Vuvuzela: Scalable Private Messaging that Hides Metadata

by

David Lazar

B.S., University of Illinois at Urbana-Champaign (2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2016
[February 2016]

Signature redacted
Author .....
Department of Electrical Engineering and Computer Science
January 29, 2016

Signature redacted
Certified by .....
Nickolai Zeldovich
Associate Professor
Thesis Supervisor

Signature redacted
Accepted by .....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Theses

Vuvuzela: Scalable Private Messaging that Hides Metadata

by

David Lazar

Submitted to the Department of Electrical Engineering and Computer Science
on January 29, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

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 [39], 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 an attacker, and to use differential privacy techniques to add noise to all observable 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 68,000 messages per second for 1 million users with a 37-second end-to-end latency on commodity servers.

Thesis Supervisor: Nickolai Zeldovich
Title: Associate Professor

Preface

This dissertation presents Vuvuzela, the first text messaging system that provides strong metadata privacy and scales to millions of users. One artifact of this dissertation is the system itself, which is free software and is available on Github:

> https://github.com/davidlazar/vuvuzela

Future work related to Vuvuzela will be published in this repository. The other major artifact of this work is our SOSP 2015 paper, on which this dissertation is based:

> ⋆Jelle van den Hooff, ⋆David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP)*, Monterey, California, October 2015.

Thank you to my coauthors, Jelle, Matei, and Nickolai; this dissertation is the result of our combined efforts. I further thank my advisor, Nickolai Zeldovich, for guiding me toward interesting problems, and then helping me solve them. Thank you to PDOS for boundless feedback and for introducing me to systems research. Thank you to Joanna and my family for their love and ongoing support. Finally, thank you to Edward Snowden for showing the public that metadata privacy is an important problem.

Contents

1 The Adversary **9**

1.1 Why is hiding metadata hard? 13

1.2 Security goals 14

1.3 Threat model 16

2 Hiding Metadata **17**

2.1 Conversation protocol 21

2.2 Dialing protocol 26

2.3 Analysis 30

3 Evaluation **41**

3.1 Server performance 42

3.2 Client performance 45

4 Research Prospects **47**

5 Related Work **51**

Conclusion **55**

ONE

The Adversary

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" [35: ¶7] and that "we kill people based on metadata" [24]. 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 [39] and Riposte [13]. 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 [36]. As a result, these systems have scaled to just 5,000 users [39] or hundreds of messages per second [13].

On the other hand, scalable systems like Tor [17] and mixnets [9] 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 adversary over time [15, 28], while adversaries that can actively disrupt traffic (e.g., inject delays) gain even more information [1].

This dissertation presents Vuvuzela, a system that provides scalable private point-to-point text messaging. Vuvuzela prevents an adversary from learning 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.

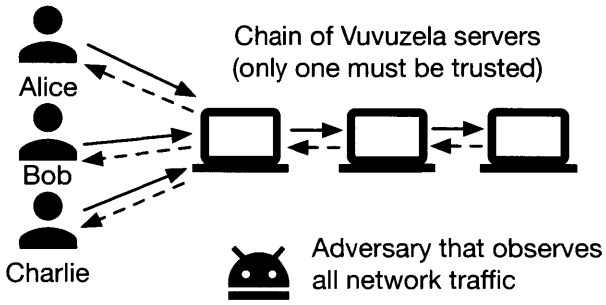

Figure 1.1: Vuvuzela's overall architecture. The Vuvuzela network consists of a chain of servers, at least one of which is assumed to be trustworthy.

Vuvuzela works by routing user messages through a chain of servers, as shown in Figure 1.1, 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 guarantees about the level of privacy provided by cover traffic. We achieve this using two key techniques.

[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.

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 conversation protocol, used for sending messages, exposes just two variables: the total number of users engaged in a conversation, and the total number of users not engaged in one. It does *not* reveal which users are in each 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 [19] 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—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.

Vuvuzela's privacy guarantees are expressed in terms of differential privacy, which can be thought of as "plausible deniability." Each time a user sends a message in Vuvuzela, an adversary may be able to infer a small amount of statistical information—e.g., based on what the adversary observed, it seems a bit more likely that Alice and Bob were talking. However, Vuvuzela ensures that even the total information, over many messages exchanged by a user, still provides a strong level of differential privacy. For instance, our typical configuration ensures that, for a user who sends and receives 200,000 messages, the adversary's confidence about any given suspicion (e.g., whether Alice and Bob are talking) remains within 2× of what it was before the adversary monitored Vuvuzela.

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

To evaluate Vuvuzela, we implemented a prototype in Go on several commodity servers (36-core VMs on EC2). Results show that Vuvuzela can support 1 million users exchanging text messages (up to 240 bytes each) with an end-to-end latency of 37 seconds, achieving a throughput of 68,000 messages/sec, with the privacy level described in the previous paragraph. The cover traffic needed for this level of privacy is equivalent to about 1.2 million active users. More importantly, the cover traffic is *independent* of the number of active users, which helps Vuvuzela scale up well. For instance, scaling up to 2 million users increases the latency from 37 to 55 seconds.

Vuvuzela's results come at a non-trivial bandwidth cost. In the above configuration, clients use an average of 12 KB/sec (adding up to 30 GB over a month of continuous use, which may be high for a mobile phone with metered data service). Servers use an average of 166 MB/sec, and Vuvuzela also requires an untrusted CDN or BitTorrent-like system to distribute public dialing information to users (12 KB/sec per user, or 12 GB/sec in aggregate). Nonetheless, Vuvuzela is the first system to achieve private communication at this scale.

In summary, the contributions of this dissertation are:

- 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 analysis of the privacy provided by Vuvuzela and an evaluation of Vuvuzela's performance and scalability.

1.1 Why is hiding metadata hard?

Vuvuzela aims to provide point-to-point messaging between users in a way that is *private* in the face of a strong adversary, who 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 system.

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 the server operates in fixed rounds, and each round, it first collects messages from all clients that wish to send one, and then sends each message to its recipient. Although the adversary cannot immediately tell which sender was responsible for a given recipient's message, he can still discover relevant information. For instance, if the adversary suspects that Alice and Bob are communicating, he can temporarily block network traffic from Alice, and see whether Bob stops receiving messages. Or, in our model of a strong adversary, he can block traffic from *all* clients except for Alice and Bob, and see whether any messages got exchanged when just they are online.

As discussed in the chapter introduction, previous systems handle this problem using mechanisms that limit scaling, such as broadcasting all messages to all users [13, 39] or cryptographic schemes like PIR [36]. 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 he 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.

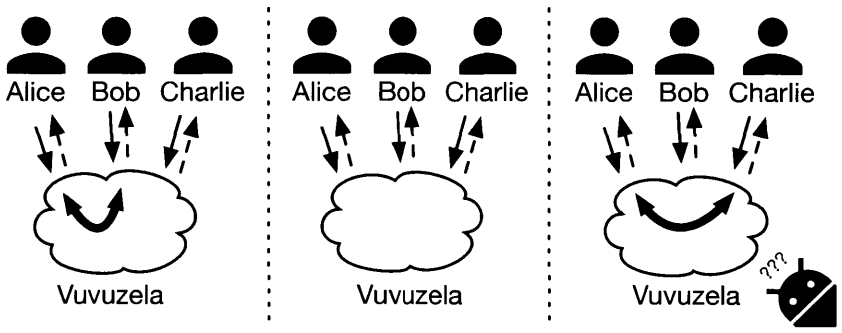

Figure 1.2: Vuvuzela's security goal. An adversary must not be able to distinguish between various possible worlds. In one world, Alice is communicating through Vuvuzela with Bob. In another, she is connected but not exchanging messages with other users. In a third, she is communicating with Charlie. Vuvuzela gives Alice differential privacy: any event observed by the adversary has roughly equal probability in all worlds.

1.2 Security goals

Informally, the security definition we want is the following: for each user (call her Alice), the adversary *should not be able to distinguish between any of Alice's possible communication patterns*, even after Alice exchanges many messages.

We make this definition precise using differential privacy [18], which can be thought of as a formalization of "plausible deniability." Differential privacy says that for any observation O that the adversary might make of the system, the probability of observing O should be similar regardless of Alice's communication pattern, as shown in Figure 1.2. Formally, we define differential privacy for Vuvuzela as:

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

In our case, inputs x and y are the user actions (i.e., which users are communicating). We consider two inputs *adjacent* if they differ only in the messages exchanged by one user (say, Alice).[3] One of the inputs represents the real actions taken by Alice (e.g., that she exchanged messages with Bob), while the other input represents Alice's hypothetical "cover story," which is an alternative set of actions that Alice could claim to have made, and which should appear almost as plausible as her real actions (e.g., that she never communicated with anyone, or that she exchanged messages with Charlie). The func-

[3] As we describe in §2.1, when two users are communicating through Vuvuzela, each one is performing a message exchange, so it makes sense to talk about the message exchanges of one user.

tion M represents the observations made by the adversary after he performs whatever manipulation he wishes of the system.

Intuitively, the definition says that any set of observations by an adversary (the payload of network packets, the state of compromised servers, etc.) is almost as likely given Alice's real actions as it is given some cover story for Alice. As a result, regardless of what the adversary suspects Alice is doing (e.g., talking to a reporter from the Guardian), monitoring Vuvuzela provides only a limited improvement in the adversary's certainty of that suspicion (bounded by e^{ε} and δ).

Vuvuzela does not require users to explicitly specify a cover story; rather, the definition says that all user actions (both real and any possible "cover stories") will look about the same to an adversary. This covers *all* information that an adversary might learn about Alice's communications: not only whether she's talking to Bob, but whether she's communicating at all (or just running an idle client). This definition subsumes most other privacy guarantees that Alice might want in practice: distinguishing whether she is talking to Bob or Charlie, whether she has ever talked to a particular 100-person group, etc.

Vuvuzela operates in *rounds* during which each user can send and receive one message. Despite hiding a lot of information, Vuvuzela does allow an adversary to learn some information each round. Thus, the degree of privacy depends on how many rounds Alice participated in—or, more precisely, on the number of rounds in which her actions differ from her potential cover stories. We discuss this in more detail in §2.3. In practice, we usually configure Vuvuzela to provide $\varepsilon = \ln 2$ and $\delta = 10^{-4}$ for 200,000 rounds, which means that even after Alice exchanges 200,000 messages, an adversary should believe that the likelihood of Alice's cover story is within $2\times$ of the likelihood of what she actually did (unless the adversary gets lucky, with probability 10^{-4}, and learns a bit more).

Vuvuzela cannot hide the fact that a user is connected to the system. To limit the information disclosed by the fact that Alice connects to Vuvuzela, we recommend that users run the Vuvuzela client at all times. In principle, users are allowed to connect at any time, but if this correlates with information they are trying to hide, Vuvuzela cannot help. For instance, if Alice and Bob always start their Vuvuzela clients before their daily chat, and then promptly shut down their clients after, an adversary could infer that they are talking. On the other hand, if their Vuvuzela clients are running at all times, an adversary cannot learn when or with whom they are talking.

1.3 Threat model

Vuvuzela's design assumes an adversary who 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. [4] 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 dissertation, which is difficult even with these assumptions.

We further assume that honest Vuvuzela clients and servers faithfully implement the Vuvuzela protocol, and that there is no data leakage through side channels. Of course, some clients and servers may be controlled by an adversary (in which case, they need not follow the protocol), but honest clients and servers are assumed to be running bug-free implementations.

In terms of availability, Vuvuzela assumes that clients can misbehave, but that each server will handle requests properly. Any server can mount a denial-of-service (DoS) attack by blocking messages. This is unavoidable given our assumptions that the adversary can actively tamper with the network, and that at least one server is honest (thus, messages cannot bypass any server). However, even if an adversary mounts a DoS attack on Vuvuzela, the adversary would not learn any additional information (unless the users, as a result of the DoS attack, switch to a less-secure chat protocol).

[4] Because Vuvuzela does rely on cryptography, it only provides computational differential privacy [29], although we do not include a full formalization of such in this dissertation.

TWO

Hiding Metadata

To help understand Vuvuzela's design, this section provides a high-level overview of the system, and the next two sections will dive into the details of Vuvuzela's protocols.

Vuvuzela consists of a single chain of *servers* to which *clients* connect to communicate. We assume that the chain of servers, along with each server's public key, is known to clients ahead of time; all clients use the same chain. Clients always connect to the first server in the chain, which in turn connects to the second server, and so on.

Vuvuzela clients participate in two protocols. The first protocol, called the *conversation* protocol, allows a pair of users to exchange messages, assuming that they both decided to communicate with one another. The second protocol, called *dialing*, allows one user to request a conversation with another. When a client first connects to Vuvuzela, it starts by listening for incoming calls through the dialing protocol. When the client receives an incoming call, the user can choose to enter into a conversation with the caller, which enables them to exchange messages. Conversely, one can dial another user, and preemptively enter into a conversation with that user, in anticipation that user will reciprocate.

Protocol mechanics

Vuvuzela's two protocols communicate through *dead drops*: virtual locations on Vuvuzela's servers where one client deposits a message, and another client picks it up. Figure 2.1 gives an overview of Vuvuzela's dead-drop-based design. For example, to hold a conversation, two Vuvuzela clients agree on a randomly chosen *conversation dead drop* for each message

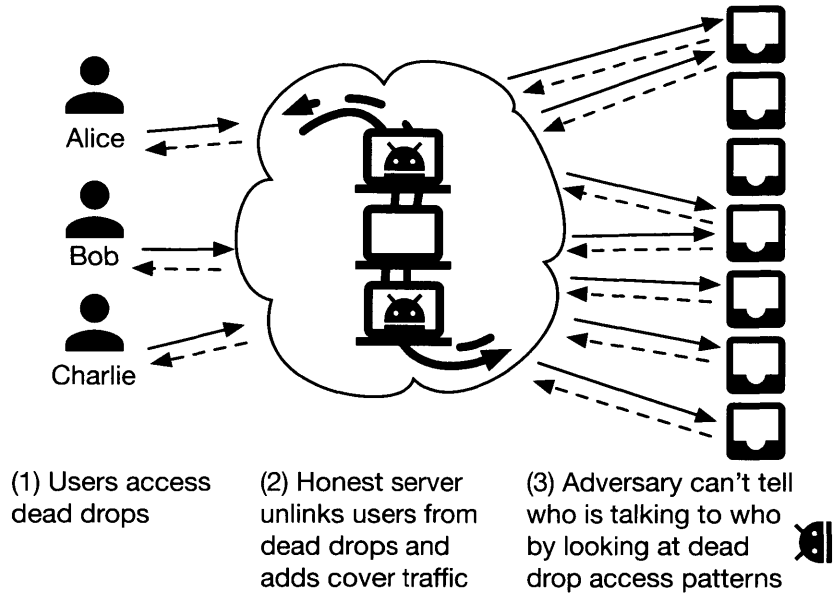

Figure 2.1: Overview of Vuvuzela's conversation protocol.

exchange. The two clients can now exchange messages by placing them in (and retrieving them from) this dead drop. Dead drops are named by 128-bit IDs, so honest clients should never collide in the dead drops they choose.

Similarly, each user (identified by the user's public key) is assigned to an *invitation dead drop*, based on a hash of the user's public key. This dead drop is shared with other users. Dialing a user thus requires placing a message into that user's invitation dead drop. Each user's client periodically polls its invitation dead drop, and checks if any of the messages there are for it. As we describe in §2.2, we will prevent an adversary from learning whether Alice is receiving invitations by adding cover traffic to invitation dead drops.

Vuvuzela's dead drops are ephemeral, meaning they do not persist over time. Instead, Vuvuzela works in synchronous *rounds*, each with a new set of dead drops. 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 declare what dead drop they want to access. The servers collect all of the requests in a given round, perform the accesses requested by clients (e.g., put a message into a dead drop, or get the contents

of a dead drop), and return the results to each client. There is no way to access a dead drop once the corresponding round is over. If a client temporarily goes offline, it might be unable to send a message in a particular round, or might miss a message meant for it; Vuvuzela deals with these issues through retransmission at a higher level (in the client itself). Vuvuzela's round-based design makes it difficult for an adversary to correlate dead drop accesses over time; for instance, the conversation protocol chooses a new pseudo-random dead drop for each round.

Achieving privacy

Building on the round-based dead drop design, Vuvuzela achieves privacy through a combination of *constant-bandwidth protocols*, *mixnets*, and *cover traffic*, as illustrated in Figure 2.1. In particular, Vuvuzela addresses three classes of attacks as follows; we discuss these defenses in more detail in §2.1 and §2.2, and formally analyze the resulting privacy in §2.3.

Network traffic. To limit the information that an adversary can learn by watching the network traffic between Vuvuzela clients and Vuvuzela servers, Vuvuzela encrypts all messages. Furthermore, Vuvuzela ensures that message sizes, and the rate at which messages are sent, are independent of user activity (via padding, splitting, etc). Vuvuzela clients also send messages at a fixed rate (queueing messages if the user types too fast, or generating empty messages if the user has not typed anything). One implication of this design is that there's a fixed number of conversations that a client can participate in per round (in our prototype, we set this to one).

Anonymizing dead drop accesses. Dealing with server compromises is a challenge in Vuvuzela. Dead drops are stored in memory on the last server in the chain, and all requests to this server are encrypted. However, we assume that any server—including this last server—could be compromised. This can be problematic if an adversary can determine which pair of users accessed a given dead drop.

To address this attack, Vuvuzela uses a mixnet approach. In particular, all requests are recursively encrypted under the public key of each server in Vuvuzela's chain. Each server is responsible for decrypting incoming requests, and randomly shuffling all of the

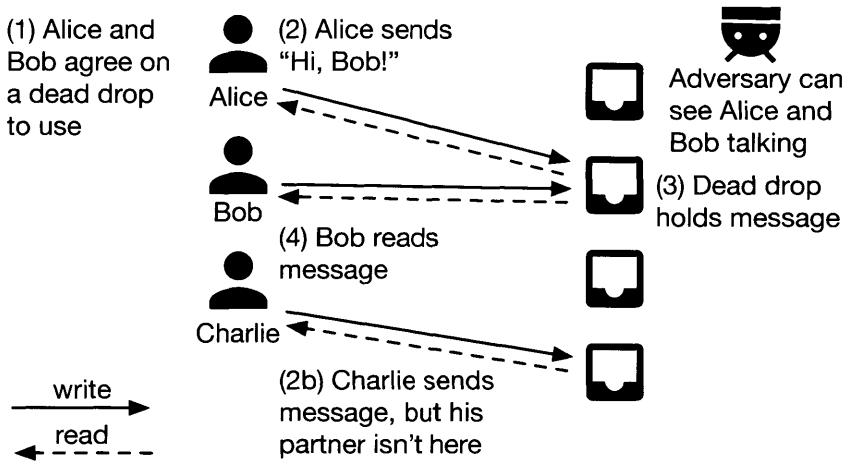

Figure 2.2: Strawman conversation protocol that does not hide information about which users accessed a given dead drop.

requests in a round before forwarding them to the next server. This design ensures that, if there is an honest server in the chain, an adversary cannot figure out which incoming request corresponds to an outgoing request, and thus prevents an adversary with access to the dead drops on the last server from learning *which* users accessed them.

Hiding dead drop access counts. Even with a mixnet, an adversary can still learn some information from just the *number* of dead drops that are accessed each round. For instance, an adversary might correlate the fact that a conversation seems to stop every time either Alice's or Bob's network disconnects. To obscure this information, Vuvuzela servers add *noise* requests—randomly generated requests that are indistinguishable from real user requests—to prevent statistical correlation attacks. As we show later, techniques from differential privacy allow us to precisely quantify the resulting level of privacy.

2.1 Conversation protocol

To understand the design of Vuvuzela's conversation protocol, consider a strawman version of the protocol shown in Figure 2.2, where users access dead drops by sending messages to a single server controlled by an adversary. In this protocol, clients issue just one kind of request—a message exchange—which deposits a message into a dead drop, and returns the other message (if any) that was deposited into the same dead drop in that round. While it forms the basis of the Vuvuzela protocol, this strawman allows an adversary to observe three variables:

1. Which users participated in each round.

2. Which users accessed each dead drop, which allows the adversary to link users to one another. (The adversary can compromise the server storing the dead drops to see this.)

3. How many messages were successfully exchanged each round, which is equivalent to how many dead drops were accessed twice in that round. This is a subset of the information that an adversary can derive from the above variables, but we list it separately because hiding it requires a different approach.

Vuvuzela's conversation protocol, shown in Algorithms 1 and 2, hides all but the last variable from the strawman design, and for the last variable, obscures it with enough cover traffic to provide a high degree of privacy. The rest of this section describes how Vuvuzela achieves this.

Hiding variables

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 2.2, Charlie performs an exchange with a random dead drop. To make sure that all exchange requests look the same to an adversary, all messages are padded to the same length and encrypted using an indistinguishable encryption scheme, and the last Vuvuzela server returns an empty message when it receives only one exchange for a dead drop.

Algorithm 1 Conversation round: client

Consider a user Alice, with public key pk_{alice} and secret key sk_{alice}. Each server i, ranging from 1 to n, has a public key pk^i_{server}. Alice's client takes the following steps for each round r:

1a. **Compute dead drop and encrypt message** (if Alice is in an active conversation with user Bob, whose public key is pk_{bob}): Using Diffie-Hellman, compute a shared secret $s_{n+1} = \text{DH}(sk_{\text{alice}}, pk_{\text{bob}})$. The dead drop will be $b = H(s_{n+1}, r)$, where H is a hash function. Pad and encrypt Alice's message m using nonce r and secret key s_{n+1} to get $e_{n+1} = (b, \text{Enc}(s_{n+1}, m))$. If Alice has not typed in any message this round, m is the empty message.

1b. **Construct fake request** (if Alice is not in an active conversation): Generate a random public key pk_{rand}, and let m be the empty message. Compute shared secret s_{n+1}, dead drop b, and resulting ciphertext e_{n+1} as above, with pk_{rand} instead of pk_{bob}.

2. **Onion wrap the request and send it to the servers**: Alice's client encrypts the request 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 request. 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's client gets back e'_1. If she is not in an active conversation, the result is irrelevant (it is an encryption of an empty message). If she is in an active conversation with Bob, Alice's client decrypts the layers: $e'_{i+1} \leftarrow \text{Dec}(s_i, e'_i)$. After decrypting e'_{n+1} and unpadding the message, Alice's client can display the message Bob sent her (if it's not the empty message).

Randomizing dead drop IDs. If Alice and Bob were to always use the same dead drop ID, then an adversary might correlate the fact that Alice and Bob are online with the fact that a particular dead drop is active. To ensure that an adversary cannot learn anything from the dead drops IDs accessed each round, Vuvuzela clients use a cryptographically secure pseudo-random number generator to generate a dead drop ID each round based on a shared secret and the round number. (Two users generate their shared secret based on their public keys.) This ensures that an adversary cannot learn any information from the dead drop IDs being accessed in a given round, and cannot correlate the dead drop IDs across rounds.

Algorithm 2 Conversation round: server

1. **Collect and decrypt requests**: Server i receives many encrypted requests of the form $e_i = (pk_i, \text{Enc}(s_i, e_{i+1}))$, either from clients or from the previous server in the chain. For each such request, the server computes the shared secret $s_i = \text{DH}(sk^i_{\text{server}}, pk_i)$, and then decrypts e_i.

2. **Generate cover traffic** (servers $i < n$): The server samples a random n_1 and n_2 from Laplace(μ, b) capped below at 0. Then, the server generates $\lceil n_1 \rceil$ individual accesses to random dead drops with random requests, and $\lceil n_2/2 \rceil$ pairs of accesses. These fake requests are added to the pool of requests for this round.

3a. **Shuffle the requests and send them to the next server** (servers $i < n$): The server computes a permutation π for this round, shuffles all the requests according to π, and sends them to the next server. After the next server returns the results, the server unshuffles them by applying the inverse permutation π^{-1}.

3b. **Process all requests and dead drops** (server $i = n$): The last server matches up all the accesses to each dead drop. For each pair of exchanges on the same dead drop, the server exchanges the contents of the requests and returns those.

4. **Encrypt results and return them**: Each resulting (non-noise) message e'_{i+1} gets encrypted to $e'_i = \text{Enc}(s_i, e'_{i+1})$, and all messages get returned.

Unlinking users from requests. To eliminate the observable connection between the sender of a message, the dead drop that the message is placed in, and the eventual recipient of the message, Vuvuzela employs a mixnet design. To make sure that exchange requests get mixed, each client encrypts their request 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 request r to form $E_{pk_1}(E_{pk_2}(E_{pk_3}(r)))$.[1] This onion construction ensures that the request r can be decrypted only if each server removes its encryption layer in turn. 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 requests to come in, decrypts its layer of encryption, and shuffles all the requests with a random permutation. Since obtaining r requires every server to decrypt the onion,

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

this means that at least one server will shuffle r along with all other requests in the same round (since we assume one server is honest and following the protocol).

This provides a strong degree of security. More precisely, if the honest server is the last one, then the adversary has no visibility into which dead drops are being accessed by users. If the honest server is one of the mixing servers (i.e., not the last server), the adversary cannot correlate the requests going into the honest server with the ones coming out, as the adversary is assumed not to have that server's private key. Consequently, a user cannot be linked to their dead drop requests 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 dead drop exchange.

2.1.1 Obscuring the number of messages exchanged

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

The problem lies in the one remaining observable variable: the histogram of dead drop access counts. While the dead drop IDs and dead drop contents are all indistinguishable from our adversary's perspective, some dead drops still look different. For example, in Figure 2.2, one dead drop is accessed twice in a round, while another dead drop is accessed just once. As we described in §1.1, this can provide valuable information by exposing the number of messages exchanged.

One possible attack with a mixnet design 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, he collects requests from all users at the first server, but then throws away all requests except those from Alice and Bob. Then, he passes these requests to the second server. The second server will mix the requests and send them to the third server. If the adversary controls the third

server, he 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 dead drop; otherwise, it will not.

An adversary who might not be willing to perform such invasive attacks could still learn a lot from dead drop access patterns. For example, the adversary can simply wait for Alice to go offline, and look at the difference in dead drop access counts between rounds. If the number of dead drops 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 dead drops that had one exchange request, and the number of dead drops that had two exchange requests. Beyond their number of accesses, the dead drops are cryptographically indistinguishable.[2]

To hide these last two variables, each server generates *cover traffic* requests that look like accesses to random dead drops. The server draws two samples, n_1 and n_2, from the distribution $\max(0, \text{Laplace}(\mu, b))$, and adds $\lceil n_1 \rceil$ requests that each access a dead drop once, and $\lceil n_2/2 \rceil$ pairs of requests that access the same dead drop. The server shuffles these requests along with the real ones before passing them to the next server, and removes them when results come back. §2.3 explains why we chose this distribution, and explains how to set μ and b.

Cover traffic adds random noise to the dead drop access counts at the last server. As a result, an adversary will no longer learn much by throwing away all requests except those from Alice and Bob, and an adversary will also no longer learn much when Alice goes offline, as the cover traffic hides those valuable small changes in the access counts.

Although cover traffic hides the exact number of dead drops accessed once or twice, an adversary can still tell roughly how many people are talking, by looking at the number of dead drops accessed twice and subtracting the average amount of noise. This meets our security goal, which is to prevent an adversary from learning information about a single individual.

[2] It is possible for a dead drop to get more than two exchanges if an adversary issues many exchanges for a dead drop. However, uncompromised users choose random dead drops, making the probability of a collision negligible. Thus, we focus on dead drop access patterns of uncompromised users; the adversary already knows the accesses by compromised users.

2.2 Dialing protocol

Vuvuzela's conversation protocol is useful for pairs of users that have agreed to talk to each other. However, users need a way to start conversations with new partners or restart conversations with previous partners (users can have a fixed number of conversations per round, so a user may end one conversation to make room for another). This is handled by Vuvuzela's dialing protocol. We expect that users use dialing each time they want to start a new conversation with a partner, even if they have talked to this partner before (but have since stopped exchanging messages through the conversation protocol).

In principle, combining the conversation and dialing protocols could improve privacy, by making conversation and dialing requests indistinguishable from one another. However, as we show in the rest of this section, dialing has significantly different message size requirements, which led to our decision to expose two distinct protocols in Vuvuzela.

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 adversary as possible, and to add the right amount of noise to those variables.

Unlike the conversation protocol, dialing cannot use random dead drop IDs, because users do not know which other users may wish to dial them. Instead, the dialing protocol uses a number of large *invitation dead drops*, as shown in Figure 2.3. Each such dead drop receives all invitations for a fixed set of public keys; with m invitation dead drops, public key pk's invitations are stored in dead drop $H(pk) \bmod m$, where H is a standard cryptographic hash function. Each user downloads all invitations from their dead drop (including noise invitations added as cover traffic) and tries to decrypt every invitation to find any that are meant for them. If a user wishes to accept a sender's invitation, the user simply starts the conversation protocol based on that sender's public key.[3]

[3] 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.

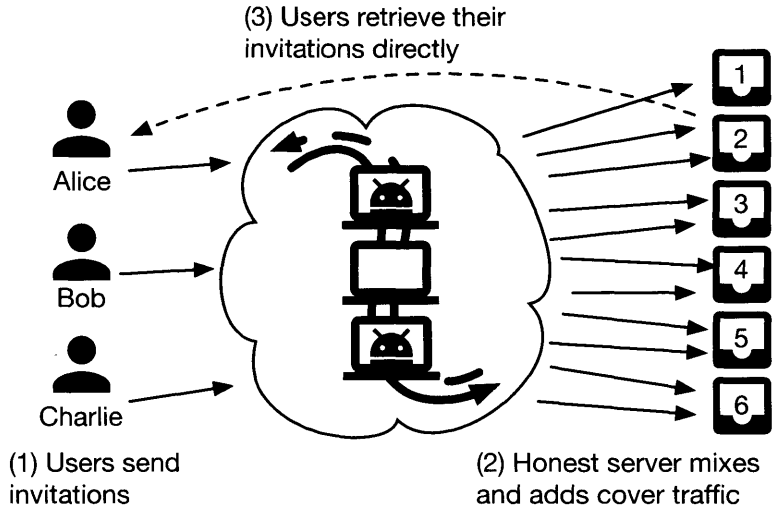

Figure 2.3: Overview of Vuvuzela's dialing protocol.

Similar to the conversation protocol, Vuvuzela hides three sets of variables from an adversary:

1. Which users participated in the protocol each round?
2. What dead drop did some sender add an invitation to?
3. Given a dead drop, how many invitations are in it (since the adversary can link recipients to dead drop IDs)?

As in the conversation protocol, Vuvuzela hides the first set of variables using fake invitations, the second set using a mixnet, and the third set by adding cover traffic.

However, there are three important differences from the conversation protocol. First, Vuvuzela does not hide which dead drop a client downloads; this is because Vuvuzela assumes the user's public key is well known, and thus the adversary knows the client's invitation dead drop. Second, the responses are variable length (since there can be a varying number of invitations in a dead drop). Third, the observable variable is now the number of invitations in a dead drop, rather than simply the histogram of dead drop access counts.

Even though the dialing protocol has more observable state that needs noising, we show that the noise required is manageable because invitations are smaller and less frequent than conversation messages. By carefully choosing the number of dead drops, Vuvuzela can also make the total noise proportional to the number of actual invitations; each client needs to download just one dead drop worth of noise.

Unlinking senders from invitations

Like conversations, dialing in Vuvuzela take place in rounds. Our prototype starts a new dialing round every 10 minutes, which translates into the latency of starting a new conversation.

For each round, the Vuvuzela servers fix a number m of invitation dead drops to create (a later section describes how to best set m). Then, each client chooses an invitation dead drop (e.g., the dead drop of a friend that the user wants to communicate with), and sends an invitation to it. If a user does not want to start a conversation in a given round, the client writes into a special no-op dead drop that is not used by any recipient. Each invitation 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 dead drop $H(pk) \bmod m$. Invitations are also onion-encrypted and shuffled, so that they are unlinked from their sender.

Hiding the number of invitations per dead drop

To achieve differential privacy, Vuvuzela must mask the number of invitations in each invitation dead drop; otherwise, an adversary could infer whether people are talking to Alice, based on the number of invitations in dead drop $H(pk_{\text{alice}}) \bmod m$. One crucial difference from the conversation protocol is that an adversary can observe the size of the invitation dead drop based on the sizes of responses (or simply by asking to download it, since invitation dead drops are shared by many users). Thus, *every* server (including the last one) must add a random number of noise invitations to *every* invitation dead drop.

Like in conversations, each server adds noise drawn from a truncated Laplace distribution $\lceil \max(0, \text{Laplace}(\mu, b)) \rceil$ to each invitation dead drop. The parameters μ and b can be different from the conversation protocol, as we discuss in §2.3.

Tuning the number of dead drops

In the dialing protocol, the amount of noise that needs to be added to each invitation dead drop turns out to be constant, based only on the desired security parameters and not on the number of users. However, the *number* of invitation dead drops, m, presents a trade-off between the amount of cover traffic that will be generated by the servers and the amount of data that will be downloaded by clients.

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

The value of m is purely an optimization: regardless of m, each user is protected by the level of noise, μ, added to their invitation dead drop. Thus, the last server (which hosts the invitation dead drops) can compute the optimal value of m as above, and propose this value of m for upcoming rounds to other servers (which have to generate noise for each of the m invitation dead drops). The first server then informs clients of the value of m for a given dialing round.

Downloading invitations

In Vuvuzela's dialing protocol, each dead drop contains a large amount of data (on the order of megabytes, as we show in §3), and each dead drop is downloaded by a large number of clients whose public keys map to that dead drop ID. This traffic can overwhelm Vuvuzela's servers, but at the same time, requests for downloading invitations do not need to be routed through Vuvuzela's servers, since they do not need to be mixed or noised. Thus, we envision that Vuvuzela could use a CDN or BitTorrent-like design to distribute the contents of invitation dead drops to clients.

2.3 Analysis

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

Observable variables

Recall from §1.2 that our goal is to provide (ε, δ)-differential privacy, where every event observed by our adversary is nearly equally likely if we change the actions of one user.

To understand what observations an adversary can make from one round, consider the client and server pseudocode shown in Algorithms 1 and 2. The client sends a fixed-size request regardless of who the user is communicating with, and the request is encrypted in an onion, using fresh Diffie-Hellman keys exchanged with each of the servers' public keys. The client also receives a fixed-size response from the servers, similarly encrypted in an onion with the same keys. Assuming that our encryption scheme is cryptographically secure, an adversary cannot learn anything from these requests and responses (except for the set of client machines that are connected to Vuvuzela), unless the adversary has also compromised some of the servers.

Suppose an adversary compromises all servers *except* the last server (which is responsible for storing the dead drops). In this case, the adversary learns no information. This is because the adversary cannot decrypt the last layer of the onion in the requests or responses (since the adversary does not have the key of the last server), and consequently, all requests and responses are indistinguishable, owing to their fixed size.

Suppose instead that an adversary compromises the last server. Then by our assumption, there must exist at least one honest server, which is not the last server. For the purposes of our security proof, we rely on this honest server to perform two functions: generating cover traffic (server step 2) and shuffling the requests and responses (server step 3a). Since the honest server shuffles the responses, and since our encryption scheme is secure, once the responses pass through the honest server, they become indistinguishable to any adversaries that have compromised the subsequent (earlier in the chain) servers. However, requests may provide the adversary with some information, as we now discuss.

Alice's cover story	Alice's real action: Idle	Conversation with b	Conversation with x
Idle	0, 0	−2, +1	0, 0
Conversation with b	+2, −1	0, 0	+2, −1
Conversation with c	+2, −1	0, 0	+2, −1
Conversation with x	0, 0	−2, +1	0, 0
Conversation with y	0, 0	−2, +1	0, 0

Figure 2.4: Difference in the number of dead drops with one access (Δm_1) and two accesses (Δm_2) between a user's (call her Alice) real action and her cover story, shown as $\Delta m_1, \Delta m_2$ in each table entry. b and c denote distinct other users in a conversation with Alice. x and y denote distinct other users not in a conversation with Alice.

The issue is that the adversary may have compromised all of the subsequent servers in Vuvuzela's chain, and thus may be able to trace each request. The adversary does not know who sent each request (due to shuffling), but we must consider what information can be learned from the request itself. The request's message payload is encrypted, and thus reveals no information. The dead drop ID is chosen at random, and is not reused across rounds, so the only possible information that the adversary learns is when the dead drop IDs from two different requests are equal. Since the dead drop ID space is large (128 bits), legitimate users will almost never choose the same dead drop IDs by accident; two users choose the same dead drop ID only if they are in an active conversation. Without loss of generality, we can ignore any requests generated by the adversary (which might use arbitrary dead drop IDs), since their contents gives the adversary no additional information.

Thus, the only variables the adversary can see are the set of users connected to the system, the number of dead drops that are accessed twice, and the number of dead drops that are accessed once. We now show that, based on these observable variables, the protocol can be made (ε, δ)-differentially private.

One round of conversations

Let m_1 and m_2 be the number of dead drops that are accessed once and twice in a given round, respectively. Figure 2.4 shows how the difference between the real actions of a user (call her Alice) and her cover story for that round would affect m_1 and m_2. The columns represent what Alice really did that round: she either (1) was idle, (2) performed a dead drop exchange with some user b who was likewise doing an exchange with Alice, or (3) performed an exchange with some user x who did *not* reciprocate the exchange. The rows describe her possible cover story: she (1) was idle, (2) exchanged messages with either the same user b or a different user c who reciprocated the exchange, or (3) did an exchange with the same user x or a different user y who did not reciprocate. In all cases, m_1 is affected by at most 2 and m_2 by at most 1.

Next we will show that the noise added to m_1 and m_2 in §2.1.1, which was generated with Laplace distributions capped below at 0, provides differential privacy. We begin by looking at the effect of this form of noise on a single variable r, then examine m_1 and m_2 together.

Lemma 1. Consider an algorithm M that adds noise $N_0 \sim \max(0, \text{Laplace}(\mu, b))$ to the result r of a query. Suppose r has sensitivity t. Then M is (ε, δ)-differentially private for parameters $\varepsilon = \frac{t}{b}$ and $\delta = \frac{1}{2}\exp\left(\frac{t-\mu}{b}\right)$.

Proof. Let $N \sim \text{Laplace}(\mu, b)$. Consider two query results r_1, r_2 such that $|r_1 - r_2| \leq t$. Theorem 3.6 from [19] gives for all sets of values S,

$$\Pr[r_1 + N \in S] \leq e^{\varepsilon} \Pr[r_2 + N \in S].$$

Let $z = \max(r_1, r_2)$. First, we consider the case where adding noise yields values $> z$ by restricting the previous inequality to all sets $T \subseteq (z, \infty)$,

$$\Pr[r_1 + N \in T] \leq e^{\varepsilon} \Pr[r_2 + N \in T].$$

Observe that the probabilities are 0 for $N \leq 0$, since $\forall x \in T$, $x > r_1$ and $x > r_2$. Thus, we can replace N with N_0,

$$\Pr[r_1 + N_0 \in T] \leq e^{\varepsilon} \Pr[r_2 + N_0 \in T].$$

Next, we consider the case where adding noise yields values $\leq z$:

$$\begin{aligned}
\Pr[r_1 + N_0 \leq z] &\leq \Pr[r_1 + N_0 \leq r_1 + t] \\
&= \Pr[N_0 \leq t] \\
&= \Pr[\text{Laplace}(\mu, b) \leq t] \\
&= \frac{1}{2}\exp\left(\frac{t-\mu}{b}\right) \\
&= \delta
\end{aligned}$$

Finally, we use these two cases to show that for any set of values S we have,

$$\begin{aligned}
\Pr[r_1 + N_0 \in S] &= \Pr[r_1 + N_0 \in S \cap (-\infty, z]] + \Pr[r_1 + N_0 \in S \cap (z, \infty)] \\
&\leq \delta + \Pr[r_1 + N_0 \in S \cap (z, \infty)] \\
&\leq \delta + e^{\varepsilon} \Pr[r_2 + N_0 \in S \cap (z, \infty)] \\
&\leq \delta + e^{\varepsilon} \Pr[r_2 + N_0 \in S]
\end{aligned}$$

Thus, M satisfies the definition of (ε, δ)-differentially privacy. □

Lemma 2. Consider an algorithm M that adds noise $N_1 \sim \lceil \max(0, \text{Laplace}(\mu, b)) \rceil$ to a query result $r \in \mathbb{Z}$ with sensitivity t. Then M is (ε, δ)-differentially private for parameters $\varepsilon = \frac{t}{b}$ and $\delta = \frac{1}{2}\exp\left(\frac{t-\mu}{b}\right)$.

Proof. We have shown in Lemma 1 the result for noise $N_0 \sim \max(0, \text{Laplace}(\mu, b))$ without the ceiling. By Proposition 2.1 in [19], the mechanism $\lceil r + N_0 \rceil$ is differentially private, because postprocessing the result of a differentially private function keeps it differentially private. For $r \in \mathbb{Z}$, $\lceil r + N_0 \rceil = r + \lceil N_0 \rceil = r + N_1$. Thus, M satisfies the definition of (ε, δ)-differentially privacy. □

Theorem 3. Consider the algorithm M that adds noise $N_1 \sim \lceil \max(0, \text{Laplace}(\mu, b)) \rceil$ to m_1 and $N_2 \sim \lceil \max(0, \text{Laplace}(\frac{\mu}{2}, \frac{b}{2})) \rceil$ to m_2. Then M is (ε, δ)-differentially private with respect to changes of up to 2 in m_1 and 1 in m_2, for $\varepsilon = \frac{4}{b}$ and $\delta = \exp\left(\frac{2-\mu}{b}\right)$.

Proof. By Lemma 2 we have that M is $(\varepsilon_1, \delta_1)$-differentially private with respect to changes of up to 2 in m_1 for,

$$\varepsilon_1 = \frac{2}{b} \quad \text{and} \quad \delta_1 = \frac{1}{2}\exp\left(\frac{2-\mu}{b}\right).$$

We also have that M is $(\varepsilon_2, \delta_2)$-differentially private with respect to changes of up to 1 in m_2 for,

$$\varepsilon_2 = \frac{2}{b} \quad \text{and} \quad \delta_2 = \frac{1}{2}\exp\left(\frac{1-\mu/2}{b/2}\right).$$

Theorem 3.16 in [19] says that we can compose two (ε, δ)-differentially private algorithms by adding their epsilons and deltas. That is, M is (ε, δ)-differentially private with respect to m_1 and m_2 for,

$$\varepsilon = \varepsilon_1 + \varepsilon_2 = \frac{2}{b} + \frac{2}{b} = \frac{4}{b}$$

$$\delta = \delta_1 + \delta_2 = \frac{1}{2}\exp\left(\frac{2-\mu}{b}\right) + \frac{1}{2}\exp\left(\frac{1-\mu/2}{b/2}\right) = \exp\left(\frac{2-\mu}{b}\right).$$

□

This theorem gives ε and δ in terms of the parameters μ and b of our noise, but we can also use it to compute the μ and b needed for a target level of privacy:

$$b = 4/\varepsilon \qquad \mu = 2 - \frac{4\ln\delta}{\varepsilon} \tag{1}$$

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 property that the average amount of noise μ needed for a given ε and δ grows only with $\sqrt{k}$:

Theorem 4. Consider the algorithm M that adds noise $N_1 \sim \lceil \max(0, \text{Laplace}(\mu, b)) \rceil$ to m_1 and $N_2 \sim \lceil \max(0, \text{Laplace}(\frac{\mu}{2}, \frac{b}{2})) \rceil$ to m_2 over k Vuvuzela rounds. Then M is (ε', δ')-differentially private with respect to the actions of one user in these rounds 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 3.[4]

Proof. Direct application of Theorem 3.20 in [19]. □

Equation 1 shows that μ is proportional to $1/\varepsilon$ and only logarithmically dependent on δ. Theorem 4 shows that, to support a given ε' and δ', the per-round δ must shrink proportionally to k and the per-round ε must shrink proportionally to $\sqrt{k}$. This means the per-round μ grows proportionally to $\sqrt{k}$.

Although this theorem considers k rounds, it can also be applied in cases when the user was running the Vuvuzela client for more than k rounds. In that case, Theorem 4 still provides (ε', δ')-differential privacy for any cover story that differs from the user's real actions in at most k rounds. This allows a user that is idle a significant fraction of the time to extend Vuvuzela's privacy guarantees to many more rounds, by having the cover story match the real actions during idle rounds.

[4] d is a free parameter that lets one trade off between ε' and δ'.

Noise needed in practice

To set the level of noise (determined by μ and b) in practice, we need to know the values of ε and δ that will be acceptable to users of an (ε, δ)-differentially private algorithm.

In differential privacy, ε gives a multiplicative change in the probability of each event based on the user's actions, while δ gives an additive change. Usually, ε is recommended to be set between 0.1 and $\ln 3$ [18], and δ should be small, e.g., 10^{-4}. The multiplicative ε provides *plausible deniability*: the probability of observing some event when two users are talking is within e^{ε} times the probability of making the same observation had the users been doing something else. This means that users always have a plausible cover story that is within $1.1\times$ the probability of the real story for $\varepsilon = 0.1$ and within $2\times$ for $\varepsilon = \ln 2$. In contrast, δ covers events that might have *zero* probability under some actions but happen under others. For example, if the number of dead drops accessed twice is 0 in some rounds after adding noise, the adversary will know for sure that no users communicated. This δ arises in Vuvuzela because we cannot "subtract" noise. However, a low δ ensures such events are extremely unlikely.[5]

As a concrete example, consider an adversary Eve who believes that two users, Alice and Bob, might be talking using Vuvuzela. In general, Eve will already have a certain prior belief that Alice and Bob are talking, say $p_{\text{prior}} = 50\%$. We can apply Bayes' rule to compute Eve's posterior belief that the two users are talking based on any observation in Vuvuzela. With $\varepsilon = \ln 2$, Eve's posterior belief increases to 67%. With $\varepsilon = \ln 3$, it goes up to 75%. In any case, we see that observing Vuvuzela can aid Eve, but does not provide damning evidence even with high ε. If p_{prior} were smaller, Eve's posterior probability would increase by a larger factor, but not more than e^{ε}. For example, if $p_{\text{prior}} = 1\%$ and $\varepsilon = \ln 3$, it would increase to 3%, but this probability is still small.

[5] In a large population of suspected users, some individuals *might* have metadata revealed under this definition. However, the extra risk per user is likely negligible compared to other security risks that whistleblowers or reporters already face, and it is fairly inexpensive to reduce δ (the average amount of noise needed grows only logarithmically with $1/\delta$).

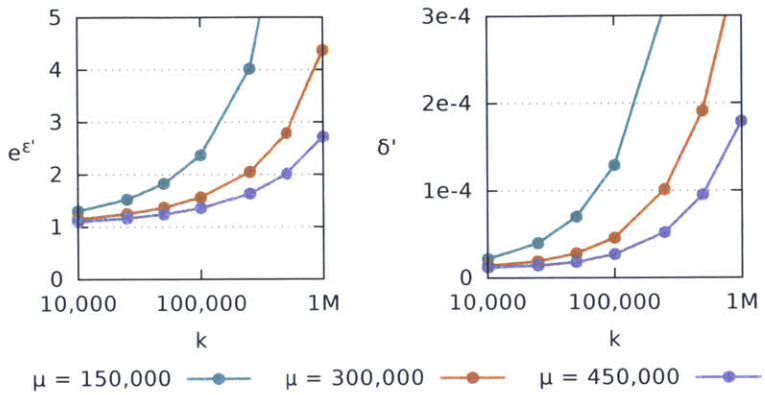

Figure 2.5: Values of ε' and δ' after k conversation rounds with the three noise distributions considered in the text. We plot $e^{\varepsilon'}$ instead of ε' so that you can easily see the level of deniability.

Given this background, we plot ε' and δ' as functions of the number of rounds k for three different distributions of noise in Figure 2.5:

Distribution	μ	b
n_1	150,000	7,300
n_2	300,000	13,800
n_3	450,000	20,000

Each distribution is a Laplace distribution with mean μ (the average noise added by each server) and standard deviation $\sqrt{2}b$.

We chose these distributions using Theorems 3 and 4 as follows. For each μ value, we tried to find the parameters that would achieve differential privacy with $\varepsilon' = \ln 2$ and $\delta' = 10^{-4}$ for the most number of rounds k. First, we set d in Theorem 4 to 10^{-5} in order to get δ' close to our target (neither ε' nor δ' is very sensitive to d). Then, for each μ, we used a parameter sweep to compute b such that k is maximized for our target ε' and δ'. [6] We see that in all cases, it is possible to support a fairly large number of rounds at $\varepsilon' = \ln 2$ and $\delta' = 10^{-4}$: this number of rounds is 70,000 for n_1, 250,000 for n_2, and 500,000 for n_3. In addition, both ε' and δ' change smoothly with different k.

[6] In general, δ' grows with b and ε' falls with it.

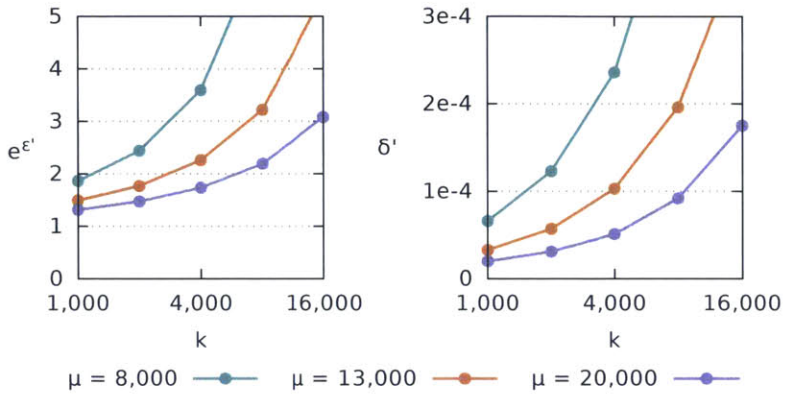

Figure 2.6: Values of ε' and δ' after k dialing rounds with the three noise distributions considered in the text.

Finally, from Theorems 3 and 4, we can also derive how the mean noise μ required to meet a given ε' and δ' scales with each parameter. We see that:

- μ increases proportionally to $\sqrt{k}$.
- μ increases linearly with $1/\varepsilon'$.
- μ increases proportionally to $\log(1/\delta')$.
- μ is independent of the total number of users.

Dialing protocol

In Vuvuzela's dialing protocol, we need to add noise to every dialing dead drop 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 minutes.
2. Dialing is less common than conversation messages, decreasing the values of k we might worry about.
3. Invitations are shorter than messages.

The analysis for dialing is similar to Theorems 3 and 4, except that modifying each user's action in a round only changes up to two dead drop 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 the conversation protocol for a given ε' and δ'. Furthermore, in the dialing protocol, k represents the number of invitations the user sends, so it can likely be small compared to the k used for conversations (e.g., a user who makes 5 calls per day only needs $k = 1800$ for one year).

Figure 2.6 plots ε' and δ' in dialing for different levels of noise using the same methodology as Figure 2.5:

Distribution	μ	b
n_1	8,000	500
n_2	13,000	770
n_3	20,000	1,130

We achieved differential privacy with $\varepsilon' = \ln 2$ and $\delta' = 10^{-4}$ for 1,200 rounds with noise distribution n_1, 3,500 rounds with n_2, and 8,000 rounds with n_3.

THREE

Evaluation

To evaluate Vuvuzela's design, we implemented a prototype of Vuvuzela in Go. Our prototype consists of approximately 3,000 lines of code. The source code is available at:

`https://github.com/davidlazar/vuvuzela`.

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.

There are a few differences between our prototype implementation and the design described in this dissertation. First, we implement an additional *entry server*, whose job is to handle a large number of connections from clients, multiplex client requests into a single round that's sent to the chain of Vuvuzela servers, and to demultiplex the results to individual clients. The entry server is not trusted. Second, we have not yet implemented either the client-side retransmission logic, or the adaptive choice of the number of invitation dead drops (at the scale we are operating Vuvuzela in our experiments, the optimal number of introduction dead drops is one). Finally, we have not implemented CDN- or BitTorrent-based distribution for the dialing protocol.

Our evaluation quantitatively answers the following questions:

- Can Vuvuzela servers support a large number of users and messages? (§3.1)
- Does Vuvuzela provide acceptable performance? (§3.2)

Experimental setup

To answer the above questions, we run a series of experiments on Amazon EC2 virtual machines (VMs). 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 1.5.

We use the following parameters across our experiments. Our chain consists of 3 Vuvuzela servers, each corresponding to one VM. An additional VM runs the entry server. Conversation messages are 256 bytes long (including 16 byte encryption overhead). Invitations are 80 bytes long (including 48 bytes of overhead). To ensure that clients are not the bottleneck, we use an additional five VMs to simulate user clients, and we multiplex several Vuvuzela clients onto a single TCP connection to the entry server, to avoid running out of source TCP port numbers. Every simulated user sends a message each conversation round to another user (although Vuvuzela's performance is the same regardless of whether users are actively communicating or are idle). Each dialing round, 5% of the users dial another user. Since we have not implemented a CDN/BitTorrent for downloading the dialing dead drops, only 100 clients fetch their dialing dead drop. This lets us estimate the dialing latency, and we extrapolate the bandwidth needed for distributing the dialing dead drops to all clients. We pick $\mu = 300{,}000$ for the conversation protocol and $\mu = 13{,}000$ for the dialing protocol. Finally, to not let noise affect the clarity of the graphs, we configure servers to always add exactly μ noise, rather than sampling the Laplace distribution; this produces the same average results with less variance.

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 is largely dominated by the CPU cost of cryptography on the servers and by the bandwidth for transferring all of the encrypted requests in a round.

3.1 Server performance

To evaluate whether Vuvuzela can support many users and messages, we measure the end-to-end latency and throughput of Vuvuzela's conversation and dialing protocols when

faced with anywhere from ten users to two million users. With one million users, our prototype achieves a throughput of approximately 68,000 conversation messages per second. We then evaluate the underlying costs behind Vuvuzela's performance.

Conversation protocol. Figure 3.1 shows the total latency of a conversation round, with the number of online users ranging from ten users 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. As mentioned in §2.3, the cover traffic required for Vuvuzela's conversation protocol is constant: the amount for ten users is the same as for two million users. The baseline time to process cover traffic can be seen with ten users (20 seconds end-to-end latency). Even though there are only ten real users and messages, Vuvuzela servers must process an entire round worth of requests at once, so the latency is dominated by the noise requests introduced by Vuvuzela servers. Each server in the chain, except for the last one, adds $\mu \times 2$ noise requests on average, for a total of 1.2 million requests when there are no users. With 2 million users, each adding one request, we get 3.2 million messages on the right side of the graph, for an end-to-end latency of 55 seconds (and a throughput of 84,000 messages/second). This demonstrates that Vuvuzela's costs scale linearly with the number of requests processed.

Dialing protocol. Figure 3.2 shows the end-to-end latency for Vuvuzela's dialing protocol. Here, we have 5% of users dialing another user each round, and the other 95% of users are not actively dialing (and thus their client sends a dialing request to the special idle dead drop). Like the conversation protocol, the dialing protocol scales linearly, from 13 seconds with ten users to 50 seconds with two million users.

Dominant costs. Most of the CPU time on Vuvuzela servers is spent wrapping and unwrapping of encryption layers. Each 36-core machine can perform about 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 3.2 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

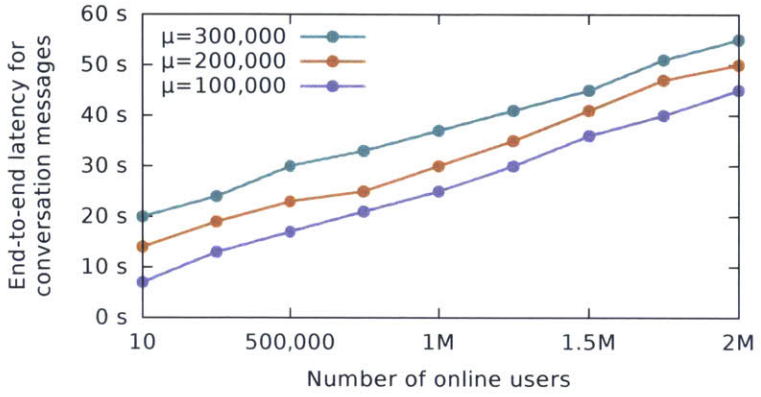

Figure 3.1: Performance of Vuvuzela's conversation protocol when varying the number of users online. Every user sends a message every round.

the best-case end-to-end conversation round latency would be $(3.2 \cdot 10^6 \times 3)/(3.4 \cdot 10^5) \approx$ 28 seconds. This shows that Vuvuzela's full protocol (serialization, shuffling, cover traffic generation, etc), is within 2× 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.

Vuvuzela servers also require a significant amount of bandwidth. With 1M users, servers use an average of 166 MB/sec (excluding invitation dead drop downloads). This is dominated by the total size of conversation messages (message exchange requests and responses) from real users and from server-generated cover traffic, with RPC and encoding overhead. Vuvuzela's server bandwidth requirements are comparable to any other messaging system with the same number of users and messages (albeit where one server must be capable of processing all messages).

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 (since each message must travel through more servers) and increases the number of messages each server has to process each round (due to cover traffic from each previous server). Figure 3.3 shows total end-to-end latency for different numbers of servers. Performance scales roughly quadratically with the number of servers in the chain. This is to be expected, since each of

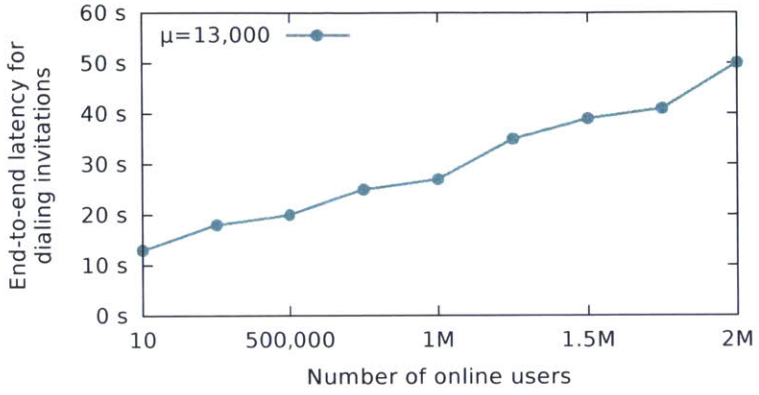

Figure 3.2: Performance of Vuvuzela's dialing protocol when varying the number of users online. 5% of the users dial someone every round. The conversation protocol is running concurrently with μ=300,000.

the 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.

3.2 Client performance

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, which means a client must wait on the order of 10 minutes to start a conversation. This makes Vuvuzela well-suited for slower-paced, e-mail-like, communication patterns where users queue up messages to send. We could increase the frequency of dialing rounds (the servers complete a dialing round in less than a minute), at the cost of increasing required client bandwidth. Conversations themselves move fairly quickly, with sub-minute end-to-end latencies for 240-byte text messages, even with 2 million active users. Clients can pipeline conversation messages, sending a new message every round even before receiving responses from previous rounds; in our experiment with 1M users, this amounts to a throughput of 4 messages per minute per client.

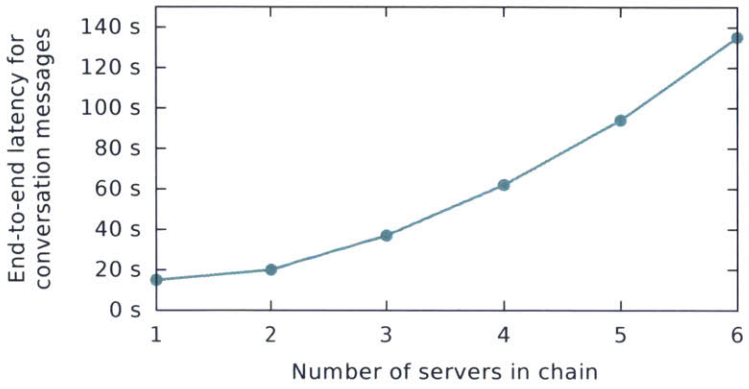

Figure 3.3: Performance of Vuvuzela's conversation protocol when varying the number of servers with 1 million active users and μ=300,000.

Bandwidth usage. For the conversation protocol, each client sends and downloads a 256-byte message per round (tens of seconds). 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 dead drop worth of invitations. With μ = 13,000 and 3 servers, that comes out to about 39,000 noise invitations, in addition to any real invitations (for instance, 50,000 real invitations if there are 1M users and 5% of them are dialing any given round). This adds up to a total of about 7 MB per round. With 10-minute rounds, a client uses an average of 12 KB/sec for downloading invitations. While not insignificant, Vuvuzela's design crucially avoids downloading the noise invitations generated by the rest of the users that are not dialing in a given round, since they are directed to a separate "idle" invitation dead drop (in our example, this would be 950,000 more noise invitations).

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 keep up with the required tens of kilobytes per second of bandwidth (although bandwidth charges may be prohibitive, depending on the user's service agreement).

FOUR

Research Prospects

Even though Vuvuzela provides strong guarantees regarding the inferences an adversary can make about each user, the system still has limitations that require care. We hope that some of these limitations can be addressed in future research.

Bandwidth use. Vuvuzela's fixed chain of servers enables a simple analysis of Vuvuzela's privacy guarantees, but translates into significant bandwidth requirements for each server (since every server must process every message). 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 privacy.

CPU use. Another side-effect of Vuvuzela's centralized design is that each server needs significant CPU power to sustain high message throughput. Assuming the network is not saturated, one way to increase throughput is to add more cores to each server. Alternatively, the Vuvuzela server algorithm can be sharded across multiple machines, but extra care must be taken to ensure that at least one cluster of servers is honest. Perhaps the most promising approach to eliminating CPU as a bottleneck in Vuvuzela's design is to use GPUs for the Diffie-Hellman operations [27].

Deployment costs. A significant roadblock to a practical deployment of Vuvuzela is the bandwidth cost incurred by every server, as mentioned above. Using Amazon's

EC2 prices as of September 2015, a Vuvuzela server would cost about $10,000/month, dominated by bandwidth costs, although bulk bandwidth prices may be about an order of magnitude lower [20]. Relying on volunteers to run Vuvuzela servers seems infeasible. On the other hand, if the cost of running Vuvuzela servers is amortized over 1M users, it comes out to less than $1 per year per user. Whether such a business model would work in practice is outside of the scope of this dissertation.

Treating users as noise. Vuvuzela's use of differential privacy is conservative: to ensure privacy for two users, Alice and Bob, that might or might not be communicating, Vuvuzela assumes that the adversary might control (or know everything about) every other user in the system. This forces Vuvuzela to add a significant amount of noise in order to mask the information about whether Alice and Bob are communicating. In a system with many users, it may be reasonable to assume that some fraction of users are honest (i.e., the adversary does not know what they are doing), formalized with the help of coupled-worlds privacy [3] or noiseless database privacy [5]. This could allow Vuvuzela to achieve its security goals while adding less cover traffic.

PKI for dialing. Vuvuzela's dialing protocol requires a public key infrastructure in two situations. First, 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; this 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 who compromises a user's private key (and saves old invitations, which are publicly accessible) 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 each round, and existing techniques can achieve forward secrecy for message contents [33].

It may be possible to achieve forward secrecy for dialing by using IBE [6], with each Vuvuzela server running an independent private key generator that generates a new master key every day. Another approach would be to rely on a forward-secure public-key encryption scheme [8].

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 [19]. For example, if an adversary suspects that a group of 1,000 people communicate frequently with each other, he can block all other users from the system. If the adversary now observes a significant number of dead drops being accessed twice, it would confirm his suspicion. However, he cannot distinguish whether *any specific individual* in the isolated group is actually communicating.

Denial of service attacks. As mentioned in §1.3, Vuvuzela's availability should be resilient to misbehaving users—e.g., users that send many requests or open many connections. Since all clients must communicate with Vuvuzela's entry server, the entry server can mitigate client DoS attacks through existing approaches: requiring users to sign up for an account, requiring proof of an account on other systems (e.g., Facebook), proof-of-work, or even payment. 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.

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.

FIVE

Related Work

Recent work has shown that secure messaging systems can provide end-to-end encryption at scale, starting from systems such as OTR [7, 23], Axolotl [33], and TextSecure [32], and now being deployed by WhatsApp [31]. 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 [26] 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. Research on anonymous communication also aims to hide metadata. Chaum's early work on mixnets [9] and DC-nets [10] laid the foundations for anonymous communication systems. Unlike Vuvuzela, however, previous systems do not simultaneously provide both scalability and protection against traffic analysis.

Mixnet-style systems [9, 16, 21], Crowds [34], Freenet [12], and onion routing [17, 38] 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 [14, 30] or by actively delaying some users' packets to see the effect on others [1]. In principle, cover traffic can help defeat traffic analysis, but it is difficult to determine how much cover traffic is enough (and the design of these systems is not amenable to reasoning about the privacy guarantees provided by cover traffic); this is precisely the problem addressed by Vuvuzela.

On the other hand, systems with provably strong security guarantees have relied on mechanisms that scale quadratically in the number of users. Herbivore [22] and

Dissent [39] 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). P5 [37] is another broadcast-based system that lets users trade-off anonymity for communication efficiency, but does not protect against a global active adversary. Riposte [13] 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 [11], such as the Pynchon Gate [36], 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 [4, 21], or by delaying messages [25]. However, it has been shown that these schemes still reveal information after multiple rounds of observation [28]. 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* existing anonymous communication schemes. However, to our knowledge, none have designed new schemes that minimize the amount of noise required, and none provide strong privacy over many rounds with similar performance. One key insight in Vuvuzela is that techniques used to *ensure* differential privacy, namely, adding Laplace noise, can be applied in practice to messaging systems (as long as the systems minimize the number of observable variable, so that reasonable amounts of noise can protect users over many rounds).

AnoA [2] offers a theoretical framework for formalizing the privacy of protocols, but limits the analysis to one round, and does not say how one might achieve privacy in practice. In contrast, Vuvuzela's formalization captures many rounds, and Vuvuzela presents a new design that achieves strong privacy guarantees in a practical system.

Danezis [15] uses differential privacy over multiple rounds in a mixnet. However, he does not study how to reduce the amount of information leakage each round to make strong levels of privacy possible over hundreds of thousands of rounds.

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 who 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.

Bibliography

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

[2] Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar 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] Raef Bassily, Adam Groce, Jonathan Katz, and Adam Smith. Coupled-worlds privacy: Exploiting adversarial uncertainty in statistical data privacy. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, Berkeley, CA, October 2013.

[4] Oliver Berthold and Heinrich Langos. Dummy traffic against long term intersection attacks. In *Proceedings of the Workshop on Privacy Enhancing Technologies*, pages 110–128, Dresden, Germany, March 2003.

[5] Raghav Bhaskar, Abhishek Bhowmick, Vipul Goyal, Srivatsan Laxman, and Abhradeep Thakurta. Noiseless database privacy. In *Proceedings of the 17th Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 215–232, Seoul, South Korea, December 2011.

[6] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, 2001.

[7] Nikita Borisov, Ian Goldberg, and Eric 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, October 2004.

[8] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *Journal of Cryptology*, 20(3):265–294, July 2007.

[9] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), February 1981.

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

[11] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, November 1998.

[12] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore 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.

[13] Henry Corrigan-Gibbs, Dan Boneh, and David 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.

[14] George 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.

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

[16] George Danezis, Roger Dingledine, and Nick 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.

[17] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Usenix Security Symposium*, pages 303–320, San Diego, CA, August 2004.

[18] Cynthia Dwork. Differential privacy: a survey of results. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*, Xi'an, China, April 2008.

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

[20] Drew Fitzgerald. Bulk bandwidth prices get steadier after long swoon. *Wall Street Journal*, September 2014. http://www.wsj.com/articles/bulk-bandwidth-prices-get-steadier-after-long-plunge-1411602100.

[21] Michael J. Freedman and Robert 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, November 2002.

[22] Sharad Goel, Mark Robson, Milo Polte, and Emin G. Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical Report 2003-1890, Cornell University, Computing and Information Science, Ithaca, NY, 2003.

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

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

[25] Dogan Kesdogan, Jan Egner, and Roland 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, April 1998.

[26] Adam Langley. Pond, 2015. https://pond.imperialviolet.org/.

[27] Eric M. Mahé and Jean-Marie Chauvet. Fast GPGPU-based elliptic curve scalar multiplication, 2014. http://eprint.iacr.org/2014/198.

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

[29] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In *Proceedings of the 29th Annual International Cryptology Conference (CRYPTO)*, pages 126–142, Santa Barbara, CA, August 2009.

[30] Steven J. Murdoch and George 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.

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

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

[33] Trevor Perrin and Moxie Marlinspike. Axolotl ratchet, 2014. https://github.com/trevp/axolotl/wiki.

[34] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[35] Alan Rusbridger. The Snowden leaks and the public. *The New York Review of Books*, November 2013.

[36] Len Sassaman, Bram Cohen, and Nick Mathewson. The Pynchon gate: A secure method of pseudonymous mail retrieval. In *Proceedings of the Workshop on Privacy in the Electronic Society*, Arlington, VA, November 2005.

[37] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A protocol for scalable anonymous communication. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Washington, DC, 2002.

[38] Paul F. Syverson, David M. Goldschlag, and Michael 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.

[39] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the 10th Symposium on Operating Systems Design and Implementation (OSDI)*, Hollywood, CA, October 2012.