differentially private with respect to the actions of each user, with parameters

$$\varepsilon' = \sqrt{2k \ln(1/d)}\varepsilon + k\varepsilon(e^{\varepsilon} - 1) \quad \text{and} \quad \delta' = k\delta + d,$$

for any $d > 0$, where ε and δ are as in Theorem 2.

Proof. Direct application of Theorem 3.20 in [16]. □

Note that although this theorem considers k rounds, if a user uses Vuvuzela for more than k rounds but only sends messages in k of them, the adversary still cannot learn anything about the user's correspondents. Even if the adversary knew precisely which k rounds the user communicated in (i.e., exchanged messages with a partner instead of a random mailbox), she would be unable to distinguish the observations during these rounds from those in a scenario where the user never communicates (i.e., exchanges fake messages with a random mailbox).

5.3 Summary: Noise needed for conversations

To give a sense of the amount of noise required for different levels of security, Table 1 shows the mean μ and standard deviation $\sqrt{2}b$ of the Laplace distribution for various k, ε' and δ'. In general, ε' is recommended to be set between 0.1 and ln 3 [15], and δ' should be small, *e.g.,* 10^{-4}. This is because δ' covers events that might have *zero* probability under one behavior of the user but might happen under another. For example, if the count of mailboxes that are accessed twice, m_2, is 0 in some rounds after adding noise, the adversary will know that no users communicated, because our noise never subtracts messages. A low δ' ensures such events are extremely unlikely.

In addition, given Theorems 2 and 3, we see that the noise required, μ, scales as follows with each parameter:

- μ increases proportionally to $\sqrt{k}$.
- μ increases linearly with $1/\varepsilon'$.

k	ε'	δ'		Mean	KB/round
20,000	ln 3	10^{-4}	$\rightarrow$	24,000	1,900
50,000	*(same)*	*(same)*	$\rightarrow$	40,000	3,200
100,000	*(same)*	*(same)*	$\rightarrow$	59,000	5,000
50,000	ln 2	10^{-4}	$\rightarrow$	62,000	5,000
(same)	ln 3	10^{-5}	$\rightarrow$	48,000	3,800

Table 2: Mean μ of the Laplace noise that must be added to each dialing mailbox to achieve a given level of differential privacy after k rounds. All rows use $d = 0.4\delta'$ in Theorem 3. We also show the bytes per round of noise that each user will need to download assuming 80-byte invitations. In practice a round can be as much as 1000 seconds.

- μ increases proportionally to $\log(1/\delta')$.
- μ is *independent* of the total number of users.

5.4 Dialing protocol

In Vuvuzela's dialing protocol, we need to add noise to every dialing mailbox because adversaries can distinguish between them. Nonetheless, the total amount of noise per second can be smaller than in conversations, for three reasons:

1. Dialing rounds can be longer than conversation rounds, say 10–15 minutes.
2. Dialing is less common than conversation messages, decreasing the number of protected rounds k we might want.
3. Invitations (just a public key) are shorter than messages.

The analysis for dialing is similar to Theorems 2 and 3, except that modifying each user's action in a round only changes up to two mailbox counts by 1 each, which gives $\varepsilon = \frac{2}{b}$ and $\delta = \frac{1}{2}\exp\left(\frac{1-\mu}{b}\right)$. As a result, the number of noise messages is about half as large as in Table 1 for a given k, ε' and δ'.

In addition, for dialing, k represent the maximum number of distinct invitations we want to protect a user for, which can likely be set smaller (*e.g.,* 50,000).[6]

Table 2 summarizes the required noise per round per mailbox for varying security parameters. As discussed in Section 4.4, the number of mailboxes should be set so that the number of "real" invitations per mailbox is roughly equal to this noise. This means the total noise processed by servers will be roughly twice the traffic due to the invitations themselves.

6 Implementation

To evaluate Vuvuzela's design, we implemented a prototype of Vuvuzela in Go. Our prototype consists of approximately 2000 lines of code.

To take advantage of multiple many-core machines, our prototype is parallelized and shards the work of a single logical Vuvuzela server across multiple physical machines. The sharding implementation consists of an extra shuffle phase that randomly distributes all incoming messages in a round across all physical machines comprising a single logical Vuvuzela server.

The most computationally expensive part of Vuvuzela's implementation is the repeated use of Diffie-Hellman in the wrapping and unwrapping of encryption layers. Vuvuzela must use new keys for each individual message, as otherwise the same key appearing twice would be a variable visible to an adversary. For performance, we use an optimized assembly implementation of Curve25519 from Go's crypto library.

[6]Note that if an adversary purposefully sends many invitations to the same user, they will not count against this bound—since the invitations are sent by the adversary, they do not convey any new information about the user.

7 Evaluation

Our evaluation answers the following questions:

1. Can Vuvuzela servers support a large number of users and messages? (§7.2)
2. Does Vuvuzela provide strong privacy guarantees? (§7.3)
3. Does Vuvuzela provide acceptable performance to clients? (§7.4)

7.1 Experimental setup

To answer the above questions, we run a series of experiments on Amazon EC2 virtual machines. All servers used are `c4.8xlarge` machines with 36 Intel Xeon E5-2666 v3 CPU cores, 60 GB of RAM, and 10 Gbps of network bandwidth. The machines run Linux kernel version 3.14 and Go 1.4.2.

We use the following base parameters across our different performance experiments. Our chain consists of 3 Vuvuzela servers, each corresponding to one physical server. Conversation messages are 512 bytes long (including 16 byte encryption overhead). Invitation messages are 80 bytes long (including 48 bytes of overhead). Using one additional server, we simulate 1 million users. All users send messages each round; two-thirds are talking to other users, and one-third are not communicating with anyone (and thus are accessing mailboxes by themselves). We pick $\mu = 300{,}000$ for the conversation protocol, which protects users for $k = 500{,}000$ message exchanges, and $\mu = 40{,}000$ for the dialing protocol, which covers $k = 50{,}000$ dialing rounds.

All our experiments run on servers in the same data center. In an actual deployment, servers should run in different data centers so that no single operator controls all servers. Running in multiple data centers would increase the latency between servers, but network latency has little effect on Vuvuzela's performance, as each round requires just one round-trip between adjacent pairs of servers. Vuvuzela is bandwidth-bound, and has comparable bandwidth requirements to any other messaging system with the same number of users and messages.

7.2 Server performance

To evaluate whether Vuvuzela can support many users and messages, we measure the performance of Vuvuzela's conversation and dialing protocols when faced with anywhere from zero to two million users, two-thirds of which are active (conversing or dialing respectively). With one million users, our prototype achieves a throughput of approximately 15,000

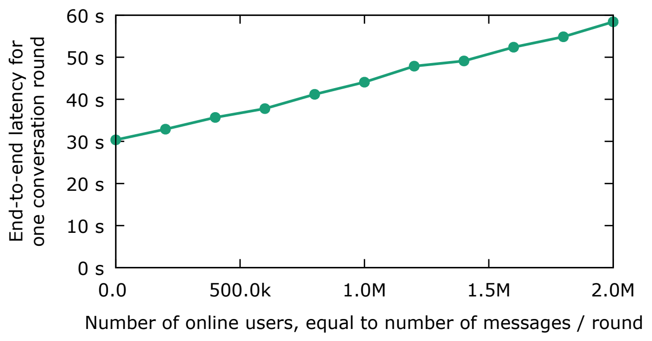

Figure 8: Performance of Vuvuzela's conversation protocol when varying the number of users online. Every user sends a message every round; two-thirds of them are active conversations, and one-third are fake messages from users not engaged in an active conversation.

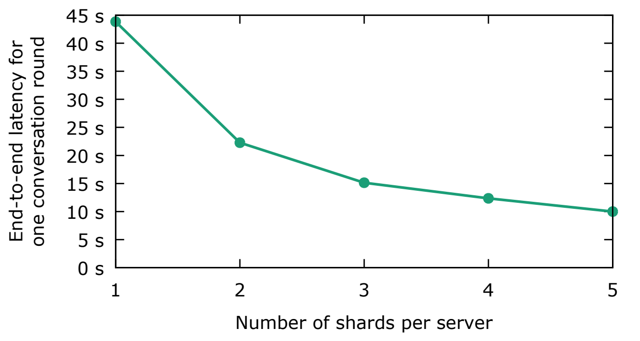

Figure 9: Performance of Vuvuzela's conversation protocol when varying the number of shards.

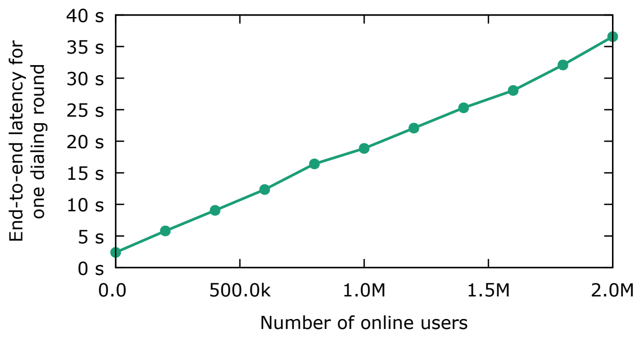

Figure 10: Performance of Vuvuzela's dialing protocol when varying the number of users online. Two-third of the users dial someone every round.

conversation messages per second. In each case, the key result of interest is the time it takes the Vuvuzela network to finish a round and return the results to users (although multiple rounds can be pipelined with one another). We then evaluate the underlying costs behind Vuvuzela's performance, and finish by examining how Vuvuzela performs as more servers are added.

Conversation protocol. Figure 8 shows the total latency of a conversation round, with the number of online users ranging from zero to two million users. The latency is end-to-end: it includes shuffling, generation of cover traffic, encryption and decryption, RPC overhead, and so forth, and is thus representative of overall performance.

Our results show that Vuvuzela scales linearly with the number of users and messages, from a 30 second round latency with zero users to approximately 60 seconds with two million users. With a million online active users, a single round of the conversation protocol takes 45 seconds.

The cover traffic required for Vuvuzela's conversation protocol is constant: the amount for 10 users is the same as for two million users. The baseline time to process cover traffic can be seen with zero users, and is 30 seconds. Each server in the chain adds $\mu \times 3$ fake messages on average, for a total of 1.8 million messages when there are zero users. With 2 million users, each adding a single message, we get 3.8 million messages on the right side of the graph. Vuvuzela's performance scales linearly with the number of messages sent.

Dialing protocol. Figure 10 shows the end-to-end round latency for Vuvuzela's dialing protocol. Like the conversation protocol, the dialing protocol scales linearly, from a couple of seconds with zero users to 36 seconds with two million users. As with the conversation protocol, two-thirds of the users are dial another user each round, and the other one-third do not, and instead write to the special no-dial mailbox.

Our prototype aims to get approximately 3μ real invitations in each mailbox, in addition to the baseline 3μ fake invitations. Because the μ in the dialing protocol is much smaller (only 40,000) than in the conversation protocol, the base cost with zero users online is much smaller, and so a round completes in only a couple of seconds. With two million online users, the total number of invitations in all mailboxes is 3 million.

Dominant costs. The most expensive part of Vuvuzela is the wrapping and unwrapping of encryption layers. Each 36-core machine can perform approximately 340,000 Curve25519 Diffie-Hellman operations per second. In the conversation protocol, with two million users, each server must perform one Diffie-Hellman operation for each of the 3.8 million messages. To avoid leaking information about a server's permutation of messages, one server cannot start processing a round until the previous server finishes, so the best-case end-to-end conversation round latency would be $(3.8 \cdot 10^6 \times 3)/(3.4 \cdot 10^5) \approx 33$ seconds. This shows that Vuvuzela's full protocol (serialization, shuffling, cover traffic generation, etc), is within $2\times$ of the cost of the inevitable cryptographic operations. Vuvuzela's dialing protocol is similarly close to the lower-bound cost of the underlying cryptographic operations.

Adding more machines per logical server. With many millions of active users, Vuvuzela would have to scale past a single machine per logical Vuvuzela server. To evaluate how well Vuvuzela scales with more physical machines, Figure 9 shows the conversation round length as we vary the number of machines per logical server. By distributing the expensive

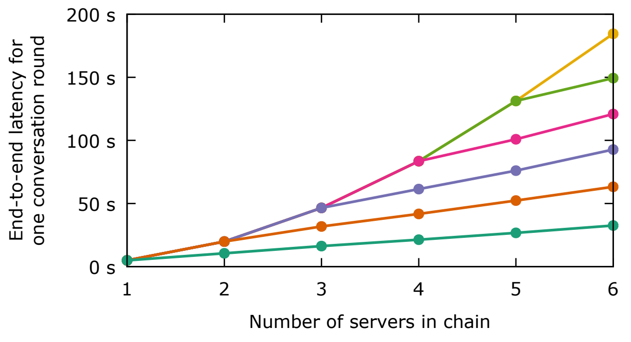

Figure 11: Performance of Vuvuzela's conversation protocol when varying the chain length.

cryptographic operations, Vuvuzela scales close-to-linearly with multiple machines, decreasing round length from 45 seconds on one machine to around 10 seconds on 5 machines.

Adding more logical servers to the chain. Deployments of Vuvuzela can vary the number of Vuvuzela servers. Increasing the number of servers provides stronger security. On the other hand, adding more servers increases end-to-end latency and increases the amount of work each server must do to add cover traffic, as each cover traffic message must be encrypted once for every server later in the chain. Figure 11 shows total end-to-end latency for different chain lengths. The graph shows both the total round time (the top line), as well as the amount time spend by each server (space between pairs of lines).

Performance scales roughly quadratically with the number of servers in the chain. This is as expected, as each of s servers must decrypt cover traffic from all previous servers in the chain, with $O(s)$ work for all $O(s)$ servers, leading to $O(s^2)$ scaling. The graph reflects this with the amount of space between pairs of lines growing linearly.

7.3 Privacy

We evaluate two aspects to privacy: does our Vuvuzela prototype produce a random distribution of mailbox accesses, and at what long-term time scales can Vuvuzela provide privacy?

Access counts. We looked at the distribution of m_1 and m_2 with varying numbers of users and percentages of users talking. As expected, m_1 and m_2 include noise and do not reveal the exact number of people talking. However, an adversary can learn the approximate percentage of people talking by averaging m_2 over time. This is in line with our differential-privacy goal of protecting the privacy of individual users but not protecting aggregate information about all users.

Long-term privacy. In the experiments we used $\mu = 300{,}000$ for the conversation protocol, which protects users for $k = 500{,}000$ message exchanges. If Vuvuzela runs a conversation round every minute, this corresponds to 347 days of active message exchange using the system, or three years of activity even if the user is chatting through Vuvuzela for eight hours a day. Hopefully, in 3 years' time, hardware will have improved to the point where ε can be increased to maintain privacy for an even longer time scale.

For dialing, we used $\mu = 40{,}000$, which corresponds to 50,000 dialing rounds. This also provides 3 years of privacy. These numbers show that Vuvuzela can provide privacy over time.

7.4 Client requirements

To evaluate whether Vuvuzela is practical for end users, we measure the latency and throughput achievable by a single user, and also measure the bandwidth requirements that the Vuvuzela client places on the user's network connection.

Latency and throughput. In our analysis we assumed 10-minute dialing rounds, and 1-minute conversation rounds. This means that a client must wait at most 20 minutes to start a conversation. This makes Vuvuzela well-suited for slower-paced, e-mail-like, communication patterns where users queue up a bunch of messages to send. We could increase the frequency of dialing rounds, at the cost of increasing required client bandwidth. Conversations themselves move fairly quickly, with 1-minute message latency and 512 bytes per message.

Bandwidth usage. For the conversation protocol, each client sends and downloads a 512-byte message plus encryption overhead per minute. Thus, the bandwidth requirements for sending and receiving conversation messages are negligible.

The dialing protocol is more expensive, as each client must download an entire mailbox worth of invitations. With $\mu = 40{,}000$ and 3 servers, that comes out to about 120,000 fake invitations, as well as 120,000 real invitations, for a total of 19 MB per round. With 10-minute rounds, a client uses an average of 32 KB/sec for downloading invitations. While not insignificant, Vuvuzela provides a substantial improvement over a strawman design that would download all 2,000,000 invitations every round. Until Vuvuzela has that many users, however, clients might be better off downloading all invitations, until that becomes prohibitively expensive.

Note that the cost for both protocols is *independent* of the number of users, so that even with many millions of users, a single DSL or 3G phone could easily keep up with the required tens of kilobytes per second of bandwidth.

8 Discussion and Limitations

To design a private messaging system protected from strong adversaries in Vuvuzela, we took an approach consisting of two steps. First, we *identified and minimized the number of observable variables* that an adversary can see. Second, we hid the few remaining variables using *noise with quantifiable security properties*, leveraging tools from differential privacy to make guarantees about them. We hope that this approach also proves useful in the design of other secure systems.

Even though Vuvuzela provides strong guarantees regarding the inferences an adversary can make about each user, the systems still has limitations that require care.

PKI. Vuvuzela depends on an external public key infrastructure. Specifically, Vuvuzela's dialing protocol requires public keys in two ways. First, in order to start a conversation, a user must know the public key of the other party. Looking up this key on-demand over the Internet via some key server would disclose who the user is dialing, so Vuvuzela clients should store public keys for contacts ahead of time. Second, when receiving a call via the dialing protocol, the recipient needs to identify who is calling, based on the caller's public key. Here, the caller can supply a certificate along with the invitation, if the recipient does not already know the caller; doing so avoids the need for the recipient to contact a key server.

Forward secrecy. Vuvuzela does not achieve forward secrecy for metadata in the dialing protocol. This is because invitations are encrypted under the long-term public key of the recipient, so an adversary that compromises a user's private key *and* compromises the last server in Vuvuzela's chain can decrypt all past invitations with the user's private key to determine who called this user in the past. On the other hand, Vuvuzela's communication protocol provides forward secrecy by choosing new server keys for each round, and existing techniques can achieve forward secrecy for message contents [27].

It may be possible to achieve forward secrecy for dialing by rotating user public and private keys. One approach would be to rely on a more sophisticated PKI system that supports rotation of user keys, although contacting an external PKI requires care to avoid disclosing information about what users are communicating. Another approach would be to use a forward-secure public-key encryption scheme [5].

Online users. Vuvuzela does not hide which users are connected to the system at each time, so an adversary might infer that two users communicate if they are online at the same time. The countermeasure here is for users to stay connected to Vuvuzela at all times, even if they are not communicating, and we have shown that this requires only tens of KB/s of bandwidth.

Message size. Vuvuzela's fixed message sizes are a good fit for text communication, but they are not well-suited for transferring large files. Providing privacy for large file transfers is an interesting area for future work.

Group privacy. Differential privacy makes guarantees about individual users, but not about groups [16]. For example, if an adversary suspects that a group of 1,000 people tend to communicate frequently with each other, she can block other users from the system and observe a noticeable level of message exchanges. However, she cannot distinguish whether *any specific individual* in the group she isolated is actually communicating.

Denial of service attacks. Vuvuzela must consider two kinds of DoS attacks. First, clients can submit many messages. Since all clients must communicate with the first server in Vuvuzela's chain, that server can mitigate client DoS attacks through existing approaches: requiring users to sign up for an account, requiring users to prove ownership of an account on other systems (e.g., Facebook), proof-of-work, or even payment for service. Requiring the user to identify themselves to the first server does not weaken Vuvuzela's privacy guarantees since we assume the adversary already knows which users are using Vuvuzela.

Second, malicious servers can attempt to mount denial-of-service attacks against the Vuvuzela system. Vuvuzela does not defend against such attacks. In general, DoS attacks are unavoidable with our strong adversary model, which assumes that an adversary can knock any user offline. However, a production implementation of Vuvuzela would need to provide a mechanism for identifying and evicting servers that misbehave.

Bandwidth use. By using a fixed chain of servers, Vuvuzela requires every message to pass through each of the servers. This translates into significant bandwidth requirements for each Vuvuzela server. This fixed cascade structure enables a simple analysis of Vuvuzela's privacy guarantees. However, in future work, we hope to explore a more Tor-like distributed design where the bandwidth costs are spread out over a larger network of servers, without requiring that each message traverse every server. We expect the main challenges will be in coming up with a suitable security definition for this setting, and in constructing a provable analysis of the resulting metadata privacy.

Multiple conversations. To enable multiple concurrent conversations, Vuvuzela clients can perform multiple conversation protocol exchanges in each round. To ensure that the number of exchanges does not disclose how many active conversations a user has, the client should pick a maximum number of conversations *a priori* (say, 5), and always send that many conversation protocol exchange messages per round.

9 Related Work

Secure messaging. Recent work has shown that secure messaging systems can provide end-to-end encryption at scale, starting from systems such as OTR [4, 19], Axolotl [27], and TextSecure [26], and now being deployed by WhatsApp [25]. However, these systems encrypt only the content of the message; metadata about what users are communicating is still observable to an adversary. For example, even Pond [22] explicitly states that it "seeks to prevent leaking traffic information against everyone except a global passive attacker". In contrast, Vuvuzela is able to protect metadata even in the face of such strong adversaries.

Anonymous communication systems. Anonymous communication has been studied since Chaum's work on mixnets [7] and DC-nets [6]. Unlike Vuvuzela, however, previous systems do not simultaneously provide both scalability and protection against traffic analysis.

Mixnet-style systems [7, 13, 17], Crowds [28], Freenet [9], and onion routing [14, 31] can scale to millions of users, but are amenable to traffic analysis by a strong adversary. For example, an adversary may learn communication partners by passively observing traffic at each node [11, 24] or by actively delaying some users' packets to see the effect on others [1].

On the other hand, systems with provably strong security guarantees have relied on mechanisms that scale quadratically in the number of users. Herbivore [18] and Dissent [32] form broadcast groups of up to 5,000 users each, which limits each user's anonymity and requires significant overhead to be used for point-to-point communication (as each message is broadcast to all users in its group). Riposte [10] can scale anonymity sets to a few million users over many hours, but still relies on broadcasts and limits writes to a few hundred per second. Systems based on private information retrieval [8], such as the Pynchon Gate [30], decrease the amount of data each user reads but still require $O(n^2)$ computation for n users.

Cover traffic. Several mixnet and onion routing systems have sought to make traffic analysis more difficult using cover traffic, *i.e.,* fake traffic on each communication link [3, 17], or delaying messages [21]. However, it has been shown that all these schemes still reveal information after multiple rounds of observation [23]. To add sufficient noise to cover users for hundreds of thousands of rounds of communication, one would need tens of thousands of noise messages per link. The key insight in Vuvuzela is to *reduce the number of variables that an adversary can observe*, which subsequently allows Vuvuzela to add adequate noise (enough to provably protect hundreds of thousands of message exchanges) with an acceptable cost.

Differential privacy. Several authors have used differential privacy to analyze anonymous communication schemes. AnoA [2] offers a framework for this purpose but limits analysis to one round, while Danezis [12] considers multiple rounds. We use differential privacy in Vuvuzela to provide quantifiable security over hundreds of thousands of message exchanges.

10 Conclusion

Vuvuzela is the first system to scale private messaging to millions of users and tens of thousands of messages per second, while protecting against traffic analysis by a powerful adversary that can compromise all but one of the system's servers. Vuvuzela achieves this through a novel approach consisting of two steps. First, Vuvuzela's protocol is designed to clearly *identify and minimize the number of observable variables* in the system. Second, Vuvuzela's protocol hides these variables using *noise with quantifiable security properties*, leveraging tools from differential privacy. Together, these techniques let Vuvuzela achieve private messaging at a scale orders of magnitude higher than prior systems.

A Proof of Theorem 1

In this appendix, we prove Theorem 1 from Section 5.1. Consider the algorithm $M(x)$ that adds noise $N \sim \max(0, \lceil \text{Laplace}(\mu, b) \rceil)$ to a value $x \geq 0$. We wish to show that M is (ε, δ)-differentially private with respect to changes of up to 2 in $|x|$, for parameters $\varepsilon = \frac{2}{b}$ and $\delta = \frac{1}{2}\exp\left(\frac{2-\mu}{b}\right)$.

Proof. We will show the result for noise $N' \sim \max(0, \text{Laplace}(\mu, b))$ without the ceiling, since post-processing the result of a differentially private function (in this case rounding it) keeps it differentially private [16].

Consider $x, y \geq 0$ such that $|x-y| \leq 2$, and let $m = \max(x, y)$. For any set $T \subseteq (m, \infty)$, $\Pr[x+N' \in T] \leq e^{\varepsilon} \Pr[y+N' \in T]$ by properties of the Laplace distribution (Theorem 3.6 in [16]). In particular, to reach values in T, N' has to add positive noise to both x and y, and the shape of the Laplace distribution ensures that these probabilities are within a factor of e^{ε} for $\varepsilon = 2/b$.

On the other hand,

$$\begin{aligned}\Pr[x + N' \leq m] &\leq \Pr[x + N' \leq x + 2] = \Pr[N' \leq 2] \\ &= \Pr[\text{Laplace}(\mu, b) \leq 2] \\ &= \frac{1}{2}\exp\left(\frac{2-\mu}{b}\right) = \delta\end{aligned}$$

Thus for any arbitrary set of values S, we have

$$\begin{aligned}\Pr[x + N' \in S] &= \Pr[x + N' \in S \cap (-\infty, m]] \\ &\quad + \Pr[x + N' \in S \cap (m, \infty)] \\ &\leq \delta + \Pr[x + N' \in S \cap (m, \infty)] \\ &\leq \delta + e^{\varepsilon} \Pr[y + N' \in S \cap (m, \infty)] \\ &\leq \delta + e^{\varepsilon} \Pr[y + N' \in S]\end{aligned}$$

This means that M is (ε, δ)-differentially private. □

Acknowledgments

Thanks to M. Frans Kaashoek and Nick Mathewson for feedback that helped improve this paper.

References

[1] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Proceedings of the Workshop on Information Hiding*, pages 245–257, Pittsburgh, PA, Apr. 2001.

[2] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi. AnoA: A framework for analyzing anonymous communication protocols. In *Proceedings of the 26th IEEE Computer Security Foundations Symposium*, pages 163–178, New Orleans, LA, June 2013.

[3] O. Berthold and H. Langos. Dummy traffic against long term intersection attacks. In *Proceedings of the Workshop on Privacy Enhancing Technologies*, pages 110–128, Dresden, Germany, Mar. 2003.

[4] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 Workshop on Privacy in the Electronic Society*, Washington, DC, Oct. 2004.

[5] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. *Journal of Cryptology*, 20(3):265–294, July 2007.

[6] D. Chaum. The dining cryptographers problem: *unconditional sender and recipient untraceability*. *Journal of Cryptology*, 1(1):65–75, 1988.

[7] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), Feb. 1981.

[8] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45(6): 965–981, Nov. 1998.

[9] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, Berkeley, CA, July 2000.

[10] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, San Jose, CA, May 2015.

[11] G. Danezis. Statistical disclosure attacks: Traffic confirmation in open environments. In *Proceedings of the 18th International Conference on Information Security*, pages 421–426, Athens, Greece, May 2003.

[12] G. Danezis. Measuring anonymity: a few thoughts and a differentially private bound. In *Proceedings of the DIMACS Workshop on Measuring Anonymity*, May 2013.

[13] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of the 24th IEEE Symposium on Security and Privacy*, pages 2–15, Oakland, CA, May 2003.

[14] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Usenix Security Symposium*, pages 303–320, San Diego, CA, Aug. 2004.

[15] C. Dwork. Differential privacy: a survey of results. In *Proceedings of the 5th Intl. Conf. on Theory and Applications of Models of Computation*, Xi'an, China, Apr. 2008.

[16] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[17] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, pages 193–206, Washington, DC, Nov. 2002.

[18] S. Goel, M. Robson, M. Polte, and E. G. Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical Report 2003-1890, Cornell University, Computing and Information Science, Ithaca, NY, 2003.

[19] I. Goldberg and The OTR Development Team. Off-the-record messaging, 2015. `https://otr.cypherpunks.ca/`.

[20] M. Hayden. The price of privacy: Re-evaluating the NSA. Johns Hopkins Foreign Affairs Symposium, Apr. 2014. `https://www.youtube.com/watch?v=kV2HDM86XgI&t=17m50s`.

[21] D. Kesdogan, J. Egner, and R. Büschkes. Stop-and-go-MIXes providing probabilistic anonymity in an open system. In *Proceedings of the Workshop on Information Hiding*, pages 83–98, Portland, OR, Apr. 1998.

[22] A. Langley. Pond, 2015. `https://pond.imperialviolet.org/`.

[23] N. Mathewson and R. Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In *Proceedings of the Privacy Enhancing Technologies Workshop*, pages 17–34, May 2004.

[24] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *Proceedings of the 26th IEEE Symposium on Security and Privacy*, pages 183–195, Oakland, CA, May 2005.

[25] Open Whisper Systems. Open Whisper Systems partners with WhatsApp to provide end-to-end encryption, Nov. 2014. `https://whispersystems.org/blog/whatsapp/`.

[26] Open Whisper Systems. TextSecure protocol v2, 2015. `https://github.com/WhisperSystems/TextSecure/wiki/ProtocolV2`.

[27] T. Perrin and M. Marlinspike. Axolotl ratchet, 2014. `https://github.com/trevp/axolotl/wiki`.

[28] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. Technical Report 97-15, DIMACS, Apr. 1997.

[29] A. Rusbridger. The Snowden leaks and the public. *The New York Review of Books*, Nov. 2013.

[30] L. Sassaman, B. Cohen, and N. Mathewson. The Pynchon gate: A secure method of pseudonymous mail retrieval. In *Proceedings of the Workshop on Privacy in the Electronic Society*, Arlington, VA, Nov. 2005.

[31] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *Proceedings of the 18th IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, CA, May 1997.

[32] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the 10th Symposium on Operating Systems Design and Implementation (OSDI)*, Hollywood, CA, Oct. 2012.