A Distributed Metadata-Private Messaging System

by

Nirvan Tyagi

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

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2016 [June 2016]

Signature redacted

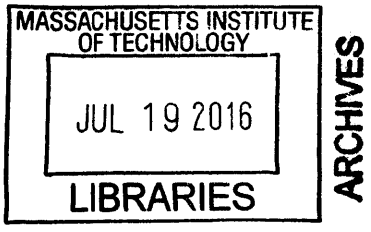
Author .....
Department of Electrical Engineering and Computer Science
May 20, 2016

Signature redacted

Certified by..... .....
Dr. Matei Zaharia
Assistant Professor
Thesis Supervisor

Signature redacted

Accepted by..... .....
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee



A Distributed Metadata-Private Messaging System

by

Nirvan Tyagi

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2016, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Private communication over the Internet continues to be a difficult problem. Even if messages are encrypted, it is hard to deliver them without revealing *metadata* about which pairs of users are communicating. Scalable systems such as Tor are susceptible to traffic analysis. In contrast, the largest-scale systems with metadata privacy require passing all messages through a single server, which places a hard cap on their scalability.

This paper presents Stadium, the first system to protect both messages and metadata while being able to scale its work efficiently across multiple servers. Stadium uses the same differential privacy definition for metadata privacy as Vuvuzela, the currently highest-scale system. However, providing privacy in Stadium is significantly more challenging because distributing users' traffic across servers creates more opportunities for adversaries to observe it. To solve this challenge, Stadium uses a novel verifiable mixnet design. We use a verifiable shuffle scheme that we extend to allow for efficient group verification, and present a verifiable distribution primitive to check message transfers across servers. We show that Stadium can scale to use hundreds of servers, support an order of magnitude more users than Vuvuzela, and cut the costs of operating each server.

Thesis Supervisor: Dr. Matei Zaharia
Title: Assistant Professor

Acknowledgments

This work would not have been possible without contributions from Yossi Gilad, Justin Martinez, Pratheek Nagaraj, and David Lazar. I am especially grateful for the guidance of my advisors, Prof. Matei Zaharia and Prof. Nickolai Zeldovich.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Private communication is difficult in today's Internet. There are many applications that allow users to encrypt their message content. However, encryption does not hide *metadata*: adversaries can still learn who is communicating with whom, at which times, and their traffic volumes. Indeed, NSA officials have stated that "if you have enough metadata you don't really need content," [23] and "we kill people based on metadata" [18]. For users such as reporters, whistleblowers, and activists, achieving higher privacy guarantees is critical.

Unfortunately, previous systems that hide metadata do not scale to large numbers of users. Systems that leak no metadata at all, such as Riposte [8] and Dissent [9], require broadcasting all messages to all users or use computationally intensive Private Information Retrieval. Recently, Vuvuzela [27] introduced an approach for protecting metadata based on differential privacy [12]. That is, each time Alice sends a message to Bob, the adversary gains some statistical information by monitoring observable variables in the system (e.g., the traffic patterns between servers). Vuvuzela provably bounds this information by injecting noise messages that obfuscate these variables, so that all of the adversary's observations would be almost equally likely if Alice were not talking with Bob. Using this definition, Vuvuzela scales to support about 84,000 messages per second. However, its design requires transmitting all messages through a single chain of relay servers, preventing it from scaling beyond this. Moreover, at this peak rate, each Vuvuzela server needs about 1.3 Gbit/sec of bandwidth, leading

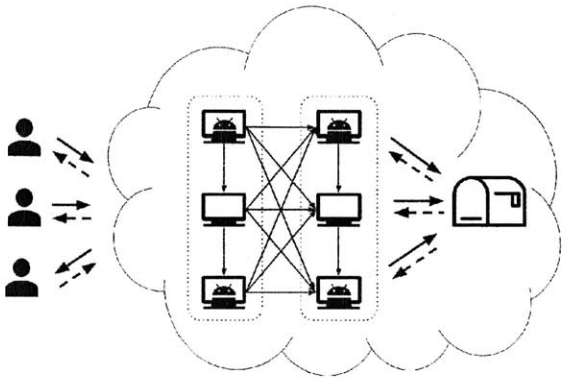


Figure 1-1: High level overview of Stadium system.

to a high operating cost for each server [27, §8.2]. Both of these problems are serious roadblocks to deploying Vuvuzela in practice.

This paper presents Stadium, the first system to provide metadata and data privacy while distributing its work scalably over multiple servers. Stadium uses the same differential privacy approach as Vuvuzela, but scales to support more users. While the total cost of running Stadium is higher than Vuvuzela, more individuals can operate servers at a much lower cost per server, and the system can scale incrementally to provide an order of magnitude higher throughput in total.

Stadium builds on a parallel mixnet design [17] where users select a fixed-length path (chain of servers) for each message through the available servers. This lets Stadium scale nearly linearly with the number of servers. However, the key challenge in Stadium is protecting the much higher number of observable variables its paths expose. Unlike in Vuvuzela, the majority of servers need to generate noise. Moreover, malicious servers can *discard* the noise generated by other servers, or modify (e.g., duplicate) user messages to learn about communication patterns. To address these problems, Stadium introduces a novel *verifiable parallel mixnet* design, which is based on two components. First, each server shuffles messages locally using verifiable shuffles [6]. To efficiently let multiple other servers audit these shuffles, we introduce a *group verification* protocol that requires each server to only generate one proof for its shuffle. Second, global shuffling also requires verifying the distribution of messages

to multiple servers. For this purpose, we introduce a *verifiable distribution* scheme that lets multiple servers audit each distribution step.

We implement a prototype of Stadium on Amazon EC2. We show that Stadium scales nearly linearly to hundreds of servers, enabling an order of magnitude improvement in throughput. Although Stadium's total bandwidth costs are higher than Vuvuzela, they get distributed evenly across the servers, lowering the cost of hosting a server. For example, in today's Tor network, only 5% of relays offer more than 100 Mbps of bandwidth, and none offer more than 1 Gbps [26]. With 100 Mbps per bandwidth, Stadium can scale to over 500,000 messages per second over hundreds of servers, while Vuvuzela is limited to around 8000 messages per second (though Vuvuzela is not designed to be deployed in this highly distributed Tor-like approach). With higher bandwidth per server, Stadium also continues to scale higher than Vuvuzela. Our design does have some limitations—in particular, it ceases to be available if servers crash. However, even in these cases, Stadium continues to guarantee privacy.

Finally, we formally analyze the security of Stadium using differential privacy. We show that Stadium can provide similar guarantees to Vuvuzela (continued privacy even if a user sends tens of thousands of messages over the system), even in the presence of active adversaries.

To summarize, our contributions are:

- The first design for a metadata-private messaging system that can scale work efficiently across hundreds of servers. Our design enables order of magnitude higher throughput than prior systems.
- A novel verifiable parallel mixnet design based on efficient protocols for group verification of shuffles and for verifiable distribution.
- An analytical and experimental evaluation of Stadium, analyzing its security and scalability.

Chapter 2

Threat Model

Stadium assumes an adversary that controls some fraction of its mixing servers, the system may be deployed to resist different fractions of compromised servers (but its efficiency deceases as their assumed fraction grows). The attacker may also control any number of users. We further assume that non-compromised servers and clients run bug-free implementations of the Stadium protocol and are not vulnerable to side-channel attacks, but adversary controlled servers and clients may deviate in any way from the protocol. We design Stadium to resist passive and active attackers, that is, we allow the attacker to monitor, block, delay, or inject traffic on any network link at all communication rounds.

In terms of availability, Stadium is not resistant to denial of service attacks. This is unavoidable given our assumption that adversaries can block traffic, which allows to disconnect any server or user from the network. However, Stadium guarantees that any denial of service attack will not risk its users' privacy.

2.1 Cryptographic assumptions and PKI

Stadium relies on standard cryptographic assumptions. We assume secure public and symmetric key encryption, key-exchange mechanisms, signature schemes, and hash functions. We further assume a public key infrastructure, i.e., Stadium servers' public keys are known its users, and two communicating users know each other's public keys

(exchanged out of band, e.g., via a PGP server).

Chapter 3

Goals and Challenges

In this section we present Stadium's key design goals: providing a private messaging service in face of powerful attackers while allowing the system to scale to handle many users. To motivate our design choices we present the challenges in realizing these two goals together.

3.1 Privacy

Stadium has the same privacy goals as Vuvuzela [27]. That is it aims to prevent attackers from distinguishing between communication patterns of its users, even if they exchange many messages.

Informally, our definition captures the following: for any user say Alice, the adversary should not be able to distinguish between any of Alice's possible communication (or non-communication) patterns, even after observing the communication through the system for a long time. We use the following definition from the differential privacy literature to formally reason about Stadium's privacy guarantees [13]:

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

In our case, the inputs to the algorithm (x and y) are the users' communication

transcripts. We consider two inputs *adjacent* if they differ only by one user's actions. For instance consider the user Alice. One of the inputs represents Alice's real actions, e.g., Alice sends a message to Bob, while adjacent inputs represent hypothetical "cover stories" (e.g., Alice sends a message to Charlie, or to no one). The differential privacy definition requires that all of these cover stories appear almost as plausible as the real communication pattern, even when the adversary tries to infer on the nature of user's communication by observing the system's observable variables.

More precisely ϵ bounds the multiplicative difference in the probability for each of the user's actions given the observations that the attacker collects. This ϵ should be small enough to keep all other cover stories plausible. The other differential privacy parameter, δ, provides an additive increase to the differential privacy property, thus it should be very small in order to limit the attacker's success. We use δ to capture extreme errors, such the probability of communicating through an all-malicious path in the mixnet.

3.2 Scalability

We design Stadium to support tens of millions of simultaneous users, such numbers are comparable to Tor, the most popular anonymity service available. To support a growing number of users, the system's capacity increases with the number of servers. More specifically, the capacity of Stadium is almost linear in the number of servers.[1] The scalability goal therefore necessitates that only a subset of the servers will process each message, and represents a departure from Vuvuzela's design [27], which passes all messages through all servers.

3.3 Challenges

Realizing a scalable system without compromising its privacy guarantees is challenging. The key idea in previous private messaging work [27] was to limit the number

[1]The reason that it is not exactly linear is that some types of observable variables grow quadratically with the number of servers, we provide more details in the following sections.

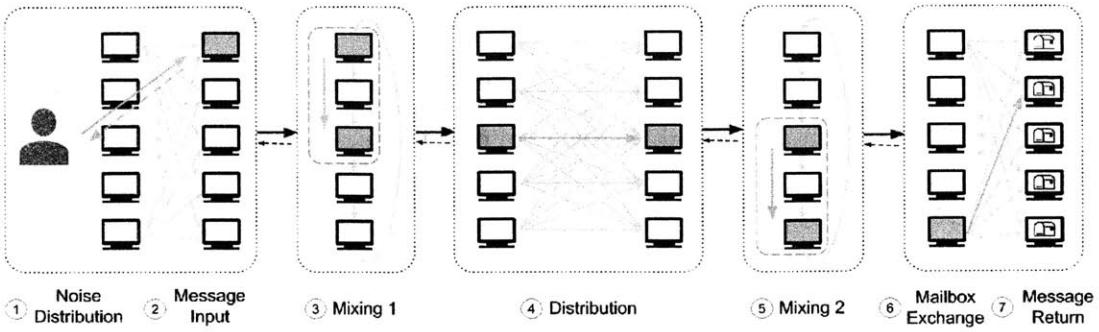


Figure 3-1: Stadium's design through 7 processing phases. Verification only takes place when a message traverses to the mailbox. The message is returned by reversing all the steps, denoted by the dashed arrows.

of observable variables so that the system can add noise to hide them. However, in Stadium, the number of observable variables grows with the number of servers as attackers may separately monitor the traffic patterns of each server and our privacy requirement mandates that each of these variables remains obfuscated by cover traffic (so as to keep any cover story plausible). Therefore, no single server can generate sufficient cover traffic to obfuscate all observable variables. Stadium includes a collaborative noise messages generation phase, where the amount of cover traffic that each server generates is limited (typically over an order of magnitude less than [27]), yet it is sufficient to obscure all observable variables.

However, utilizing the aggregated noise to provide cover traffic is a non trivial challenge that Stadium faces. Since noise is generated at a single collaborative phase, compromised servers may attempt to discard noise messages distributed to them (before it is mixed with user messages). Furthermore, compromised servers may also attempt to discard, replace, or modify some of the messages that they relay even after noise has been introduced to the chain, so as to identify the nature of few particular messages.

We solve both these challenges by constructing our distribution and mixing primitives in a verifiable manner. Specifically, each server is audited by some of the other servers on the message's path. This ensures that as long as one of these servers is honest, then either communication is private since all relaying servers followed the

protocol, or the malicious behaviour is detected and communication is cut-off by the honest server (so as to not compromise users' privacy). Allowing efficient verifications by a set of servers is an additional challenge of our design.

Chapter 4

Overview

Stadium is a distributed private messaging system. Communication takes place in rounds, every fixed interval a new round begins and Stadium passes the messages it accumulated since the last round started to their destinations. A message sent via Stadium travels in one of many available paths through the mixnet, where each path traverses only a subset of the servers to allow for a scalable deployment. Figure 3-1 illustrates the communication through Stadium. The servers in the system collaboratively form in random mixing chains, where each server begins one chain and a message's path is composed of two such chains. Thus the number of different paths in the network is quadratic in the number of servers. The use of two mixing chains to allow multiple paths through the network is similar to the parallel mixnet design in [17].

To allow anonymous communication independently of other users' traffic patters, Stadium servers collaboratively generate noise messages, used as cover traffic, and inject them to the system (phases 1 and 2 in Figure 3-1). The first mixing chain on a message's path shuffles the message with the fraction of the noise messages that enter the system through that chain (phase 3 in Figure 3-1). We later describe precisely how each server mixes the messages in a verifiable manner, but for now it may be thought of as reordering the messages through a random permutation and then re-randomizing each message. Attackers observing the communication links can no longer tell whether a message exiting this chain was sent by a user or is actually

noise, even if they control all but one of the mixing servers. To efficiently utilize the noise messages in the system, rather than only those that enter through that particular chain, messages are then distributed to a random second chain where they are mixed again, this time with messages originating in other servers (phases 4 and 5 in Figure 3-1). Finally, messages reach destination "mailboxes", virtual locations hosted on Stadium's servers that are associated with the anonymous conversations taking place, messages reaching the same mailbox are exchanged and routed to their intended recipient through the network (phases 6 and 7 in Figure 3-1).

In the following sections we describe the details behind each of these phases. Section 5 describes the group-verifiable mixing and verifiable distribution of messages, which form the two key mechanisms of Stadium's message processing pipeline. Section 6 describes how servers collaboratively generate and inject noise messages to the system, and then in Section 7 we analyze the privacy properties that Stadium can provide with a particular amount of noise. The reminder of this section deals with the other mechanisms behind the operation of Stadium, namely, forming of mixing chains and exchanging messages between users.

4.1 Mixing chains

When Stadium is deployed, its servers first jointly decide on a random seed which sets a cyclic order of the mixing servers. Once the random seed is established, Stadium creates constant length mixing chains of length l, i.e., every sequence of l servers in the cyclic order makes up a mixing chain (see illustration in Figure 3-1). A message's path begins at one server, say i, and then distributed from server $i+l$ (the last server in that chain) to another server, say j, which begins the second mixing chain. The message is passed through the mailbox by the last server in the second mixing chain (server $j+l$), as shown in Figure 3-1.

It is therefore important to guarantee that adversaries cannot set the order of the servers, or they could form all-compromised chains. To establish a random seed in a secure manner, despite malicious servers, we use a simple commitment-based

protocol as outlined in [5]. Each server first sends to all other servers a commitment to a random fixed-length bit-string, and then collects the commitments from the other servers. All servers then reveal their strings and verify that the other servers' strings match their earlier commitments. Each server then computes the XOR of all strings, this ensures that even if all but one server are malicious, then the servers' order is still random. When a client connects to the system, it first fetches the random seeds from all of its servers, and only if they are all identical (meaning that all servers agreed on a particular order) does it continue to send messages through the system. [1] The protocol runs over secure channels (e.g., using TLS), established using the servers' public keys which are distributed via a public key infrastructure (see Stadium's assumptions in Section 2). That is, an attacker cannot modify a honest server's messages.

Each chain is associated with a public/private key pair for a threshold decryption scheme that its members collaboratively establish. To establish this key for an Elgamal cryptosystem, we utilize the protocol by Gennaro et al. [15] as outlined in [10]. We set the threshold number of servers that allows to decrypt a message to be equal to the number of servers. That is, all servers need to contribute their decryption shares in order to decrypt a message. We chose the Elgamal cryptosystem to allow integration with the current verifiable shuffling protocols (described in Section 5).

4.2 Conversation setup

Similarly to Vuvuzela, a message's destination is a *mailbox*, a virtual location hosted on one of Stadium's servers (e.g., server i hosts all mailboxes with ids divisible by i). Two communicating users send and receive their messages through the same mailbox (but possibly through different paths) as we describe below. In order to avoid revealing whether a user is currently involved in a conversation, users send a message to a random mailbox in case they are not communicating with anyone at a particular round. The mailbox ID space is large, e.g., 256 bits long, to guarantee

[1] We do not ensure the system's availability in face of malicious servers. The reasoning behind this design decision is that our attacker can always break communication links any user from the network (see discussion in Section 3).

that two conversations will not 'collide' and use the same mailbox, thus, mailboxes typically receive 0, 1, or 2 messages.

The mailboxes used for conversation via Stadium are decided at random by the communicating peers. In order to coordinate a mailbox ID without revealing the identities of the peers, we employ Vuvuzela's dialing protocol [27] which uses several "invitation mailboxes" that are publicly known. Users are mapped to invitation mailboxes (e.g., by their public keys). To dial to a user, one should send their invitation mailbox a request specifying a random seed that allows both peers to derive a conversation mailbox ID as well as a symmetric key to encrypt and authenticate the content of their conversation. The invitation message is encrypted under the recipient's public key (we assume that communicating users know each-other's public key, e.g., obtained though a PGP server, see Section 2). Users periodically download the entire content of their invitation mailbox and try to decrypt each message, if they succeed then they can now communicate with the sender using the coordinated mailbox. In contrast to its communication protocol, Vuvuzela's dialing protocol can scale to handle many users by adding more invitation mailboxes. We refer to [27] for details.

4.3 Message encapsulation

When clients send a message through the mixnet, they decide on the two servers (i, j) that start the mixing chains on their message's path and the mailbox ID. To efficiently handle the message we divide it into two components. The actual content, which makes up most of the message's size, is padded to a constant length so as not to leak information about the data (e.g., 1KB). The metadata which specifies the mailbox ID and the identifier j for the server that begins the second mixing chain. The two components are encapsulated as follows.

To encapsulate the metadata, the client encrypts the mailbox ID using the public key of the second chain, and then appends the identity of the server that begins the second mixing chain (j), this is then encrypted with the public key of the first chain

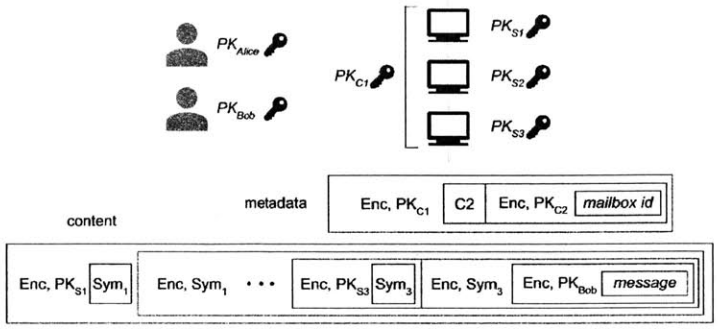


Figure 4-1: Summarizes the the different encryption schemes for the two components of a message, metadata and content.

(that starts at server i). The mixing chains work together to decrypt the metadata component with a threshold scheme once verification succeeds.

The second component of the message, message content, is onion encrypted such that each server on the path (rather than each chain) peels one of its layers. Specifically, we use a hybrid encryption scheme where the client chooses a symmetric key for each server on the path. The client then encrypts the content using the key selected for the last server, and appends the symmetric key encrypted under that server's public key. The output is used as the plain text for the next layer of hybrid onion encryption, which is encrypted for the previous server in the chain.

4.4 Message exchange and return

When a mailbox server receives two messages, it swaps their content and sends each message back through the reversed path in the network. If a mailbox server receives any other number of messages, they are routed back to their senders. In order to send the messages through the reverse path, each server re-randomizes the messages and then applies the inverse permutation it used for mixing and passes the messages in the new order backwards in the path (servers store the permutation they use in each round). To minimize latency, the return path does not employ any verifiable computations. This does not compromise privacy since after the message travels the

'forward path' in Stadium it is unlinked with its sender, for this reason we also need not trust the server hosting the mailbox. Adversaries may replace or tamper with a message as it travels the reversed path, which may deny peers from communicating, but does not on its own reveal any information about their communication patterns.

Chapter 5

Verifiable Processing

In this section we present the verifiable mixing, distribution, and message input, the mechanisms that compose Stadium's message processing pipeline. As highlighted in Section 3.3, verifying that Stadium's servers correctly follow these protocols is key to coping with the significant number of observable variables that the parallel mixnet exposes. We conclude this section by evaluating the communication cost of Stadium's message processing.

5.1 Group Verification for Mixing

In the mixing phases (phases 3 and 5 in Figure 3-1) each server shuffles the message batch that it has and passes the shuffled messages to the next server in the circular order. The shuffling procedure also re-randomizes the outputs such that an attacker observing both inputs and outputs cannot reconstruct and invert the permutation. To re-randomize the metadata, we take advantage of the malleability property of the ElGamal cryptosystem. The content part of the output is randomized when the mixing server peels its layer of the hybrid encryption. The mixing server need only to verifiably shuffle the message's metadata, since it determines the message's path and destination. The content is shuffled by same permutation, but need not be verified since changing the content does not leak information about the communication patterns (the modified message must be of the same length and will travel the same

path).[1]

To shuffle the metadata we use a verifiable shuffle technique [3] which allows the shuffler to later interact with an auditing server and prove that the output is a re-randomized permutation of the input without revealing any knowledge about the permutation itself. In Stadium every server is audited by all other servers on its chain, if one of the servers does not accept the proof, then it does not contribute its decryption shares which blocks all messages in the mixing batch from continuing their path through the mixnet. This guarantees that as long as there is one honest server in the chain, then either all messages in the batch are correctly shuffled or communication is disrupt (so as to avoid compromising users' privacy).

The servers begin auditing a shuffle as soon as the mixing-server had completed its processing. To inform that shuffling is complete, the shuffler broadcasts its output to all other servers in the chain. In Stadium all servers know which messages are input to their mixing chain, this is achieved through the message input phase (in case of the first mixing chain), or the verifiable distribution phase which we next describe (in case of the second mixing chain); the auditing servers confirm that the output of the shuffler is some permutation of the input to the chain.

5.1.1 Group-verification extension

The zero-knowledge proof protocol [3] is computationally expensive for the prover. To significantly reduce the cost of performing this protocol, we extend it to support group-verifications, that is allow the prover to satisfy multiple auditing servers with a single proof. To our knowledge previous implementations and schemes [3, 14, 21, 25] do not support for group verifications. Our extension is simple to realize. The verifier's protocol [3] consists of multiple challenge and response steps, where the challenge is selected at random by the verifier. We therefore have all the auditors select a random challenge together at each step (only after the previous step completes), this is performed using the same protocol for selecting the seed that sets the cyclic

[1]The dialing protocol [27] allows users to decide on an ephemeral symmetric key, which they can use to encrypt and authenticate messages they send, to discard corrupt messages.

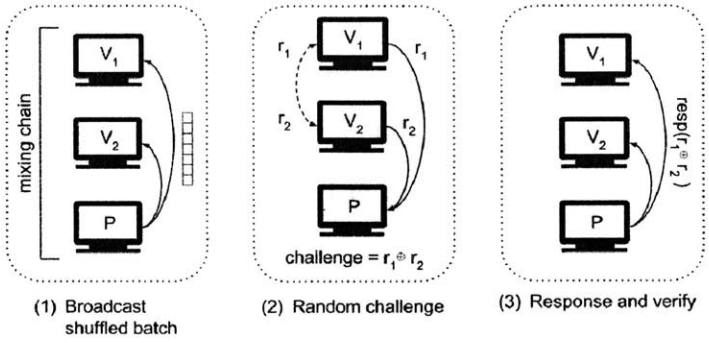


Figure 5-1: Summary of group verification. Verifiers work together to create a single random challenge for each verifying step of the interactive proof protocol.

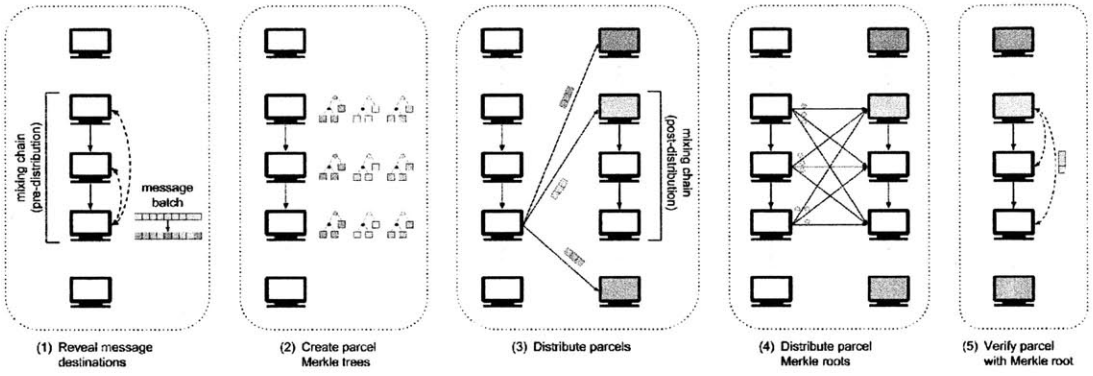


Figure 5-2: Summary of the verifiable distribution protocol. This protocol is used to allow servers to verify that messages are not tampered with during distribution.

order of the chains (see Section 4). Each of the servers runs its own instance of the protocol with the prover, but sends a collaboratively generated random challenge at each step, this allows the prover to send the same response to all auditors such that each step in computing the proof is only performed once. The group verification extension is illustrated in Figure 5-1.

5.2 Verifiable Distribution

The last server of the first mixing phase decrypts the first layer of the onion-encrypted metadata and reveals for each message the server who starts the second mixing phase

(see Figure 3-1). In the distribution phase messages are delivered to the chains of the second mixing phase. The servers on the second chain receive the parcels of the messages from the servers that complete the first mixing phases (see phase 4 in Figure 3-1). The goal of the verification of this step is to ensure that all messages were correctly distributed, namely, that the last server on the first mixing chain does not modify the message set before it distributes the messages to the next chains. The verifiable distribution protocol has two steps: first all the servers of the first chain agree on the parcel (message set) they distribute to each of the second mixing phase chains. Second, all the servers in the second chain verify that they received valid parcels (i.e., that include exactly the messages that the servers in the first chain agreed on).

To achieve the first step, i.e., agree on valid parcels to distribute, the last server in the first chain broadcasts the decrypted metadata of the messages to the other servers in mixing phase 1 (together with the ephemeral key used to encrypt each message). The servers verify that decryption is correct, i.e., that encrypting the message with the chain's public key yields the original set of encrypted metadata. If the verification succeeds, the server computes the hash for the parcel of messages that go to each of the chains in the system. If the server at the end of mixing phase 1 did not modify the set of messages, then all servers should compute the same hash.

For the second step, i.e., verify that valid parcels were received, each server in the first mixing chain broadcasts to the group of servers in the second mixing chain the hash of the parcel sent to the second mixing chain. The servers in the second mixing chain verify that they received the same hash from all servers. Finally, the last server in the first mixing chain distributes the parcels it has for each of the following chains. Each of the servers in the second mixing chain now computes the hash for that parcel and verifies that it matches the value it received from the servers on the first chain. The protocol is illustrated in Figure 5-2.

5.3 Message Input

In order to keep its observable variables obscured, Stadium must verify that adversaries do not tamper with messages as they enter the system to gain information. For instance, an attacker that injects two duplicated copies of a user message can then easily learn the destination mailbox, which will have 3 accesses or more (all copies reach the same mailbox, despite the mixing). The initial message batch of a server is composed of two sets of messages, noise messages received from noise distribution and user messages received from message input (see phase 2 in Figure 3-1). To verify that a message batch does not contain any tampered messages, both the noise message set and the user message set are broadcast and verified by all servers in the mixing chain.

Each server generates noise and distributes it to other servers during the noise distribution phase, which takes place before users input their messages to the system. To allow all servers of a mixing chain to verify that noise was correctly distributed to their chains, the server generating the noise divides it to distribution parcels for each chain (Noise generation is covered in Section 6) and broadcasts each parcel to its chain.

User messages are input in a two-step commit-then-reveal protocol to allow for verification of the user message set. Users first send a commitment of their message to all servers in their first mixing chain. After the commitment phase, users reveal the message to the start server of the mixing chain. When servers verify the user message set, they confirm that they received a user commitment for every message in the set. By the time the user messages are revealed, malicious servers can no longer input messages since the commitment step is over. Messages in the user message set that do not have commitments are caught and removed by verifying servers. Copied messages from previous rounds will also not be valid, since mixing chains renegotiate keys for every round.

User messages without commitments are removed. Commitments without user messages are ignored. Allowing the round to continue ensures that malicious users

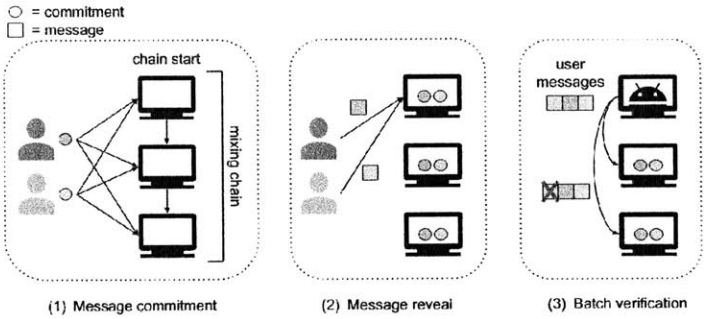


Figure 5-3: Summary of two-step user message input protocol. In the third server verification step, a malicious server attempts to add a tampered message.

do not have the power to mount a denial of service attack on the system. In return, malicious servers do not get punished for dropping user messages. We argue this fine, since we assume the adversary has the power to cut users off the network anyways.

5.4 Communication Cost

The communication and computation costs of a server are proportional to the chain length l, with the exception of broadcasting parcel hashes h to all m servers in the distribution phase. Call the number of messages in a message batch n. Call the size of the message content s_c, the size of the message metadata s_m, and collectively, s. The metadata portion of the message is passed around often for verification, while the message content portion is passed minimally. See Table 5.1 for a summary of the computation and communication costs of each Stadium processing phase.

Stadium phase	Computation costs	Communication costs
Message Input	n noise message generation ln commitment checking	$ns + lns_m + nh$
Mixing	ln re-encryptions l proofs size n l^2 verifications size n ln decryption shares n decryptions	$l^2 s_m + ls$
Distribution	ln encryptions ln parcel hashes	$lmh + lns_m + ns$
Message Return	$2ln$ re-encryptions	$2ls_c$

Table 5.1: Computation and communication costs of Stadium processing broken down by phase.

Chapter 6

Hiding Observable Variables

In this section we identify the observable variables that Stadium exposes, and describe how servers collaboratively inject noise to the system to obscure these variables.

6.1 Observable Variables

Attackers may monitor the communication between all system servers and users. However, we design Stadium such that most of the parameters studied through these observations do not leak information as to the communication patterns of its users. First, users send a message at every round regardless of whether they are active in a conversation, therefore, observing a user sending a message does not leak any information about their conversations[1]. Second, somewhat counter-intuitively, the number of messages exchanged between any pair of servers while they mix and relay messages does not provide any information on the users' communication patterns. The reason is that a message's path is independently selected (at random) by every user.

The last type of links that are available for the attacker to monitor are those between the last mixing server and the mailbox. The mailbox access pattern, in contrast to previous communication links, is affected by the communication patterns:

[1]Stadium does not attempt to obscure the fact that a user uses the system, it only hides their communication patterns.

if two messages reach the same mailbox (e.g., hosted on an attacker-controlled server), then this indicates that the mailbox is employed in a conversation, whereas if only one message reaches the mailbox in a conversation round, then that means it is used by an idle user (who is not currently involved in a conversation).

We therefore define two classes of observable variables that are affected by changes in communication pattern: (1) single access variables N_1^i, the number of mailboxes that receive exactly one message and it came from server i, and (2) double access variables $N_2^{i,j}$, the number of mailboxes that receive two messages, where one message came from server i and the other from server j. There are therefore m single access variables and $\binom{m}{2} + m = \frac{m^2+m}{2}$ double access variables.

6.2 Noise Generation

Every server generates noise messages independently of the others and the aggregated noise is then evenly distributed across all servers. We defined the two types of variables that Stadium exposes to attackers, the single and double access mailbox patterns. There are therefore two types of noise messages: single messages, that are sent from server i to a random mailbox, imitating idle users to obscure the variable N_1^i, and pairs of messages sent to the same mailbox from servers i, j, imitating two communicating users, to obscure the variable $N_2^{i,j}$.

To select the amount of noise for obscuring both types of variables our system employs the Laplace distribution. Namely, to decide on the number of noise messages that a server would generate, it first draws a number from each of the noise distributions, which decide on the amount of noise messages that the server would generate to obscure all single and double access noise variables (respectively). Since servers cannot produce 'negative noise', i.e., eliminate real users' access patterns, a server sends no noise in case it draws a negative number from the distribution. The server then uniformly distributes the number of messages it decided on across all variables of that type (i.e., m single-access variables or $\frac{m^2+m}{2}$ double access variables). The verifiable message input phase ensures that all chains in the mixnet, including those

with malicious servers, use sufficient cover traffic to provide the system's privacy guarantees (see Section 5.3).

Since the number of variables of each type differs, the parameters of the distribution differ between the two types of noise. Specifically, each honest server decides the amount of amount of single noise $x_1 \sim \max(0, \lceil \text{Laplace}(\mu_1, b_1) \rceil)$ to obscure all m single access variables. That noise is uniformly distributed across the m single access variables. Specifically, a server generates noise for a single access variable by the distribution $\max(0, \lceil \text{Laplace}(\frac{\mu_1}{m}, \frac{b_1}{m}) \rceil)$. Denote f as the fraction of honest servers in the system (that generate noise by the above distribution). The amount of noise covering one single-access variable is therefore the sum of fm such distributions, which distributes $\max(0, \mathcal{N}(f\mu_1, \frac{2fb_1^2}{m}))$ (by the central limit theorem). Similarly, we find that the noise covering a double-access variable distributes $\max(0, \mathcal{N}(2f\mu_2, \frac{8fb_2^2}{m^3}))$.

Chapter 7

Security Analysis

In this section we analyze how the privacy guarantees that Stadium can provide depend on the amount of noise each server generates. Our analysis is composed of two main parts. The first shows that by monitoring the first mixing chain an attacker can only gain very little information on the second chain that a message is distributed to. In the second part of the analysis we utilize this property to reason about the distribution of noise covering each observable variable and compose them together over multiple communication rounds to obtain the differential-privacy properties of Stadium, i.e., the ϵ and δ of our privacy definition (see Section 3).

7.1 Honest Mixing Chains

Stadium does not require all servers to be honest. Instead, to provide its security guarantees we assume that only a fraction f of the servers are honest, i.e., correctly follow the protocol. Malicious servers can deviate from the protocol in any way. In the parallel mixnet, user messages are routed along random paths, where some of which may contain malicious servers. This section examines the probabilistic assumption that every mixing chain contains at least one honest server. This is necessary to ensure that messages are mixed by at least one permutation unknown to the adversary. Probabilistic assumptions are accounted for in the differential privacy analysis of Section 7.3.

7.1.1 Mixing chain length

Let us denote the number of servers in the system by m and the length of mixing chains by l. Servers are placed in a circular order determined using a pseudo-random function (see details in Section 4). The probability that a particular mixing chain consists of entirely malicious servers is therefore $p = \frac{\binom{m(1-f)}{l}}{\binom{m}{l}}$. Therefore the probability that some chain is entirely malicious out of m total chains (each server begins a chain) is $1-(1-p)^m$. This probability, that some mixing chain is completely compromised, is incorporated into the δ of the the differential privacy analysis. In particular, consider an instantiation of the system with $m = 100$ servers, where $f = 75\%$ of which are honest. In this case for a chain length $l = 10$, the probability that some chain is compromised is $1.8 \cdot 10^{-5}$.

Assumption 1. *Every mixing chain contains at least one honest server.*

7.2 Belief Probabilities

The probability that a particular user, Alice, picks a specific output server, say server j, is $\frac{1}{m}$. However, the adversary can observe message traffic during the parallel mixnet execution and gain information to skew this probability. More specifically, adversaries may observe the entrance server of Alice's message, and therefore also learn which server distributes Alice's message at the end of the mixing phase 1, say server i. Denote server i's message batch by $batch_i$, and the messages which are distributed from $batch_i$ to server j by $parcel_{i,j}$. Since messages are distributed to random servers, these parcels may vary in size. Though parcel sizes are equal in the expected case, a random uneven distribution can give the adversary non-uniform belief probabilities to which server Alice's message was sent to. Figure 7-1 illustrates this scenario. We define the belief probability, $p_{i,j} = \frac{parcel_{i,j}\text{size}}{batch_i\text{size}}$. Intuitively, the more noise the system generates, the more likely the belief probabilities are close to uniform, i.e., $\frac{1}{m}$. [1]

[1] Although malicious servers can add dishonest noise to artificially skew the belief probabilities, adversary-controlled noise does not give the adversary any extra information on the distribution of honest messages. Thus, belief probabilities are defined relative to the amount of honest messages.

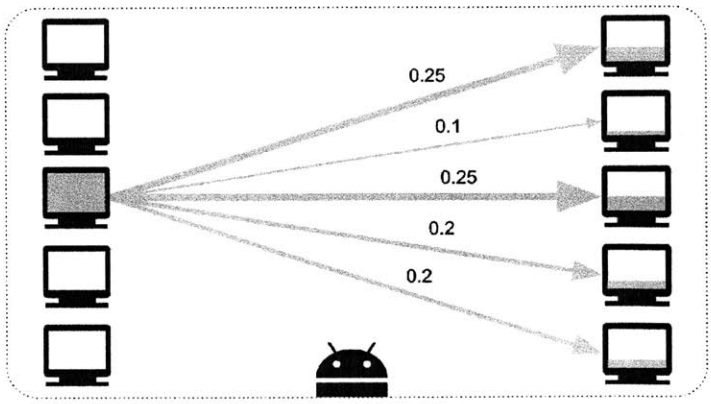


Figure 7-1: An adversary may gain information about where Alice's message is distributed to based on a non-uniform message batch distribution.

Assumption 2. *Belief probabilities are bounded,* $\exists\ \alpha > 1,\ \ \forall\ i, j \in [1, m]$ *such that the belief probability,* $p_{i,j} < \frac{\alpha}{m}$.

We want to upper bound all belief probabilities $p_{i,j}$ close to uniform, $\frac{\alpha}{m}$, for some $\alpha > 1$. The probability that some belief probability exceeds the bound is incorporated into δ of differential privacy, so the bound must hold with high probability. First, we calculate the probability that all the belief probabilities in a given batch distribution are upper bounded, then compose over m batches. We lower bound the size of the message batch, b_{batch}, with high probability using the distribution of noise messages. The probability p that a given belief probability of this batch falls under the bound $\frac{\alpha}{m}$ is upper-bounded from the binomial CDF of $\text{Binom}(b_{batch}, \frac{1}{m})$ with input $\frac{b_{batch}\alpha}{m}$. Composing these bounds together over m belief probabilities per batch and m batches in the system gives us a total probability that all belief probabilities are $< \frac{\alpha}{m}$.

7.3 Differential Privacy

Our differential privacy guarantee (defined in Section 3) keeps any adjacent communication transcripts, i.e., which differ by one user's actions, almost equally plausible. Our design realizes these adjacent transcripts if they differ by a mailbox ID used by one user. Namely, if Alice uses the same mailbox ID as Bob, then that means

that Alice and Bob communicate. On the other hand, if Alice chooses a mailbox ID that is not used by anyone else then that means she is currently not involved in a conversation (i.e., idle).

Our system exposes many variables to the attacker (quadratic in number of service). Composing many variables together may significantly degrade the differential privacy guarantees. To cope with this challenge we leverage the following key property of Stadium.

Theorem 1. *Between any two adjacent inputs there are only at most two single access variables and two double access variables that change.*

Proof. Given in Appendix A. □

7.3.1 Belief-adjusted differential privacy

Intuitively, the belief probabilities capture the attacker's uncertainty of a message's route, the following theorem allows to adjust the (ϵ, δ)-differential privacy gurantees according to the adversary's belief probability bound.

Theorem 2. *An observable variable with ϵ, δ differential privacy, which the attacker believes to describe a user action with probability p, provides $\epsilon' = \ln(1+p(e^{\epsilon}-1)), \delta' = p \cdot \delta$ differential privacy.*

Proof. Given in Appendix B. □

A single access variable is affected by an adversary's belief on where Alice's message is output, bounded by $\frac{\alpha}{m}$. Define the final differential privacy for a single access variable, (ϵ_1, δ_1), as the belief-adjusted results of applying Theorem 2 with $p = \frac{\alpha}{m}$ to the results of Theorem 3 on single access noise $\max(0, \mathcal{N}(f\mu_1, \frac{f\sigma_1^2}{m}))$. Similarly for double access variables, define (ϵ_2, δ_2) using $p = 2(\frac{\alpha}{m})^2$ and double access noise $\max(0, \mathcal{N}(2f\mu_2, \frac{4f\sigma_2^2}{m^3}))$

7.3.2 Differential privacy for a single round

As described above, the noise covering each variable in the system distributes according to the Gaussian distribution (since it is the sum of the noise generated by the servers). Dwork and Roth [13] analyze the differential privacy properties when noise is generated following the Gaussian distribution.

Theorem 3. *A variable with noise drawn from* $n \sim \mathcal{N}(0, \sigma^2)$ *is* (ϵ, δ)*-differentially private with* $\epsilon = \frac{\Delta c}{\sigma}$ *and* $\delta = 1.25 \cdot e^{\frac{-c^2}{2}} + \xi$*, for small* $\xi > 0$*.*

Proof. Direct from Theorem A.1 in [13] □

The parameter Δ specifies the maximum absolute change in the variable. We examine how much variables change between adjacent instances later in this section. The parameter c allows to trade larger ϵ for smaller δ.

The Sequential Composition theorem in [20] allows us to calculate the differential privacy guarantees when the attacker can observe multiple variables by summing their ϵ and δ parameters. However, Stadium reveals m^2+m single access and double access variables. Naively composing them together by applying the theorem would not yield the strong ϵ, δ-differential privacy properties we desire. Furthermore, the quadratic growth in number of variables indicates that this approach would not scale as the system's deployment extends to support more users. We avoid these problems with a key insight stated in Theorem 1, the number of affected variables is bounded by a constant, and is independent of the number of servers m. Although it is unknown which specific variables are affected at each round, the single round ϵ, δ-differential privacy properties is bounded by their individual sums.

7.3.3 Accounting for errors

The probability of rare errors that break our assumptions are incorporated into the δ as an additive factor. These errors are (1) the probability that there exists a chain consisting fully of malicious servers, (2) the probability that a server samples 0 noise

during noise generation, and (3) the probability that a belief probability exceeds the upper bound.

7.3.4 Conversation over multiple rounds

Stadium allows users to interactively communicate over multiple communication rounds. 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 [13] and a similar challenge was handled by the Vuvuzela system [27]. Fortunately, differential privacy provides a bound on ϵ and δ after k rounds.

Theorem 4. *Consider an algorithm M providing ϵ, δ differential privacy, then M provides ϵ', δ' differential privacy after k rounds with parameters:*

$\epsilon' = \epsilon\sqrt{2kln(1/d)} + k\epsilon(e^{\epsilon} - 1)$ *and* $\delta' = k\delta + d$, *for any* $d > 0$.

Proof. Direct from Theorem 3.20 in [13]. □

The parameter d allows to trade higher ϵ for lower δ.

Chapter 8

Implementation and Performance

We implemented a prototype of our system to measure its performance and feasibility. The control flows and networking mechanics of our system are implemented in Go, while the underlying cryptogrpahic protocols are implemented in C++ using the NTL library [24]. The C++ library includes a parallelized version of the verifiable shuffle implementation by Bayer and Groth [3] using prime groups. The total implementation of the system logic contains less than 3000 lines of Go and C++ code (atop of existing libraries our implementation uses). We deployed an unoptimized, distributed prototype using a 9 server Stadium configuration in order to get performance results for a complete chain of length 9. In order to extrapolate results for larger loads, we then measured performance for an optimized single machine workload. These results provide a good measure of Stadium's scalability to hundreds of servers since a server's workload is determined by chain length and message batch size. We simulate the number of servers in the system by considering different amounts of noise and user messages in a batch. To compare with Vuvuzela, we calculate the the amount of noise a certain server configuration needs in order to achieve (ϵ, δ)-differential privacy of $\epsilon < 2$ and $\delta < 10^{-4}$ after 10^5 rounds. The results are shown in Table 8.1. We do not implement client connections and the two-phase message input protocol. User messages are simulated by having each entry server generate extra messages in addition to their noise.

The 9 server Stadium prototype was deployed on Amazon EC2 virtual machines

# of servers	noise/server	e^{ϵ}	δ	α	μ_1	μ_2
100	5×10^4	1.94	9.8×10^{-5}	1.24	12500	187.5
200	6×10^4	1.67	9.8×10^{-5}	1.17	13200	117
300	7×10^4	1.67	9.6×10^{-5}	1.13	13300	94.5
400	8×10^4	1.77	9.5×10^{-5}	1.10	12800	84
500	10×10^4	1.57	9.8×10^{-5}	1.10	13000	87
600	11×10^4	1.80	9.8×10^{-5}	1.08	14300	79.75
700	12×10^4	1.96	9.7×10^{-5}	1.07	12000	77
800	14×10^4	1.87	9.5×10^{-5}	1.07	14000	78.75
900	16×10^4	1.82	9.9×10^{-5}	1.06	16000	80
1000	17×10^4	1.94	9.8×10^{-5}	1.06	11900	79

Table 8.1: Sample configurations for different number of servers. Gives the (ϵ, δ)-differential privacy, α belief probability, and μ_1, μ_2 mean of noise per single access and double access variable.

(VMs). Each Stadium server was run on a `c4.large` machine with 2 Intel Xeon E5-2666v3 cores and 3.75 GB of RAM. For a message batch size of 2000, the prototype completed a round in 120 seconds. The single machine optimized experiments were run on an `c4.8xlarge` Amazon EC2 VM with 36 Intel Xeon E5-2666v3 cores.

Figure 8-1 shows the number of user messages per second that different Stadium server configurations can support. The throughput results given correspond to a message batch size of 400,000 and an end-to-end latency of 6 minutes. The increase in message throughput is near linear in the number of servers and starts to taper as noise messages become a larger fraction of the message batch size (see Table 8.1). Stadium's throughput is comparable to Vuvuzela for smaller server configurations and we observe it grow to over 400,000 messages/sec for larger server configurations (4× of Vuvuzela) and continue to grow with even larger server configurations.

Figure 8-2 shows how the message latency of Stadium changes with increased user load for different server configurations. We see that for latencies less then two minutes, Vuvuzela is able to support a larger number of users. This is because Stadium's message batch is mostly noise at this level. At the breaking point of approximately 2 minutes and 5 million users supported, Stadium scales past Vuvuzela.

Our performance results indicate that a significant difference between Stadium

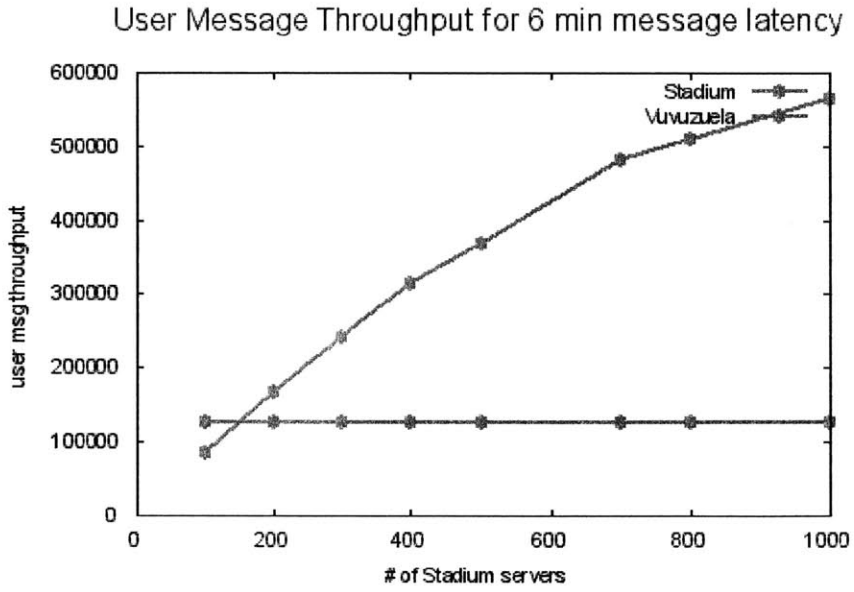


Figure 8-1: Stadium throughput as a function of number of servers.

and Vuvuzela's performance comes from the bandwidth use of each server. Vuvuzela achieves its throughput with use of 1.3 Gbps links. In contrast, each Stadium server requires only 140 mbps (10× less than Vuvuzela). Though the combined bandwidth cost of Stadium is an order of magnitude higher than Vuvuzela, we argue the order of magnitude reduction in bandwidth cost per server makes Stadium feasible for large-scale deployment.

Our implementation takes advantage of multiple available cores and is embarrassingly parallel due to the CPU-intensive verifiable shuffling. We were able to reduce the processing time linearly with the number of cores, and believe that Stadium can continue to scale past 36 cores with better hardware.

Finally, modifying the group-verifiable shuffling primitive, which is the most CPU-intensive part of Stadium's processing pipeline, to use efficient elliptic rather than prime groups can significantly improve performance; Bayer and Groth suggest a 3× factor of improvement in run-time.

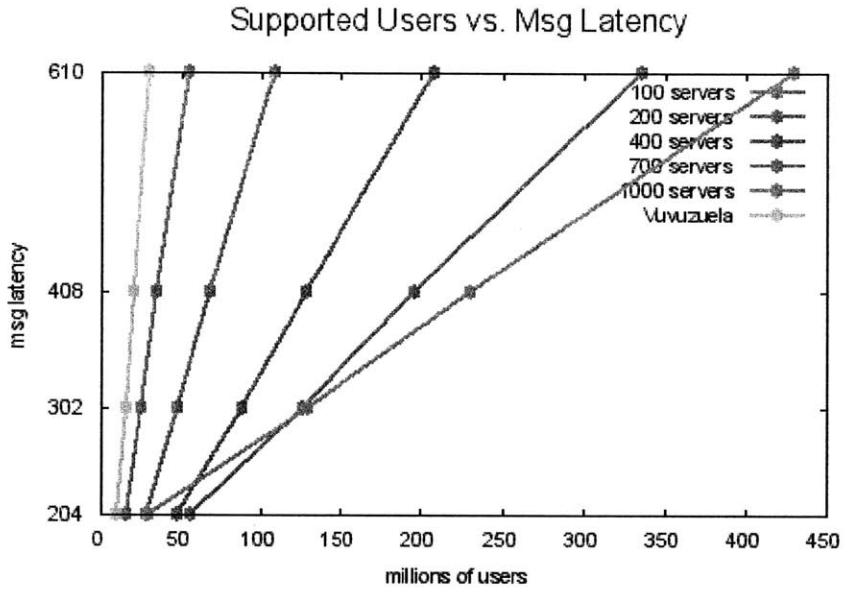


Figure 8-2: Stadium message latency as a function of number of users connected.

Chapter 9

Related Work

9.1 Anonymous communication systems

This paper compares with Vuvuzela [27], and designs Stadium to achieve the same privacy goals based on differential privacy, but allow to scale to handle more users and reduce the costs of operating each server. There are several other approaches to facilitate anonymous messaging. One extreme approach is to provide strong, provable privacy, for any subset of communicating users in the system. Systems in the category, such as Riposte [8] and Dissent [9], either rely on broadcasting all messages which introduces significant traffic volumes and communication costs, or utilize computationally intensive private information retrieval (PIR) protocols. As a result, these systems have only scaled to support thousands of users or a few hundred messages per second.

On the other extreme are systems that scale to handle tens of millions of users such as Tor [11] and mixnets [7], but do not protect against powerful adversaries. In particular, the privacy that such systems can provide depends on the number of users in the system and their traffic patterns. That is, if Alice uses Tor to secretly post a large document while most other users only download content, then it is easy to spot her traffic in the crowd. It is therefore hard to reason about the privacy provided by this systems. Moreover, adding cover traffic to reduce the dependency of one user's privacy on others' traffic has been shown to yield very limited benefits against passive

analysis over time [19]. Finally, these systems are highly susceptible to active attacks, for example, attackers may inject delays or discard some messages and observe the effect of their interference to gain more information [16].

9.2 Verifiable mixnets

Stadium uses verifiable shuffling to ensure that no server modified the messages after noise have been introduced in an initial collaborative phase. Verifiable shuffling has been originally proposed to allow globally verifiable e-voting [6, 21, 14, 3, 10, 1], another privacy related scenario. In this paper we use the verifiable shuffling protocol in the same privacy-related context, but for a different goal, that is to allow a scalable construction of a private messaging system.

9.3 Differentially-private systems

Several works present systems providing privacy-preserving services using differential privacy. PINQ [20] allows to perform set operations for processing a database of private records. Airavat [22] allows Map-Reduce processing of sensitive data. These works provide a privacy budget, which reason about the information that leaked to the user through past queries and prevent operations once the budget had been depleted to ensure that the underlying records remain private. In contrast to these systems, which assume a single trusted service that introduces noise and follows the protocol, our system is distributed and assumes that some participating parties are compromised and may deviate from the protocol.

9.4 Utilizing legitimate traffic as noise

Stadium assumes a powerful adversary, that may disconnect all but two users of interest from the system. To provide its privacy guarantees without depending on the number of users, Stadium uses noise messages which are processed through the

system and therefore introduce communication and processing overheads. Depending on the deployment environment, such powerful attackers may appear unreasonably strong. Coupled-worlds privacy [2] and noiseless database privacy [4] suggest, as an alternative, the notion of using legitimate users' traffic as noise, to use the legitimate

Chapter 10

Conclusion

Stadium is the first distributed system for providing a differentially private anonymous messaging service. The system scales to support over an order of magnitude more the number of users supported by the current state of the art. Stadium distributes the load of users allows the system to incur only moderate costs for operating each of its servers, which may encourage organizations to contribute their resources. The main challenge that Stadium's design faces is the increase of observable variables as more servers are deployed. This is efficiently handled by all servers collaboratively generating sufficient noise to obscure all these variables, and ensuring that malicious servers cannot discard that noise through a combination of verifiable distribution and group-verifiable shuffling mechanisms. Together, these mechanisms allowed us to deploy a scalable private messaging system with provable privacy guarantees.

10.1 Future work

This paper leaves several directions for future work. First, Stadium's design may be extended to provide fault tolerance, i.e., have the system automatically recover when some of the nodes fail. Second, we believe that the implementation of Stadium may be improved by changing the cryptographic constructions underlying its shuffling mechanism to use efficient elliptic curve cryptography.

Appendix A

Proof of Theorem 1

Proof. Suppose Alice has a unique shared secret with each user of Stadium. This shared secret defines a mailbox id that Alice should send her message to if she would like to communicate with said user. We define two types of users: (1) active users attempt to communicate with Alice by sending their message to the shared mailbox id and (2) inactive users that send to some other mailbox id unknown to Alice.

Alice can take one of three actions: idle (send to a random mailbox id), communicate with an active user, or communicate with an inactive user. "Communicating" with a user means that Alice sends her message to the shared mailbox id.

In an adjacent instance, the mailbox id of Alice is changed. Different access patterns are observed for servers that contain a message going to the two affected mailbox ids. Consider the case where Alice switches from idle to communicating with an active user, Bob. In the instance where Alice is idle, she sends her mailbox to a random mailbox id, which is observed as part of the single access variable of her output server. Bob sends his message to their shared mailbox id, but since Alice is not communicating with him, his message is also observed as part of the single access variable for his output server. In the instance when Alice is communicating with Bob, both Alice and Bob send their messages to the shared mailbox id, which is observed as part of the double access variable between their two servers. Switching between these two instances gives a change of 1 in two single access variables and a change of 1 in a double access variable.

Alice's cover story action	key = $(N_1^a, N_1^b, N_1^c, N_2^{a,b}, N_2^{a,c})$ Alice's real action		
	Idle	Communicating with b	Communicating with x
Idle	-	(+1,+1,0,-1,0)	(0,0,0,0,0)
Communicating with b	(-1,-1,0,+1,0)	-	(-1,-1,0,+1,0)
Communicating with c	(-1,0,-1,0,+1)	(0,+1,-1,-1,+1)	(-1,0,-1,0,+1)
Communicating with x	(0,0,0,0,0)	(+1,+1,0,-1,0)	-
Communicating with y	(0,0,0,0,0)	(+1,+1,0,-1,0)	(0,0,0,0,0)

Table A.1: Table showing change in observable variables in Alice's adjacent instances. Users b and c are active users. Users x and y are inactive.

Consider Table A summarizing the effects of each of these adjacent cover stories on the values of observable variables. □

Appendix B

Proof of Theorem 2

Proof. Consider the single access observable variable, N_1^i. Consider further distinguishing between two cases: (1) Alice is talking to Bob, (2) Alice is not talking to anyone. We denote the two cases by T and NT respectively. We will show:

$$\mathbb{P}[N_1^i = x \mid NT] \leq e^{\epsilon'} \cdot \mathbb{P}[N_1^i = x \mid T] + \delta'$$

If N_1^i is the number of single access messages, denote $\mathcal{N}_1^i$ as the number of these messages that are noise. Then, for instance T, $\mathcal{N}_1^i = x$. And for instance NT, $\mathcal{N}_1^i = x - 1$ or x depending on whether Alice's message goes through server i. Say the probability Alice's message goes through server i is p,

$$\mathbb{P}[N_1^i = x \mid NT] = p \cdot \mathbb{P}[\mathcal{N}_1^i = x - 1] + (1 - p) \cdot \mathbb{P}[\mathcal{N}_1^i = x]$$

$$\mathbb{P}[N_1^i = x \mid T] = \mathbb{P}[\mathcal{N}_1^i = x]$$

Since the noise follows a Gaussian distribution, we know from Theorem 3, for some (ϵ, δ),

$$\mathbb{P}[\mathcal{N}_1^i = x - 1] \leq e^{\epsilon} \cdot \mathbb{P}[\mathcal{N}_1^i = x] + \delta$$

This gives us,

$$\mathbb{P}[\mathcal{N}_1^i = x - 1] + (1 - p) \cdot \mathbb{P}[\mathcal{N}_1^i = x]$$
$$\leq p \cdot (e^\epsilon \cdot \mathbb{P}[\mathcal{N}_1^i = x] + \delta) + (1 - p) \cdot \mathbb{P}[\mathcal{N}_1^i = x]$$

And rearranged leads to,

$$\mathbb{P}[\mathcal{N}_1^i = x - 1] + (1 - p) \cdot \mathbb{P}[\mathcal{N}_1^i = x]$$
$$\leq (1 + p(e^\epsilon - 1)) \cdot \mathbb{P}[\mathcal{N}_1^i = x] + p\delta$$

$$\mathbb{P}[N_1^i = x \mid NT] \leq (1 + p(e^\epsilon - 1)) \cdot \mathbb{P}[N_1^i = x \mid T] + p\delta$$

The analysis for double access variables is synonymous and the case where both Alice and Bob go through server i is covered by sequential composition. □

Bibliography

[1] Masayuki Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In *Advances in Cryptology – Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 437–447, Helsinki, Finland, 31 May– 4 June 1998. Springer-Verlag.

[2] Raef Bassily, Adam Groce, Jonathan Katz, and Adam D. Smith. Coupled-worlds privacy: Exploiting adversarial uncertainty in statistical data privacy. In *FOCS*, pages 439–448. IEEE Computer Society, 2013.

[3] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology–EUROCRYPT 2012*, pages 263–280. Springer, 2012.

[4] Raghav Bhaskar, Abhishek Bhowmick 0001, Vipul Goyal, Srivatsan Laxman, and Abhradeep Thakurta. Noiseless database privacy. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2011.

[5] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO ' 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2001.

[6] Chin-Chen Chang and Wen-Bin Wu. A secure voting system on a public network. *Networks*, 29(2):81–87, 1997.

[7] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24:84–88, February 1981.

[8] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society, 2015.

[9] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 340–350. ACM, 2010.

[10] Adam M Davis, Dmitri Chmelev, and Michael R Clarkson. Civitas: Implementation of a threshold cryptosystem. 2008.

[11] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[12] Cynthia Dwork. Differential privacy. In *Automata, languages and programming*, pages 1–12. Springer, 2006.

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

[14] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *Advances in CryptologyâĂŤCRYPTO 2001*, pages 368–387. Springer, 2001.

[15] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT ' 99*, volume 1599 of *Lecture Notes in Computer Science*, pages 295–310. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1999.

[16] Yossi Gilad and Amir Herzberg. Spying in the dark: TCP and tor traffic analysis. In Simone Fischer-Hübner and Matthew K. Wright, editors, *Privacy Enhancing Technologies*, volume 7384 of *Lecture Notes in Computer Science*, pages 100–119. Springer, 2012.

[17] Philippe Golle and Ari Juels. Parallel mixing. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 220–226. ACM, 2004.

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

[19] Mathewson and Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In *International Workshop on Privacy Enhancing Technologies (PET), LNCS*, volume 4, 2004.

[20] Frank McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul, editors, *SIGMOD Conference*, pages 19–30. ACM, 2009.

[21] C Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125. ACM, 2001.

[22] Indrajit Roy, Srinath TV Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for mapreduce. In *NSDI*, volume 10, pages 297–312, 2010.

[23] Alan Rusbridger. The Snowden Leaks and the Public. The New-York Review of Book, November 2013.

[24] Victor Shoup. NTL: A Library for doing Number Theory. `http://www.shoup.net/ntl/`, 2016.

[25] Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In *Progress in Cryptology–AFRICACRYPT 2010*, pages 100–113. Springer, 2010.

[26] The Tor Project. Tor Metrics: Advertised bandwidth distribution. `https://metrics.torproject.org/advbwdist-perc.html?start=2016-02-15&end=2016-05-15&p=100&p=50`, May 2016.

[27] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In Ethan L. Miller and Steven Hand, editors, *SOSP*, pages 137–152. ACM, 2015.