Stadium: A Distributed Metadata-Private Messaging System

Nirvan Tyagi* Yossi Gilad†‡ Matei Zaharia§ Nickolai Zeldovich†

Abstract

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

This paper presents Stadium, the first system to provide metadata and data privacy while being able to scale its work efficiently across many servers. Much like Vuvuzela, the current largest-scale system that protects metadata, Stadium is based on differential privacy. However, providing privacy in Stadium is more challenging because distributing users' traffic across servers creates opportunities for adversaries to observe it in fine granularity. To solve this challenge, Stadium uses a collaborative noise generation approach combined with a novel verifiable parallel mixnet design where the servers collaboratively check that others follow the protocol. We show that Stadium can scale to support over an order of magnitude more users than the current state of the art, and cut the costs of operating each server.

1 Introduction

Private communication is difficult on today's Internet. While there are many applications that let users encrypt message *content*, encryption does not hide *metadata*: adversaries can still learn who is communicating with whom, at what times, and their traffic volumes. Metadata reveals a great deal of information: indeed, NSA officials have stated that "if you have enough metadata you don't really need content," [36] and "we kill people based on metadata" [28]. For users such as reporters, whistleblowers, and activists, higher privacy guarantees are 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 [12] and Dissent [13], require broadcasting all messages to all users or use computationally intensive Private Information Retrieval. Recently, Vuvuzela [41] introduced an approach for protecting metadata based on differential privacy [18]. 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 leakage 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. Overall, the system scales to support about 115,000 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 to a high operating cost for each server [41, §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 efficiently distributing its work over multiple servers. Stadium uses a similar differential privacy definition to Vuvuzela, but scales to support an order of magnitude more users. Although 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 over an order of magnitude higher throughput in total.

Stadium builds on a parallel mixnet design [27] where users select a fixed-length path (chain of servers) for each message through the pool of available servers. This lets Stadium scale linearly with the number of servers. Much like in Vuvuzela, in addition to forwarding user messages, each server also generates noise messages to hide traffic patterns.

*Cornell Tech
†MIT CSAIL
‡Boston University
§Stanford

However, the key challenge in Stadium is protecting the much higher number of observable variables its paths expose. Unlike in Vuvuzela, no single server can generate enough noise to hide all the variables, so Stadium must rely on many servers working together to generate enough noise. Moreover, malicious servers can *discard* the noise generated by other servers, or modify messages to learn about communication patterns. To address these problems, Stadium introduces a novel *verifiable parallel mixnet* design, allowing servers to self-verify that other servers are correctly preserving and mixing messages.

We implement a prototype of Stadium and deploy it on Amazon EC2. We show that Stadium can scale efficiently to hundreds of servers, enabling over an order of magnitude higher throughput than Vuvuzela. 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, to provide 1-minute messaging latency for 2 million users, each server in a Stadium deployment of 100 servers only needs to send at about 1.5% of a Vuvuzela server's traffic rate.

Bandwidth requirements are critical in practice. In today's Tor network, only 5% of relays (a few hundred servers) offer more than 100 Mbps of bandwidth, and none offer more than 1 Gbps [40]. With 100 Mbps per server, Stadium can scale to over 500,000 messages per second over hundreds of servers, while Vuvuzela would be limited to around 8,000 messages per second (namely, 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.

Finally, we formally analyze Stadium's security using differential privacy, taking into account the much greater number of observable variables in the system. We show that Stadium can achieve similar guarantees to Vuvuzela (continued privacy even if a user sends tens of thousands of messages), in the presence of powerful 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 an order of magnitude higher throughput than prior systems.
- A novel verifiable parallel mixnet design built around efficient protocols for verifiable shuffles and verifiable message distribution.
- An analytical evaluation of Stadium using differential privacy.

2 Goals

In this section we present Stadium's key design goals: providing private messaging in face of powerful adversaries while scaling to tens of millions of users.

2.1 Threat Model

Stadium assumes an adversary that controls some fraction of its mixing servers. The system may be deployed to resist any fraction of compromised servers (but efficiency degrades as Stadium deploys to cope with more malicious servers). The adversary may also control any number of users. Adversary-controlled servers and clients may deviate in any way from Stadium's protocol, but non-compromised servers and clients run bug-free implementations and are not vulnerable to side-channel attacks. We design Stadium to resist passive and active attacks, that is, we allow the adversary to monitor, block, delay, or inject traffic on any network link at all communication rounds.

In terms of availability, Stadium is not resistant to wide-scale denial of service (DoS) attacks. This is unavoidable given our assumption that adversaries can block traffic, which allows them to disconnect servers and users from the network. However, Stadium guarantees that any DoS attack will not risk its users' privacy.

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 under the random oracle model [4]. We further assume a public key infrastructure, i.e., Stadium servers' public keys are known to its users, and that two communicating clients hold a shared key. This can be performed using via Alpenhorn [31], a system that allows two users to privately coordinate a shared secret, or out of band.

2.2 Privacy

Stadium has similar a privacy goal to Vuvuzela [41]. It aims to prevent adversaries from distinguishing between communication patterns of its users, even if the users exchange many messages.

Informally, Stadium provides the following privacy guaranty for users communicating through non-compromised clients: for any user, call her Alice, the adversary should not be able to tell whether Alice communicates with Bob or another random user, or even identify when Alice does not communicate at all. We design Stadium to keep this guarantee even if the attacker corrupts some servers and observes traffic flowing through the system for a long time. We use the following definition from the differential privacy literature to analyze Stadium's privacy guarantees [19]:

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,* $\Pr[M(x) \in S] \leq e^{\epsilon} \cdot \Pr[M(y) \in S] + \delta$.

The inputs to the algorithm (x and y) are the users' communication actions in each round. We consider two inputs *adjacent* if they differ only by one user's, say Alice's, actions. One of the inputs represents Alice's real actions in a particular round (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). Real or not, the differential privacy definition requires that all these stories will appear almost as plausible.

2.3 Scalability

Our scalability goal is to support tens of millions of simultaneous users on Stadium. This is a comparable level to Tor, the most popular anonymity service available today. To support a growing number of users, the system must allow for *incremental deployment*. Namely, its throughput should increase linearly with the number of servers. The scalability goal therefore necessitates that only a small subset of the servers will process each message, and represents a departure from previous metadata-private systems.

3 Overview

In this section we present an overview of Stadium's design and motivate the different components in the system.

Sending and receiving messages. Communication through Stadium takes place in rounds. Every fixed interval, a new round begins to process a set of messages Stadium accumulated from its clients. Incoming messages are shuffled by Stadium's distributed servers to unlink them from their sender. Stadium borrows a *dead-drop* communication strategy from Vuvuzela [41] amenable to provable differential privacy. Dead-drops are virtual locations, hosted on the system's servers, that are associated with the anonymous conversations taking place. Dead-drop locations are revealed after the messages are shuffled. When exactly two messages reach the same dead-drop in a communication round, the server hosting that dead-drop exchanges their content. Finally, messages are sent back through Stadium, where servers invert the shuffle and return messages, such that two messages exchanged at a dead-drop reach their intended recipients. To ensure that the number of messages does not leak whether a user communicates, clients send exactly one message every round (if the user does not communicate, the client sends a dummy message which will route back to the sender). An overview of Stadium's design is illustrated in Figure 1. In order to communicate, Alice and Bob use a separate *dialing* protocol, which allows them to coordinate a dead-drop for every round as well as a symmetric key to encrypt the content of their communication. One recent scalable system for this task is Alpenhorn [31].

Parallel mixing. Given Stadium's threat model that allows compromised servers, shuffling messages securely and efficiently is a significant challenge. Traditional mix chains, where each server processes all messages, fail to perform at Stadium's targeted scale. Instead, Stadium employs a parallel mixing scheme, where each server processes a fraction of the input messages, shown in Figure 2. Messages in Stadium travel through one of multiple paths afforded by the system. In each round, the user selects a random path for its message through the system. A path selection consists of picking an *input* and *output* mixing chain. In the input chain, messages are mixed with all other messages that entered the system through the same chain. Messages exiting the input chain are distributed to output chains, where they are again mixed, this time with messages arriving from all input chains. The last server of the output chain emits messages to their dead-drops. Parallel mixing allows Stadium to efficiently process messages traveling through different chains in parallel, however, comes at the cost of imperfect mixing. The connections between input chains and output chains restrict the possible output permutations. Stadium addresses these concerns through means of differential privacy.

Verifiable Processing. Stadium provides differential privacy for user communication by adding noise in the form of fake messages. It is infeasible for a single server to generate enough noise for the entire system, so Stadium servers

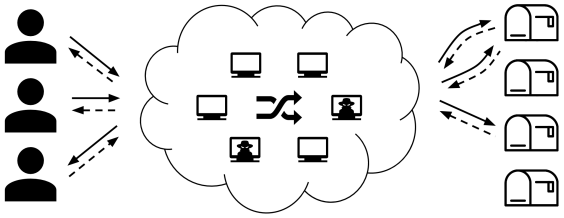

Figure 1: Stadium overview. Users send messages for a communication round. Stadium's servers work together to shuffle the messages, verifying each other for correctness. Shuffled messages are exchanged at dead-drops and reversed through the system to the recipient user.

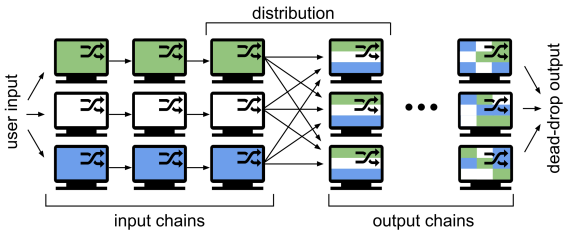

Figure 2: Stadium's distributed parallel mixing. Users determine a message path by picking an input chain and output chain.

must rely on each other for collaborative noise generation. Additionally, malicious servers may attempt to discard noise messages before they are mixed with user messages. To mitigate this risk, Stadium uses a number of verifiable processing techniques to allow honest servers to verify that others follow the parallel mixing protocol. Messages input to the system are checked using a cryptographic proof of knowledge. Mix chains provide proofs of permutation using verifiable shuffles, a cryptographic primitive common to e-voting. Finally, a verifiable distribution primitive based on cryptographic signatures allows servers to ensure all messages are correctly collected following the distribution step. Since verifiable processing is expensive, Stadium takes care to limit its use to only when necessary.

Presentation outline. Section 4 describes the round setup. Section 5 describes the verifiable processing pipeline. Stadium's provable privacy guarantees are shown in Section 6 and Section 7. Section 8 extends Stadium for fault tolerance. Section 9 and Section 10 describe our implementation prototype and evaluation.

4 Round Setup

In this section we describe the round-setup mechanisms. Specifically, how servers arrange in mix chains and establish per-chain keys, and how clients encapsulate messages before emitting them to the system. We focus on the case where fault-tolerance is not needed (e.g., in smaller deployments). Section 8 discusses possible adaptations to tolerate faults.

4.1 Arranging in Mix Chains

At the beginning of every round, participating servers jointly decide on a random seed which sets their arrangement into mix chains. Using this seed servers form in fixed-length mix chains (see illustration in Figure 2). Each server appears exactly once in each of the available positions inside the chains, this guarantees that no server will need to process more than one message batch simultaneously. In Appendix A we describe the algorithm for arranging in these chains given the random seed. To simplify our presentation we refer to a chain by its first server, i.e., chain i is the chain that begins at server i (this is correct since each server appears first at exactly one chain). By reordering the servers at every round, Stadium makes it simple to add new servers to the system and supports incremental deployments, and also makes it challenging for attackers to compromise entire chains (rounds are short).

It is important to ensure 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 the commitment-based protocol as outlined in [7]. To prevent the adversary from modifying coordination messages, the seed-selection protocol runs over secure channels (e.g., using TLS) established using the servers' long-term public keys which are distributed via a public key infrastructure (see Stadium's assumptions in Section 2.1).

4.2 Round Keys

Each server creates an ephemeral public/private ElGamal key pair for each round, all servers in the network use the same cyclic group $\mathcal{G}$ and generator g (e.g., configured into the server/client applications). Let us denote the private key

of server i by s_i, so the public key is g^{s_i}. A client retrieves the public keys from all servers, and computes a public key for each mix chain, which is the product of the public keys in the chain: for mix chain i, the client computes $pk_i = \prod_{i \le j \le i+l} g^{s_j} = g^{\sum_{j=i}^{i+l} s_j}$.

Notice that while clients perform the regular ElGamal encryption algorithm (using public key pk_i), all servers must contribute decryption shares using their own private keys (i.e., decrypt using s_i) [20]. The product of all decryption shares reveals the plaintext.

4.3 Message Encapsulation

Users decide on a path to send their message through the mixnet by designating an input chain, an output chain, and a dead-drop. We consider a user's message as two separate components. The *metadata* consists of the path information (i.e., input, output chain and dead-drop). The *content* component consists of the actual data the user wants to send to the recipient.

Stadium uses verifiable processing to ensure that messages are not tampered with throughout the protocol. Since this processing is expensive, Stadium aims to limit its use. User communication patterns are revealed by the metadata component of their message (e.g., two messages sharing a dead-drop are in a conversation). On the other hand, the content component does not reveal any information about the recipient since it is padded to fixed length and encrypted with a key shared between the users. Thus, Stadium performs verifiable processing only on the metadata component of the message.[1] This provides significant savings in processing time since the content size is larger than the metadata. To accommodate the different processing requirements between the metadata and content, the components are encrypted using different schemes.

Metadata. The metadata is further split according to the mixing chain. The input chain's metadata holds the output chain's id. The output chain's metadata holds the dead-drop id, and is included inside the content component for the input chain. It is therefore important to bind the content of the input chain to its metadata (since that content later sets the destination dead-drop). To do that, the input chain's metadata also includes a symmetric key, which the client uses to authenticate the input chain's content. The symmetric authentication code is included in the content and verified at the end of the input chain by its servers (we provide the details in Section 5). The metadata is encrypted with the mixing chain's public key and revealed at the end of the chain to find where the content should be distributed to. Public key encryption of the metadata is necessary for verifiable shuffling.

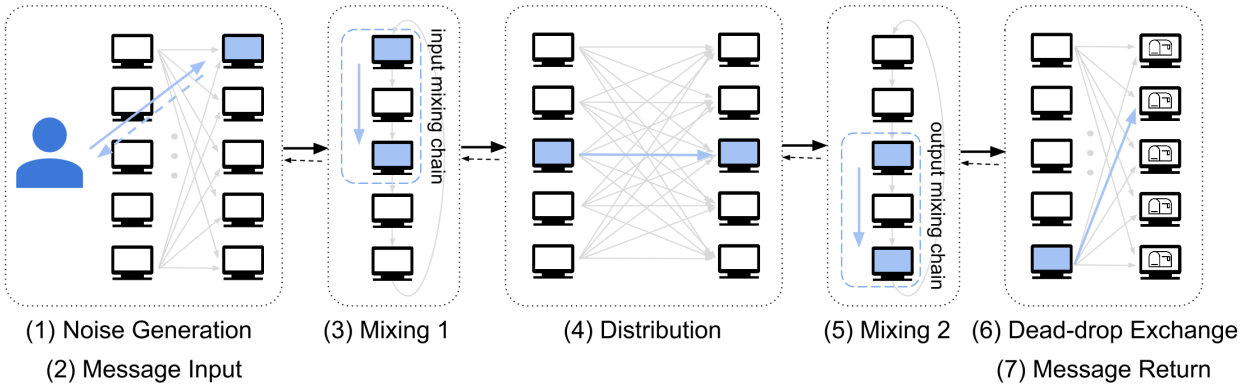

Figure 3: Stadium's design through 7 processing phases. Noise is generated and entered with user messages enter in phases 1 and 2. The messages go through the input chain in phase 3, where each node performs a verifiable shuffle; then they are distributed in phase 4 to the output chain, where they mix with messages from all input chains in phase 5. Messages reaching the same dead-drop are exchanged in phase 6. Lastly, messages are returned to users by reversing shuffle permutations in phase 7, denoted by the dashed arrows.

[1] Since the content component is not verifiably processed, malicious servers can corrupt it and deny service, but this falls well within the scope of Stadium's threat model (e.g., malicious servers can block the network).

Content. The content is encapsulated in order to allow servers to re-randomize the output as they shuffle messages. If servers did not re-randomize the content, the secret permutation would be revealed simply by observing the input and output content batches. To encapsulate the content, clients use a hybrid onion encryption scheme. For each server on the chain, starting from the last, the client encrypts the content using a fresh ephemeral symmetric key and then encrypts the symmetric key using the server's public key. The output chain metadata is added inside the onion encryption layer of the last server in the input chain (i.e., it is revealed before the first server of the output chain). Servers store the ephemeral symmetric keys revealed in the forward direction in order to re-randomize the message in the return direction.

5 Verifiable Processing Pipeline

In this section we present the mechanics behind Stadium's message processing following the phases in Figure 3.

To provide differential privacy independently of other users' traffic patterns, Stadium servers collaboratively generate noise messages (i.e., cover traffic) when the round begins (phase 1 in Figure 3). Efficiently generating noise to provide Stadium's privacy guarantees is the focus of Section 6. This section focuses on the verifiable processing of messages inside the Stadium, where the message batch is decided and messages are mixed though the input chain, distribute to output chains, and then mixed again (phases 2-5 in Figure 3). At the end of the verifiable processing pipeline, messages reach the dead-drops and are exchanged as described in Section 3. Messages then travel back through the mix chains to their recipient user (phases 6-7 in Figure 3). These two phases are not verified, since when messages reach the dead-drops they are already anonymous.

In-chain broadcasts. Our processing below uses message broadcasts, which are materialized by sending a unicast message to each recipient. To provide a viable design, we restrict broadcasts in the processing pipeline to the scope of *one mix chain*. Thus the communication cost of each broadcast is constant (the length of the chain is fixed, and independent of the number of servers). Importantly, our design copes with malicious senders who send different messages to subsets of the recipients.

5.1 Message Input

Stadium must ensure that adversaries do not tamper with messages as they enter the system, or attackers could create traceable patterns before the messages mix. For example, an attacker that injects two duplicates of a user's message can learn its destination by observing which dead-drop is accessed at least 3 times. Since Stadium uses malleable encryption to facilitate message re-randomization after shuffling (see Section 3), attackers can inject seemingly different copies of the same message.

Verification. The servers in the input chain verify, in zero-knowledge, that the sender knows the plaintext of the message emitted to Stadium. Thereby ensuring that the message is not a result of attacker forgery. Given our encapsulation technique, it is sufficient to verify that the sender knows the metadata for the input chain, since this means that the sender knows the output chain and can decrypt the dead-drop ID. Namely, we verify that the sender had set the path of this message. The metadata is encrypted with the input chain's ElGamal public key. To prove ownership of the message, senders compute a proof of knowledge of the plaintext [30] converted via the Fiat-Shamir heuristic to be non-interactive [22] and attach the proof to their messages.

The first server in the input chain receives the messages and proofs, and broadcasts the message batch to all servers on its chain to verify. Each server discards duplicates and messages without proofs, and then keeps the remainder as the *message-batch* for this communication round.

5.2 Shuffle

In the mixing phases (phases 3 and 5 in Figure 3) each server shuffles the message batch that it has, proves that it processed the messages correctly, and passes the shuffled messages to the next server on the chain. See Figure 4. The shuffling procedure also randomizes the outputs such that an attacker observing both inputs and outputs cannot learn the permutation and invert it. (To randomize the metadata, encrypted with ElGamal, we take advantage of the malleability property of ElGamal cypher-texts.)

To shuffle the metadata we use a verifiable shuffle [3]. This allows the shuffler to prove to all other servers on its chain that the output is a re-randomized permutation of the input without revealing any knowledge about the permutation

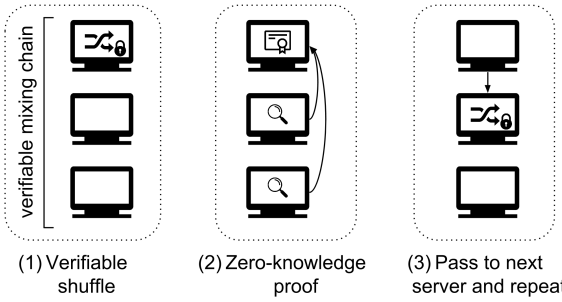

Figure 4: Verifiable shuffle in three steps.

itself (i.e., a zero knowledge proof). If one of the servers does not accept the proof, then it does not contribute its decryption shares, thus blocking all messages in the batch from continuing their path through the mixnet and terminating the communication round. 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 the communication round is aborted (so as to avoid compromising users' privacy).

The content of messages is shuffled under the same permutation as its metadata, but needs not be verifiably shuffled, since it does not determine the message's path and destination. (Unlinking the content of pre- and post-shuffle messages happens automatically when the mix server peels its layer of the hybrid encryption, see Section 4.3.)

Verification. The verifiable shuffle technique in [3] allows using the Fiat-Shamir heuristic [22] to avoid interactions between the prover and verifier. This allows the shuffling server to compute the proof and then broadcast its output to all other servers in the chain who confirm that it is some permutation of the message batch.

Decryption at the end of the chain. After the last shuffle in the mix chain is verified, each server computes the decryption share for each message's metadata and sends it to the server at the end of the chain. If all servers in the chain contributed shares, then the last server can decrypt the messages' metadata (by multiplying all shares, see Section 4.2) to reveal the next hop in the message's route (an output chain or a dead-drop).

5.3 Distribution

In the distribution phase (phase 4 in Figure 3) the servers on the output chain receive the *parcels* of the message batch from the input chains. The distribution phase ensures that all messages were correctly distributed. Namely, that (1) the last server on the input chain does not modify messages before it distributes them to their output chains, and (2) the servers starting the output chains only incorporate valid messages into their new message batch.

The verifiable distribution protocol, illustrated in Figure 5, therefore has two steps. First, all servers in the input chain agree on the parcel (message set) that they distribute to each of the output chains. Second, all the servers in the output chain verify that they received valid parcels (i.e., that parcels include exactly the messages that the servers in the input chain agreed on).

Agree on parcels (input chain). To agree on valid parcels for distribution to each output chain, the last server in the input chain broadcasts the decrypted messages to the other servers on its chain. The servers verify that decryption is correct[2]. The servers also verify that the content (i.e., data relayed to the output chain) is correctly authenticated by deriving the symmetric authentication key from the metadata (see Section 4.3). If a message's content is invalid, it is discarded (so as not to risk user privacy) and the servers reveal their re-randomization factor for that particular message, allowing to detect whether one of the servers diverted from the protocol. In this case honest servers in the input chain identify the malicious server, and discard the message batch to prevent the round from moving forward.

Each server in the input chain that verified the message batch uses the message's metadata to divide the batch into the parcels that will distribute to the output chains. Servers in the input chain sign the parcels and send their signatures to the server at the end of their chain. If that server did not modify the set of messages, then all servers in the input

[2]The last server in the input chain broadcasts the decrypted metedata for each message together with the randomness used to encrypt it. Servers in the input chain then check that encryption with the chain's public key yields the original input set.

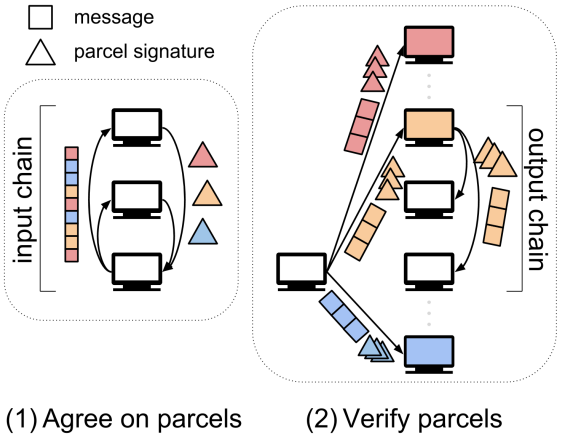

Figure 5: Verifiable distribution in two steps.

chain sign the same parcels.[3]

Verify parcels (output chain). The server at the end of the input chain sends each parcel, together with the signatures, to the server that starts the parcel's output chain. That server then broadcasts the message batch and signature to the rest of the servers in the output chain, which validate the signatures over the parcels from each input chain.

6 Hiding Observable Variables

In this section we identify the observable variables that Stadium exposes, and describe how servers generate cover traffic to obscure these variables.

6.1 Observable Variables

Attackers may monitor the communication between all of Stadium's servers and users. Since all messages that enter a mix chain must also exit that chain (since message processing is verifiable), it is sufficient to analyze the information leak in the following types of links: (1) from the user to the input chain, (2) from the input chain to output chain, and (3) from the output chain to the dead-drop. (On the return path the origin of a message is already anonymous). We design Stadium to minimize the information leaked to attackers through these links.

First, clients send a message at every round regardless of whether the user is active in a conversation. Therefore, observing a user sending a message does not leak any information about their conversations.[4]

Second, adversaries may monitor the traffic emitted from the user's input chain. Although the traffic volume from the input chain does not directly imply the user's communication patterns, attackers may still use this information to infer on the user's output chain. By identifying a user's output chain and monitoring the traffic emitted from it, the attacker may learn the users communication patterns, as we next describe.

Third, adversaries may monitor the link between the output chain and the dead-drop. They can then identify when two messages reach the same dead-drop, indicating that the dead-drop is employed in a conversation, whereas if only one message reaches the dead-drop in a conversation round, then that means it is used by an idle user (who is not currently involved in a conversation). If a message is sent to a dead-drop hosted on an adversary-controlled server, the adversary learns the dead-drop access count as well as the identity of the server that outputs the message to the dead-drop (at the end of the parallel mixnet).

We therefore define three classes of observable variables that are affected by changes in user communication pattern:

Input-output chain traffic, IO_i^x. The amount of traffic emitted from input chain x to output chain i.

[3] Messages within a parcel are ordered and then signed, such that hash computation is consistent across servers.

[4] Stadium does not attempt to obscure the fact that a user uses the system, it only hides its users' communication patterns.

Single access variables, α_i. The number of dead-drops that receive exactly one message where that message was output from chain i. There are m single access variables.

Double access variables, $\beta_{i,j}$. The number of dead-drops that receive two messages, where one message was output from chain i and the other from server j. There are $\binom{m}{2} = \frac{m^2-m}{2}$ double access variables.

6.2 Noise Distributions

In order to obscure the observable variables that Stadium exposes to attackers, servers inject noise messages to the system. We define the following two categories of random variables. The single-access noise variable α_i^x is the number of messages that travel through input chain x and output from server i and reach a random dead-drop (such that the chance that it is used by any other message is negligible). The double-access variable $\beta_{i,j}^{x,y}$ is the amount of pairs of noise messages that reach the same dead-drop, where one message in the pair travels through input chain x and output chain i and the other travels through input chain y and output chain j.

The servers draw each of these variables independently. We cannot produce negative noise (i.e., remove user-messages), nor can we send use a send a fraction of a noise message. Hence we must use a discrete non-negative distribution.

Stadium uses the Poisson distribution for noise generation, $\alpha_i^x \sim \text{Pois}(\lambda_1)$ and $\beta_{i,j}^{x,y} \sim \text{Pois}(\lambda_2)$. The Poisson distribution is well suited to Stadium for two main reasons. First, the additive property[5] of Poisson distributions makes it easy to reason about the collaborative noise distribution from summing independent samples generated by Stadium servers. Second, since Poisson distributions are discrete and non-negative, we do not need to handle rounding, especially for distributions with mean near zero. In Section 7 we select the parameters λ_1, λ_2 such that the aggregate noise (generated by all servers) provides Stadium's privacy guarantees.

Noise covering IO_j^i. Each server generates noise to cover the IO_i^x variable via α_i^x and $\beta_{i,j}^{x,y}, \beta_{j,i}^{y,x}$ (for any y, j, i.e., all noise messages that travel via input chain x and output chain i). Using the distribution's additive property we find: $\alpha_j^i + \sum_{x,y} \beta_{j,y}^{i,x} + \beta_{y,j}^{x,i} \sim \text{Pois}(\lambda_1 + 2m\lambda_2)$. Multiplied by the number of servers, we find that IO_i^x is covered by noise that distributes $\text{Pois}(m\lambda_1 + 2m^2\lambda_2)$.

Noise covering α_i. Each server generates noise to cover this variable via the α_i^x variable (for any x, i.e., all single-access messages emitted from output chain i). Therefore the amount of noise covering the observable variable α_i distributes $m \sum_x \alpha_i^x = Pois(m^2\lambda_1)$.

Noise covering $\beta_{i,j}$. Similarly, we find that the amount of noise that each server sends for covering the double-access observable variable $\beta_{i,j}$ is $\sum_{x,y} \beta_{i,j}^{x,y} + \beta_{j,i}^{y,x}$ and using all servers in the system that noise distributes $Pois(2m^3\lambda_2)$.

7 Privacy Analysis

In this section we analyze Stadium's privacy guarantees. Our differential privacy goal (defined in Section 2) is to keep any adjacent communication instances almost equally likely. Where adjacency means that one client changes its traffic pattern: by selecting a different output chain or a different dead-drop (or both). For this purpose Stadium servers inject Poisson distributed noise messages to the system (see Section 6). We therefore begin our analysis by finding the differential privacy guarantees that Poisson noise provides.

Theorem 1. *A mechanism with $Pois(\lambda)$ noise is (ϵ, δ)-differentially private with $\epsilon = \ln(1 + \frac{c\sqrt{\lambda}+1}{\lambda})$ and $\delta = Pois(\lambda; \lambda - c\sqrt{\lambda}) + Pois(\lambda; \lambda + c\sqrt{\lambda})$. (Where a positive value c that allows to trade higher ϵ for lower δ.)*

Proof. Given in Appendix B. □

We next analyze the privacy guarantees that the input and output chains provide, compose them together and analyze the guarantees in case that some servers are malicious and over many communication rounds.

[5] If $x \sim \text{Poison}(\lambda_1)$ and $y \sim \text{Poisson}(\lambda_2)$, then $x + y \sim \text{Poisson}(\lambda_1 + \lambda_2)$.

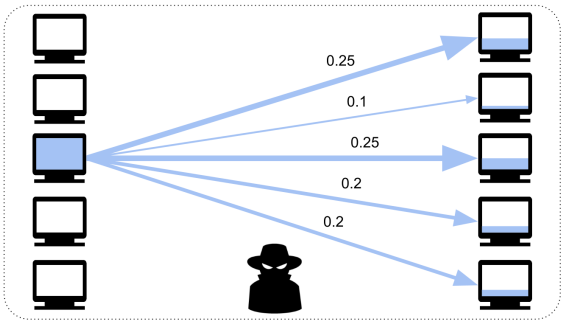

Figure 6: Adversary gains information about Alice's message output chain by observing a non-uniform distribution to parcels in her input chain.

7.1 Input-chain

Adversaries can monitor the users' communication links and observe the input chains that they select. The purpose of the input chains is to hide each user's selection of output chain. In this subsection we analyze the guarantees of the input chain in terms of differential privacy. We consider two communication instances to be adjacent if they differ by the output chain selection of one user, call her Alice.

Clients select output chains uniformly. Since Stadium makes m chains available to users (m is the number of servers), the probability that Alice uses a specific output chain in a particular round is $\frac{1}{m}$. However, the adversary can observe the traffic volumes from Alice's input chain to all output chain and gain statistical information about which output chain Alice is more likely to have used, see Figure 6. This information is captured by the observable variable IO_i^x defined in Section 6.1, where x is Alice's input chain and i her output chain. The amount of noise obscuring this variable distributes Pois$(m\lambda_1 + 2m^2\lambda_2)$ (see details in Section 6.2). We apply Theorem 1 to find the input chain's ϵ, δ guarantees.

7.2 Output Chains

Output chains mix user messages with noise to obscure dead drop access patterns, namely adversaries that manage to corrupt dead drop-hosting servers can learn from which output chain each dead drop is accessed. If they know Alice and Bob's output chains, they can then find whether any dead drop was accessed from these chains and gain some information on the existence (or nonexistence) of communication between them. Let i denote Alice's output chain. Leakage of this information is captured by the observable variables α_i (in case that Alice does not communicate), and $\beta_{i,j}$ (in case that she communicates with Bob, who uses output chain j).

In Section 6.2 we find that the amount of noise covering α_i distributes Pois$(m^2\lambda_1)$, and the amount of noise covering $\beta_{i,j}$ distributes Pois$(m^3\lambda_2)$. We apply Theorem 1 to find the ϵ, δ guarantees for both types of observable variables.

7.3 Input and Output Chain Composition

The access patterns to dead drops from output chain i only change when a user or his/her peer sends a message via output chain i. Users select an output chain uniformly out of m available chains, thus the probability of a user to select any particular output chain decreases linearly with the number of servers in the system. The adversary might monitor users' input chains to infer on their output chains. The differential privacy analysis in Section 7.1 allows to bound the adversary's advantage. Let ϵ_x, δ_x denote the differential privacy properties of input chains. The probability of an adversary to guess Alice's output chain by observing communication on her input chain is thus bounded by $p = \frac{e^{\epsilon}_x}{m} + \delta_x$. This probability captures the adversary's uncertainty of a message's route. The following theorem allows to adjust the differential-privacy guarantees provided by Stadium's output chains given the adversary's uncertainty.

Theorem 2. *A mechanism that provides (ϵ_k, δ_k)-differential privacy with probability p_k (where $\sum_k p_k = 1$), is (ϵ', δ')-differentially private where $\epsilon' = \ln(\sum_k p_k e^{\epsilon}_k)$ and $\delta' = \sum p_k \delta_k$.*

Proof. Given in Appendix C. □

In our case, the output chain mechanism provides (ϵ_i, δ_i)-differential privacy if the attacker knows to associate the output chain to the communicating user, i.e., with probability p. In contrast, if the attacker does not succeed to make this association, i.e., with probability $1-p$, the monitored output mechanism is agnostic to the user's traffic, i.e., ϵ and δ are zero. We find that the combination of input and output chain therefore provides $\epsilon' = \ln(1 - p(e^{\epsilon_i} - 1))$ and $\delta' = p\delta_i$ differential privacy.

Single-access observations. The differential privacy guarantee of Alice's the input chain x suggests that the attacker learns her output chain with at most probability $p = \frac{e^{\epsilon_x}}{m} + \delta_x$. Denote by ϵ_1, δ_1 the differential-privacy properties of Stadium's output chain for hiding the single-access patterns to the dead drop. We adjust these values using Theorem 2 and find that: $\epsilon_1' = \ln(1 - (\frac{e^{\epsilon_x}}{m} + \delta_x)(e^{\epsilon_1} - 1))$, and $\delta_1' = (\frac{e^{\epsilon_x}}{m} + \delta_x)\delta_1$.

Double-access observations. Alice's output chain affects m observable variables, one for each possible output chain that her peer may choose. Therefore guessing the double-access observable variable affected by Alice's message is more challenging than just guessing her output chain. We bound the probability that adversaries succeed in guessing the correct variable to monitor by providing them additional knowledge, let us assume that an adversary knows that if Alice is talking with someone, then her peer is Bob. Therefore, the adversary only cares about finding Alice and Bob's output chains. The probability for that is bounded by $2p^2$. Denote by ϵ_2, δ_2 the differential-privacy properties of Stadium's output chain for hiding the double-access patterns to the dead drop. We adjust them using using the above theorem and find that $\epsilon_2' = \ln(1 + 2(\frac{e^{\epsilon_x}}{m} + \delta_x)^2(e^{\epsilon_2} - 1))$, and $\delta_2' = 2(\frac{e^{\epsilon_x}}{m} + \delta_x)^2\delta_2$.

7.4 Malicious Servers

Our analysis above assumed that all servers contribute their fair share of noise to Stadium. However, malicious servers may deviate from the protocol and refrain from injecting noise. In order to generate enough noise to provide the guarantees above, despite an expected f-fraction of malicious servers, each honest server uses $\lambda_1' = \frac{\lambda_1}{f}, \lambda_2' = \frac{\lambda_2}{f}$ as the means for generating single- and double-access noise.[6]

Our analysis also assumes that in every chain there is at least one honest server that correctly shuffles messages. However, experience suggests that for wide-scale deployments this is often not the case (e.g., see for Tor [15]). Instead our model assumes that a server is corrupt with probability f, and therefore a chain of l servers is corrupt with probability f^l. Attackers may get lucky, and at some communication rounds corrupt multiple chains, while in other rounds it might happen that no chain turns corrupt (i.e., each chain has one honest server). We compute the probabilities for each combination of number of corrupt input and output chain and adjust Stadium's differential privacy guarantees ($\epsilon_1', \delta_1', \epsilon_2', \delta_2'$ above) using Theorem 2.[7]

7.5 Communication Rounds

Our system exposes multiple variables, specifically, adversaries can observe the single-access pattern to dead drops from any of the m output chains, and the double-access pattern to dead drops from any of the $\frac{m^2}{2}$ pairs of output chains. To compose these variables together we leverage the following key property of Stadium.

Theorem 3. *Between any two adjacent inputs there are at most two single access variables and two double access variables that change.*

Proof. Given in Appendix D. □

The theorem above bounds the number of variables that may leak information to the adversary at two single access variables and two double access variables. All other variables do not change (and therefore do not leak information). We compose these variables together using the Sequential Composition theorem in [33]. The theorem allows us to calculate the differential privacy guarantees of Stadium when the adversary can observe multiple variables by summing their ϵ and δ parameters. We find that for a single round Stadium provides $\epsilon = 2(\epsilon_1' + \epsilon_2'), \delta = 2(\delta_1' + \delta_2')$ differential privacy.

[6] Attackers may also deviate from the protocol by generating noise according to another distribution, but we assume that attackers can always identify and discard their own noise. Thus we only care for noise generated by honest servers.

[7] In practice, to efficiently compute Stadium's guarantees we assume that if more than 10% of the chains are compromised, then all chains are compromised.

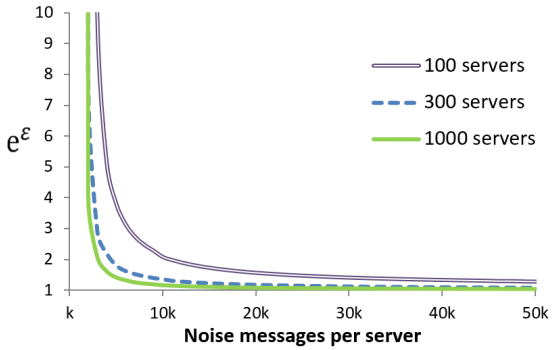

Figure 7: Stadium privacy guarantee after 10^5 communication rounds as a function of noise per server.

Conversation over multiple rounds. Stadium allows users to interactively communicate over multiple communication rounds. Adversaries may constantly monitor the system to learn information about users across multiple rounds, 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 [19] and a similar challenge was handled by the Vuvuzela system [41]. The composition of k rounds is also differentially private, as we next describe in Theorem 4. The parameter d allows to trade higher ϵ for lower δ.

Theorem 4. *Consider an algorithm M providing ϵ, δ differential privacy, then M provides ϵ', δ' differential privacy after k rounds with parameters: $\epsilon' = \epsilon\sqrt{2k\ln(1/d)} + k\epsilon(e^{\epsilon} - 1)$ and $\delta' = k\delta + d$, for any $d > 0$.*

Proof. Direct from Theorem 3.20 in [19]. □

7.6 Noise Volumes in Practice

We apply the analysis above to find the best noise distribution (i.e., the parameters λ_1, λ_2) for deployments of $m = 100, 300$, and 1000 servers, and for different budgets of noise messages sent by each server. Namely, given a strict limit for $\delta \leq 10^{-4}$, wish to minimize e^{ϵ} (by picking the selecting λ_1, λ_2).

Figure 7 plots our results after users communicate through 10^5 rounds. In this plot Stadium chains have 8 servers, and the system is deployed to resist 25% of colluding servers. We find that with only 10K noise messages per server, a 100-server Stadium deployment ensures that Alice talking to Bob is no more likely than twice as likely than Alice not talking to anyone. (A comparable privacy guarantee with Vuvuzela requires $20\times$ the amount of noise per server.) The amount of noise that each server needs to generate decreases with deployment size, converging to about 4K noise messages per server with a 1000 servers deployment.

8 Fault Tolerance Extension

Fault tolerance becomes increasingly important as Stadium scales to utilize more servers. Namely, users should be able to communicate despite a few servers going abruptly offline. Stadium naturally recovers from such faults at the next communication round, since the faulty server will not participate in the round setup protocols (see Section 4). We next extend our design to function despite some servers disconnecting in the *middle of the round*.

To allow the system to recover from faults, each server stores shares of its private ephemeral round key at a designated set of *recovery* servers [37]. Servers in the recovery set keep a connection with their subject to identify when it goes offline. When a server in the recovery-set realizes this event, it sends its share of the failing server's key to others. Only if all servers in the recovery set provide their shares, then the server's ephemeral round key is recovered and allows a peer server, e.g., the predecessor of the failing server in the mix chain, to take the failing server's place in the processing pipeline (i.e., decrypt the metadata and content).

Balancing security and efficiency. Fault tolerance allows a colluding "recovery set" of servers to reveal a honest server's private key, and therefore remove the noise it introduces to the system, revert its shuffles, etc. Using a large,

randomly chosen, set of servers for recovery allows for better security, yet costs in efficiency since these servers must monitor the subject to identify when it fails. To balance these two requirements we set a server's recovery-set to be all other servers included with it in mix chains. These servers keep a connection with the subject server anyway in order to prove and verify shuffles through the round; monitoring piggybacks on these sessions.

Importantly, since our threat model allows the attacker to disconnect the link between any pair of servers, attackers can exploit the fault tolerance mechanism to persuade even an honest recovery-set to disclose a server's key by disconnecting it. To mitigate this risk each server in the recovery set may disclose a secret share for no more than *one* server per chain in each round. Therefore, if a server identifies that one of its chains has two faulty servers, the round is aborted (Stadium recovers at the next round, as we described above).

9 Implementation

We implemented a prototype of Stadium to evaluate its performance and feasibility of deployment. Our system's control and networking logic is implemented in Go, while the underlying verifiable processing protocols, described in Section 5 are implemented in C++. In particular, our C++ code implements the verifiable shuffling protocol in [3] extended for non-interactive proofs via the Fiat Shamir Heuristic and instantiated over a 1536-bit prime order group. We believe that our processing efficiency may be improved by instantiating the protocol over elliptic curves.

Our implementation uses OpenMP to parallelize different parts of the processing pipeline. Most steps of the verifiable computations independently process each message in the batch (see Section 5), and can therefore take advantage of multiple available cores.

The implementation of the system logic contains less than 3000 lines of Go and C++ code (atop of existing libraries that our implementation uses, such as NTL [38]). We deploy our prototype over 10 servers, with chain length of 8 servers, on Amazon EC2 to ensure that it can be feasibly deployed on today's hardware and test the correctness of our implementation. Our prototype currently does not support message input, threshold decryption, or fault tolerance.

10 Evaluation

We evaluate the performance of Stadium by extrapolating the performance of our 10 server, chain length 8 deployment to larger deployments. Almost all inter-server communication in Stadium is with respect to the chain length, with the exception of the distribution phase. We find that the CPU utilization is the dominant factor in Stadium's performance, and network I/O usage can be overlapped with this processing (i.e., one message batch is transmitted while another is being processed). Hence we expect the latency of a 10 server, chain length 8 deployment to provide a good estimate of the system's real performance even for large deployments with chain length 8. In the future, we would like to deploy our system at larger scale to remove the need for extrapolation.

Single server performance. We first measure the time of a single mix-server to process a 100K message batch size. We evaluate the performance with different number of available cores. Our results, illustrated in Figure 8, show that Stadium can significantly benefit from better server hardware. Processing times show a linear speedup, and provide evidence that Stadium will continue to scale beyond the 36 cores currently available for EC2 machines.

Scaling to large deployments. We now turn to evaluate different deployments of Stadium and compare with Vuvuzela. In order to evaluate the performance, we use the Amazon EC2 `c4.8xlarge` VM, which has 36 cores. We use the results in Figure 7 to find the amount of noise required for each deployment scenario and incorporate it into the input message batch for each server.

We measured how the message batch size affects Stadium's latency (length of the round). The more servers that are available to the system, the more distributed the user message load becomes. We compare our results, illustrated in Figure 9, to Vuvuzela running with a chain of 3 servers (plotted as dashed line in the figure). We find that that Stadium can scale to support hundreds of millions of users with reasonable latency of 80 seconds. In contrast, Vuvuzela does not gain from additional servers. It has a very steep incline in latency compared to Stadium's deployments and with a similar latency (80s), Vuvuzela can only support about 3.5 million users.

Deployment cost. To support 2 million users with 55 seconds latency, each of Vuvuzela's servers sends at rate 1.3Gbps per server (see results in [41]). In contrast, to support the same number of users and latency, under deployment of 100

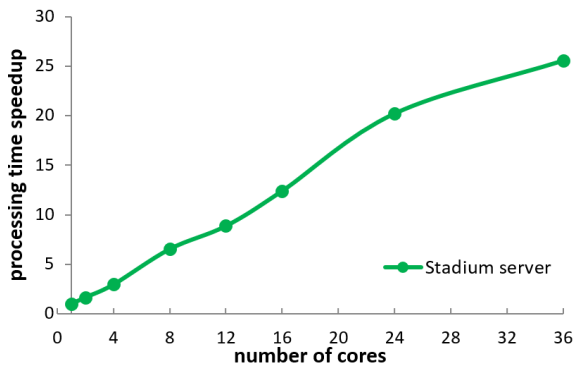

Figure 8: Processing speedup with number of cores

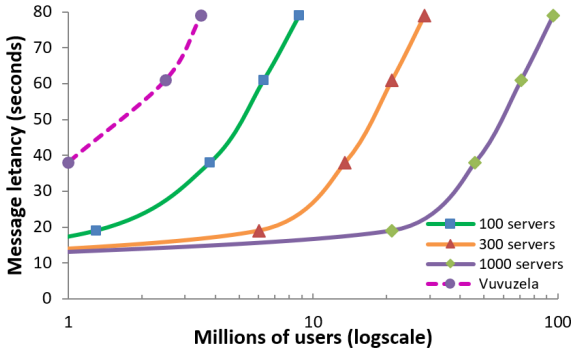

Figure 9: Stadium message latency as a function of number of connected users

servers, each Stadium server sends only at a rate of 20Mbps (about 1.5% of a Vuvuzela server). We argue that this reduction in bandwidth utilization is significant in order to support large scale deployments of Stadium.

Communication is the dominant factor in the cost of maintaining a server. Consider, for example, the price of running a 36-core server on Amazon EC2. The cost of using such server for processing is $1.5/hour. In contrast, the cost of transmitting at 1.3Gbps is approximately $14/hour, almost 10× the cost of processing (Using Amazon's prices as of September 2016). To further illustrate the feasibility of deploying Stadium, consider the top 300 relays in the Tor network, each offers more than 140 Mbps of bandwidth (see [40]). Using only these servers, Stadium could support over 10 million users with reasonable 30 seconds latency (see Figure 9). In contrast, no Tor relay provides more than 1Gbps of bandwidth (as Vuvuzela requires).

11 Related Work

Anonymous communication systems. This paper compares with Vuvuzela [41], which has similar privacy goals based on differential privacy. In contrast to Vuvuzela, Stadium allows for incremental and scalable deployments for efficiently handling millions users and cuts the cost of operating each server (see comparison in Section 10).

Parallel mixnets [27, 17] were suggested as a scalable mixnet [9] design. Stadium adopts ideas from [27], where messages can take multiple paths through the mixnet. However, the design of parallel mixing is such that when some relationships between inputs and outputs are known to the attacker, e.g., when attacker-controlled clients send messages through the system, the attacker gains information about other communicating pairs of users. The problem grows worse when the attacker controls some of the mix servers and can significantly degrade privacy by combining information gained over time [6]. Stadium uses random paths and differential privacy to provably bound information leakage, and its privacy guarantees are independent of other users traffic.

Systems for providing provable privacy include Riposte [12] and Dissent [13, 42], which rely on broadcasting all messages which introduces significant traffic volumes and communication costs, or utilize computationally intensive private information retrieval protocols. As a result, these systems have only scaled to support thousands of users or a few hundred messages per second.

In contrast, Tor [16], the most popular anonymity system today, and cMix [10], a recently suggested efficient mixnet, scale to handle tens of millions of users, but do not protect against powerful adversaries. In particular, user privacy depends on the number of other users in the system and their traffic patterns, and Tor was shown vulnerable to both passive [32] and active [26] attacks. AnonPoP [25] has a scalable mixnet design similar to Vuvuzela extending to allow for multiple conversations and offline clients, but does not provide provable privacy guarantees.

Verifiable shuffles. Stadium uses verifiable shuffles to ensure that no server modified the messages after they were sent to the system. Verifiable shuffles were originally proposed to allow globally verifiable e-voting [1, 3, 8, 14, 23, 34], 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. Other verifiable mixnet approaches such as randomized partial checking [29] allow a small probability that malicious servers will get

away with changing a message. Even though this probability is small, over many servers and many rounds, it becomes too much for our privacy budget.

Differentially-private systems. Several works present privacy-preserving services using differential privacy. PINQ [33] allows to perform set operations for processing a database of private records. Airavat [35] allows Map-Reduce processing of sensitive data. These systems assume that servers are trusted, and keep a 'privacy budget', in order to reason about the information leaked to the external user through past queries. In contrast, Stadium limits information leaked to attacker, who possibly controls some of the servers in the system.

Stadium generates sufficient noise such that communication is private regardless of the number of users introducing baseline communication and processing overhead. As suggested in [2, 5, 24], Stadium can benefit from using legitimate users' traffic as noise.

Another related line of work is user privacy for untrusted aggregators wishing to learn summary statistics over user populations [21, 11]. Similar to our scenario, noise is generated independently (by users vs. by honest servers) and privacy is achieved in aggregate.

12 Conclusion

Stadium is the first private messaging system to protect both data and metadata while scaling effectively across multiple servers. The system scales to support over an order of magnitude more users than prior systems through a novel verifiable parallel mixnet design. Stadium distributes the message load across servers and can be incrementally deployed by organizations with modest resources, thus providing a tangible path to Internet-wide deployment.

Acknowledgments

Thanks to Justin Martinez and Pratheek Nagaraj for helping us implement and evaluate Stadium, and to David Lazar and Jelle van den Hooff for their feedback on the design of Stadium and on this paper. This work was supported by NSF awards CNS-1053143, CNS-1413920, CNS-1414119, NSF GRFP Grant No. 2016206490, and by Google.

References

[1] M. 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] R. Bassily, A. Groce, J. Katz, and A. D. Smith. Coupled-Worlds Privacy: Exploiting Adversarial Uncertainty in Statistical Data Privacy. In *FOCS*, pages 439–448. IEEE Computer Society, 2013.

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

[4] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[5] R. Bhaskar, A. Bhowmick, V. Goyal, S. Laxman, and A. Thakurta. Noiseless Database Privacy. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2011.

[6] N. Borisov. An analysis of parallel mixing with attacker-controlled inputs. In G. Danezis and D. M. M. Jr, editors, *Privacy Enhancing Technologies*, volume 3856 of *Lecture Notes in Computer Science*, pages 12–25. Springer, 2005.

[7] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *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.

[8] C.-C. Chang and W.-B. Wu. A secure voting system on a public network. *Networks*, 29(2):81–87, 1997.

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

[10] D. Chaum, F. Javani, A. Kate, A. Krasnova, J. de Ruiter, and A. T. Sherman. cMix: Anonymization by High-Performance Scalable Mixing. https://eprint.iacr.org/2016/008.pdf, 2016.

[11] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards statistical queries over distributed private user data. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 169–182, 2012.

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

[13] H. Corrigan-Gibbs and B. Ford. Dissent: accountable anonymous group messaging. In *ACM Conference on Computer and Communications Security*, pages 340–350. ACM, 2010.

[14] A. M. Davis, D. Chmelev, and M. R. Clarkson. Civitas: Implementation of a threshold cryptosystem. 2008.

[15] R. Dingledine. Vulnerabilities in Tor: (past,) present, future. The Free Haven Project, 2008.

[16] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320. USENIX, 2004.

[17] R. Dingledine and P. Syverson. Reliable mix cascade networks through reputation. In *International Conference on Financial Cryptography*, pages 253–268. Springer, 2002.

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

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

[20] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[21] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067. ACM, 2014.

[22] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[23] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *Advances in Cryptology–CRYPTO 2001*, pages 368–387. Springer, 2001.

[24] J. Gehrke, M. Hay, E. Lui, and R. Pass. Crowd-blending privacy. In *Advances in Cryptology–CRYPTO 2012*, pages 479–496. Springer, 2012.

[25] N. Gelernter, A. Herzberg, and H. Leibowitz. Two cents for strong anonymity: The anonymous post-office protocol. *Proceedings on Privacy Enhancing Technologies*, 2016(2):1–20, 2016.

[26] Y. Gilad and A. Herzberg. Spying in the dark: TCP and tor traffic analysis. In *Privacy Enhancing Technologies - 12th International Symposium, PETS 2012, Vigo, Spain, July 11-13, 2012. Proceedings*, volume 7384 of *Lecture Notes in Computer Science*, pages 100–119. Springer, 2012.

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

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

[29] M. Jakobsson, A. Juels, and R. L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security Symposium*, pages 339–353. San Francisco, USA, 2002.

[30] J. Katz. Efficient and non-malleable proofs of plaintext knowledge and applications. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2003.

[31] D. Lazar and N. Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *OSDI*. Usenix, 2016.

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

[33] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD Conference*, pages 19–30. ACM, 2009.

[34] C. A. 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.

[35] I. Roy, S. T. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *NSDI*, volume 10, pages 297–312, 2010.

[36] A. Rusbridger. The Snowden Leaks and the Public. The New-York Review of Book, Nov. 2013.

[37] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), Nov. 1979.

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

[39] E. Stefanov and E. Shi. FastPRP: Fast pseudo-random permutations for small domains. *IACR Cryptology ePrint Archive*, 2012:254, 2012. informal publication.

[40] The Tor Project. Tor Metrics: Advertised Relay Bandwidth. https://metrics.torproject.org/advbwdist-perc.html?start=2016-03-15&end=2016-09-15&p=97, May 2016.

[41] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *SOSP*, pages 137–152. ACM, 2015.

[42] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 179–182, 2012.

A Arranging in Mix Chains

In this section we describe how Stadium servers arrange in l-server long mixing chains after they coordinated a random seed s (see details in Section 4.1). In order to ensure that no server needs to handle the message batch from two different chains simultaneously, we select the order such that each server appears exactly once in each position of the chains. To do this, we use a pseudo random permutation (PRP) keyed using the random seed s, that maps the range $1..m$, where m denotes the number of servers (e.g., constructed using [39]). We invoke it $l \cdot m$ and arrange the outputs in an $l \times m$ matrix M, such that $M_{i,j} = PRP_{s||i}(j)$:

$$\begin{bmatrix} \text{PRP}_{s||1}(1) & \text{PRP}_{s||1}(2) & \dots & \text{PRP}_{s||1}(m) \\ \text{PRP}_{s||2}(1) & \text{PRP}_{s||2}(2) & \dots & \text{PRP}_{s||2}(m) \\ & \dots\dots\dots\dots\dots & & \\ \text{PRP}_{s||l}(1) & \text{PRP}_{s||l}(2) & \dots & \text{PRP}_{s||l}(m) \end{bmatrix}$$

We use each column to identify the servers in a mix chain and their order (there are m mix chains). Since each row is generated by the same keyed permutation, no server can appear twice at the same position in the chain, and therefore no server will need to process two message batches simultaneously.

B Diff-Privacy with Poisson Noise

In this section we prove Theorem 1. To simplify presentation we use the $\text{Pois}(\lambda; k)$ to denote $\Pr[x = k | x \sim \text{Pois}(\lambda)]$. We show that $\text{Pois}(\lambda)$ noise provides differential privacy with $\epsilon = \ln(1 + \frac{c\sqrt{\lambda}+1}{\lambda})$ and $\delta = \text{Pois}(\lambda; \lambda - c\sqrt{\lambda}) + \text{Pois}(\lambda; \lambda + c\sqrt{\lambda})$. Where c is a positive value that allows to trade-off higher ϵ for lower δ. Our proof combines two lemmas. First, we show that the ratio between $\text{Pois}(\lambda; k)$ and $\text{Pois}(\lambda; k+1)$ for values of k that are close to λ is bounded by e^{ϵ}. Second, we show that for values of k that are more distant from λ, the difference between the two probabilities is no more than δ.

Lemma 5. *If $|\lambda - k| \leq c\sqrt{\lambda}$, then $Pois(\lambda; k) \leq e^{\epsilon} Pois(\lambda; k+1)$.*

Proof. Using the Poisson probability mass function, it follows that we require: $\frac{\lambda^k e^{-\lambda}}{k!} \leq e^{\epsilon} \frac{\lambda^{k+1} e^{-\lambda}}{(k+1)!}$. Hence:

$$1 \leq e^{\epsilon} \frac{\lambda}{k+1} \rightarrow \epsilon \geq \ln(\frac{k+1}{\lambda}) \tag{1}$$

Since $k \leq \lambda + c\sqrt{\lambda}$ and $\ln$ is monotonously increasing, we find that it is sufficient to select: $\epsilon \geq \ln(\frac{\lambda + c\sqrt{\lambda}+1}{\lambda}) = \ln(1 + \frac{c\sqrt{\lambda}+1}{\lambda})$. □

For the second part of the proof, which is composed of Lemmas 6 and 7, we will use the following property: $\text{Pois}(\lambda; k)$ is increasing for $k < \lambda$ and decreasing for $k > \lambda$.

$$\text{Pois}(\lambda; k+1) - \text{Pois}(\lambda; k) = \frac{\lambda^{k+1} e^{-\lambda}}{(k+1)!} - \frac{\lambda^k e^{-\lambda}}{k!} =$$
$$\frac{\lambda^k e^{-\lambda}}{k!}(\frac{\lambda}{k+1} - 1) = Pois(\lambda; k)(\frac{\lambda}{k+1} - 1) \tag{2}$$

For any $k, \text{Pois}(\lambda; k) \geq 0$. Therefore the above difference is non negative (i.e., function is not decreasing) when $\frac{\lambda}{k+1} - 1 \geq 0 \rightarrow k \leq \lambda - 1$, and non-positive when $k \geq \lambda - 1$.

We next show that for any subset $\mathcal{S} \subseteq \{k | |\lambda - k| > c\sqrt{\lambda}\}$ of the range of $\text{Pois}(\lambda)$ it holds that $|\Pr[k \in \mathcal{S}] - \Pr[k+1 \in \mathcal{S}]| < \delta$. We divide the proof into two lemmas.

Lemma 6. *Let $\mathcal{S}_1 = \{k | k < \lambda - c\sqrt{\lambda}\}$. We show that $|Pr[k \in \mathcal{S}_1] - Pr[k+1 \in \mathcal{S}_1]| < \delta_1$.*

Proof. In this range it holds that $\text{Pois}(\lambda; k+1) > \text{Pois}(\lambda; k)$ (the Poisson pmf is increasing). Thus, $|\Pr[k \in \mathcal{S}_1] - \Pr[k+1 \in \mathcal{S}_1]| = \Pr[k+1 \in \mathcal{S}_1] - \Pr[k \in \mathcal{S}_1] \leq \sum_{0 \leq i < \lambda - c\sqrt{\lambda}} \text{Pois}(\lambda; i+1) - \text{Pois}(\lambda; i)$. The reason is that for every $k \in S_1$ the difference $\Pr[k+1 \in \mathcal{S}_{\infty}] - \Pr[k \in \mathcal{S}_{\infty}]$ is positive. So the term is upper bounded by summing for all $k \in S_1$. This is a telescopic series and its sum is $\text{Pois}(\lambda; \lambda - c\sqrt{\lambda} + 1) - \text{Pois}(\lambda; 0) < \text{Pois}(\lambda; \lambda - c\sqrt{\lambda}) = \delta_1$. □

Lemma 7. *Let $\mathcal{S}_2 = \{k | k > \lambda - c\sqrt{\lambda}\}$. We show that $|Pr[k \in \mathcal{S}_2] - Pr[k+1 \in \mathcal{S}_2]| < \delta_2$.*

Proof. $\text{Pois}(\lambda; k+1) < \text{Pois}(\lambda; k)$ (the Poisson pmf is decreasing), thus we are interested in bounding $\sum_{i > \lambda + c\sqrt{\lambda}} \text{Pois}(\lambda; i) - \text{Pois}(\lambda; i+1)$. This is a telescopic series, and its sum less than $\text{Pois}(\lambda; \lambda + c\sqrt{\lambda}) = \delta_2$. □

Combining Lemmas 6 and 7 we find that since $\mathcal{S}_{\infty} \bigcup \mathcal{S}_{\in} = \mathcal{S}$, we get: $\delta = \delta_1 + \delta_2 = \text{Pois}(\lambda; \lambda - c\sqrt{\lambda}) + \text{Pois}(\lambda; \lambda + c\sqrt{\lambda})$

C Probabilistic Differential Privacy

In this section we prove Theorem 2.

Proof. For any subset S of the image of mechanism M, the following holds:

$$\Pr[\mathrm{M}(x) \in \mathrm{S}] =$$
$$\sum_i p_i \cdot \Pr[\mathrm{M}(x) \in \mathrm{S} | \text{case} = i] \leq$$
$$\sum_i p_i (e^{\epsilon_i} \Pr[\mathrm{M}(x) \notin \mathrm{S}] + \delta_i) =$$
$$e^{\ln \sum_i p_i \cdot e^{\epsilon_i}} \Pr[\mathrm{M}(x) \notin \mathrm{S}] + \sum_i p_i \delta_i$$

Therefore, $\epsilon = \ln \sum_i p_i \cdot e^{\epsilon_i}$ and $\delta = \sum_i p_i \delta_i$ □

D Affected Observable Variables

We prove Theorem 3, which bounds by the number of observable variables that are affected by a user's communication patterns at two single-access and two double-access variables.

Proof. Consider a user Alice. In each round Alice may communicate with one of her peers, say Bob, or may not communicate at all. Her cover stories afforded by Stadium are, that instead of Alice's real actions, she is either communicating with some random (non-compromised) idle user or not communicating at all (in case that Alice was actually talking with Bob). We analyze how each of the possible pairs of real/cover actions affect Stadium's observable variables and show that Theorem 3 holds in each case.

Case 1: Alice communicates with Bob. If Alice claims to be idle, then she claims to have used a different, random, dead-drop ID as destination for her message (instead of the one coordinated with Bob). By this cover story the double-access variable matching her and Bob's output chains is decreased by 1, and the single access variables of both output chains increase by 1. (i.e., 1 double- and 2 single- access variables change.)

If Alice instead claims to communicate with another user, X, then the double-access variable matching Alice and Bob's output chains is decrease by 1, the double-access variable matching Alice and user X's output chains is increased by 1. Bob's output chain single-access variable is increased by 1 and X's output chain single access variable is decreased by 1. (i.e., 2 double- and 2 single- access variables change.)

Case 2: Alice is idle. If Alice claims to be communicating with user X, then the single-access variables for Alice and X's output chains are decreased by 1, and the double-access variable matching their output chains is increased by 1. (i.e., 1 double- and 2 single- access variables change.) □