Stadium: A Distributed Metadata-Private Messaging System

Nirvan Tyagi
Cornell University

Yossi Gilad
Boston University
MIT CSAIL

Derek Leung
MIT CSAIL

Matei Zaharia
Stanford University

Nickolai Zeldovich
MIT CSAIL

ABSTRACT

Private communication over the Internet remains 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 anonymity systems, such as Tor, are susceptible to traffic analysis attacks that leak metadata. In contrast, the largest-scale systems with metadata privacy require passing all messages through a small number of providers, requiring a high operational cost for each provider and limiting their deployability in practice.

This paper presents Stadium, a point-to-point messaging system that provides metadata and data privacy while scaling its work efficiently across hundreds of *low-cost* providers operated by different organizations. Much like Vuvuzela, the current largest-scale metadata-private system, Stadium achieves its provable guarantees through differential privacy and the addition of noisy cover traffic. The key challenge in Stadium is limiting the information revealed from the many observable traffic links of a highly distributed system, without requiring an overwhelming amount of noise. To solve this challenge, Stadium introduces techniques for distributed noise generation and differentially private routing as well as a *verifiable parallel mixnet* design where the servers collaboratively check that others follow the protocol. We show that Stadium can scale to support 4× more users than Vuvuzela using servers that cost an order of magnitude less to operate than Vuvuzela nodes.

CCS CONCEPTS

• **Security and privacy → Pseudonymity, anonymity and untraceability**; **Privacy-preserving protocols**; **Distributed systems security**;

SOSP '17, October 28, 2017, Shanghai, China

KEYWORDS

anonymous communication, differential privacy, mixnet, verifiable shuffle

1 INTRODUCTION

The continued prominence of anonymous whistleblowing and private communication in world affairs means that these issues, and the systems that enable them, have become an integral part of society. As a result, there is substantial interest in systems that offer strong privacy guarantees—often against a *global* adversary with the ability to monitor and inject network traffic, such as an ISP or nation-state.

Services such as WhatsApp, Signal, and Telegram have deployed end-to-end encryption solutions to popular reception from the public. Even so, encryption is only the first step in protecting users' communications. Although encryption hides communication *content*, it does not hide *metadata* because adversaries can still learn who is communicating with whom and at what times. Metadata reveals a great deal of information; in fact, NSA officials have stated that "if you have enough metadata, you don't really need content" [39].

In recent years, researchers have made significant progress in designing systems with provable protection against metadata leakage [1, 12, 14, 15, 44]. Despite the variation in solutions, these past systems share a similar deployment strategy and trust scheme that limits their scalability (with a few notable exceptions [1, 32], see §11). Specifically, in most of these systems, a small set of *providers* carries out the anonymity protocol, and users must trust that at least *one* provider in the set is honest (i.e. "the anytrust assumption").

This strategy limits the number of users these systems can support in two main ways. First, because each individual provider is responsible for ensuring global privacy, each provider's workload increases proportionally with the total number of users of the system (e.g., mixing messages sent by all users). For example, Vuvuzela [44] reported exchanging 68,000 messages per second with 3 providers at a bandwidth cost of 1.3 Gbps per provider, which would cost each provider around $1,000 per month [36]. Second, users must trust one

of the system's few participating providers. As the number of users grows, it becomes hard to find a small set of organizations for which the anytrust assumption holds for all users. Relaxing this assumption by adding more providers quickly degrades performance by raising costs that are at least linear in the number of providers [14, 44]. In summary, these systems are restricted by a combination of a small number of providers and a large operating cost per provider. We refer to these systems as scaling *vertically*, in that providers must run more infrastructure to support more users.

In contrast, the Tor network [18] is an example of an anonymity system that scales *horizontally*. Tor consists of thousands of commodity servers run by volunteer providers. Instead of trusting any single provider, users trust that some fraction of participating providers are honest. The total load of all users on the system is distributed evenly across the providers. Thus, the system scales to support more users with the addition of new providers. However, Tor is vulnerable to traffic analysis and injection attacks that can deanonymize users and leak metadata [2, 7, 21, 30, 45].

This paper presents Stadium, a point-to-point messaging system that uses a *horizontally scaling* deployment strategy and provides *provable privacy* guarantees. We show that Stadium scales to support up to 4× more users than previous systems with provable guarantees, using servers whose operating costs are an order of magnitude smaller than the equivalent Vuvuzela deployment. For example, at a latency of 2 minutes, we find Stadium can support more than 20 million simultaneous users with 100 providers, while Vuvuzela [44], the current largest-scale metadata-private system, is limited to 5 million. Additionally, Stadium's capacity scales near-linearly with the number of participating providers, i.e., adding a provider additively increases the total number of users that can be supported.

Like Vuvuzela, Stadium uses differential privacy to bound the metadata leakage of the system over time. Informally, differential privacy can be thought of as "plausible deniability." By observing the system, an adversary may gain some small amount of statistical information in learning the communication patterns of users (e.g. is Alice talking to Bob?). Differential privacy bounds the information the adversary can gain from these observations: the adversary's observations are almost equally likely to be caused by Alice talking to Charlie, or Alice not talking to anyone. To provide differential privacy, Stadium adds noise to the adversary's observations in the form of decoy message traffic.

Stadium mixes user messages and noise messages together using a *mixnet* architecture, which takes as input a set of messages and outputs a random permutation of those messages [8]. Furthermore, Stadium requires its mixnet to scale horizontally. Messages must be distributed among providers and mixed in *parallel* so that each provider only handles a fraction of messages. Secure and efficient parallel mixing is challenging due to the observable links between distributed mixing servers; advanced traffic analysis techniques can be used to trace messages [6].

We provide a differential privacy analysis of Stadium's mixnet routing to bound the information leaked through traffic analysis. To meet the requirements of the differential privacy model, we augment Stadium's parallel mixnet with a number of novel verifiable processing techniques, allowing providers to verify that other providers are correctly relaying and mixing messages. Providers are organized into small groups called *mixchains*, and messages are distributed among mixchains and mixed in parallel. The mixing and verification workload of a provider is confined to the fraction of messages that flow through its mixchain. As long as there exists a single honest provider in each mixchain, Stadium achieves *global* verification and mixing.

Stadium's scalability does come with one significant caveat. In order to achieve comparable privacy to previous systems, Stadium requires both more noise and a higher computation cost due to verifiable processing, which often translates into a higher latency (several minutes vs. 30–60 seconds). Nonetheless, Stadium's total *bandwidth* cost, which constitutes the dominant monetary cost for providers, is within a factor of 1 to 3 of Vuvuzela, and Stadium can *divide* this cost across hundreds of small providers. Therefore, we believe that Stadium is a compelling new design point for metadata-private messaging for two reasons: it can be deployed incrementally using smaller providers, and it does not require a single centralized anytrust set as in previous systems.

To summarize, our contributions are:

- The design for a horizontally scaling metadata-private messaging system that can distribute work efficiently across hundreds of servers.
- A novel *verifiable parallel mixnet* design that allows participating providers to efficiently verify the correctness of the mix, including techniques such as hybrid verifiable shuffling, verifiable distribution, and intra-mixchain verification.
- A privacy analysis of Stadium, including techniques for designing distributed, differentially private systems, such as the Poisson noise mechanism, routing network analysis, and collaborative noise generation across mutually untrusted servers.
- An implementation and experimental evaluation of Stadium over a 100 server deployment on Amazon EC2.

2 GOALS

In this section we present Stadium's key design goals: providing private point-to-point messaging in the 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 providers' servers. Adversary-controlled servers and users may deviate in any way from Stadium's protocol. We allow the adversary to monitor, block, delay, or inject traffic on any network link at all communication rounds. Servers are arranged in mixchains, where each chain consists of servers that belong to different providers which are unlikely to collude; we assume that at least one server in every chain follows the protocol.

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 users hold a shared key. One recent scalable system, Alpenhorn [34], provides a service that allows two users to privately coordinate a shared secret.

2.2 Privacy

Stadium has a similar privacy goal to Vuvuzela [44]. It aims to prevent adversaries from distinguishing between communication patterns of its users, even if the users exchange many messages. We use the following definition from the differential privacy literature to analyze Stadium's privacy guarantees [20]:

Definition 2.1. A randomized mechanism M is (ϵ, δ)-differentially private if for any two adjacent inputs D and D' and for all sets of outputs S, $\Pr[M(D) \in S] \leq e^{\epsilon} \cdot \Pr[M(D') \in S] + \delta$.

The inputs to the Stadium mechanism (D and D') are the set of 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 is not talking to anyone). Real or not, the differential privacy definition requires that all these stories appear almost as plausible. In the above definition, ϵ and δ bound the information that an adversary might learn about Alice's communication in each round. We design Stadium to provide differential privacy such that ϵ and δ are small enough to keep all other cover stories plausible even after Alice sends hundreds of thousands of messages.

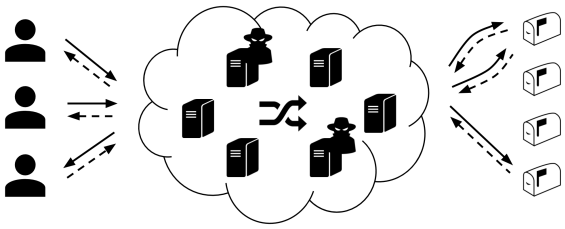


Figure 1: Stadium overview. Users send messages for a communication round. Stadium's servers work together to shuffle the messages, verifying each other for correctness. Shuffled messages are exchanged at dead-drops and reversed through the system to the recipient user.

2.3 Scalability

Our scalability goal is to support tens of millions of simultaneous users on Stadium, which is comparable to the number of Tor users. The scalability goal necessitates that the user message load is distributed across servers and represents a departure from previous metadata-private systems.

3 COMMUNICATION OVERVIEW

Communication through Stadium takes place in rounds. Every fixed interval, a new round begins to process a set of messages Stadium accumulated from its users. To provide differential privacy of user communication patterns, Stadium servers generate noise messages (i.e., cover traffic) which are also input to the system when the round begins. All messages are shuffled by Stadium's distributed servers to unlink them from their sender. Stadium borrows a *dead-drop* communication strategy from Vuvuzela [44] 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. Message destination dead-drops 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 contents. 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 observing a

user's communication line does not leak whether they communicate, users send exactly one message every round, and thus, receive exactly one message every round. If the user does not communicate, a dummy message is sent which will route back to the sender. An overview of Stadium's design is illustrated in Figure 1. In order to start communication, Alice and Bob use a separate *dialing* protocol [34], assumed as a precursor to our system (§2.1), which allows them to coordinate a dead-drop for every round as well as a symmetric key to encrypt the content of their communication.

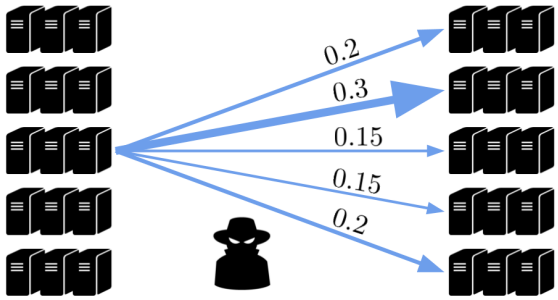


Figure 2: Adversary gains probabilistic information about Alice's message path by observing an uneven distribution of messages out of a server.

4 PARALLEL MIXING CHALLENGES

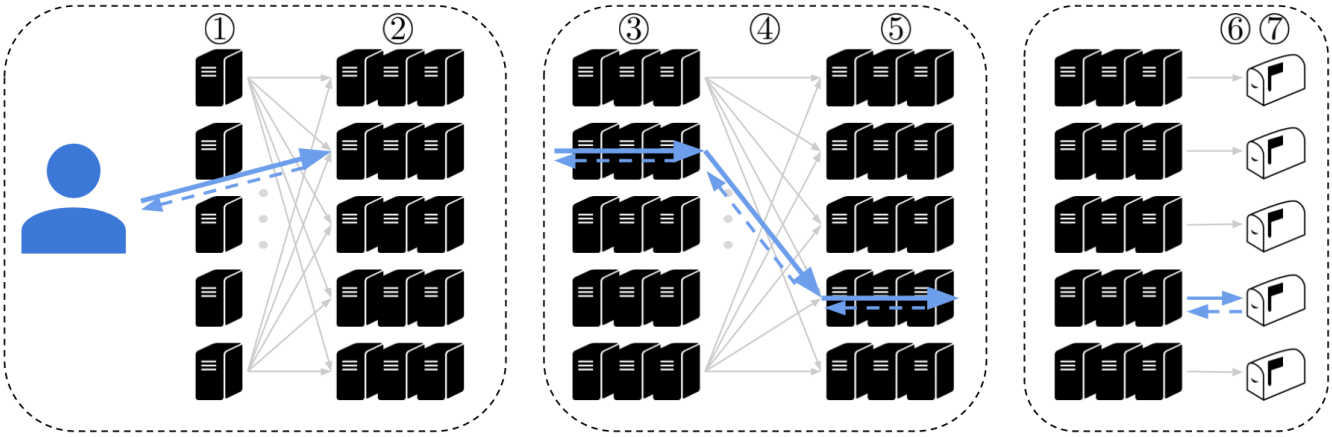
A key challenge to Stadium's design is in mixing messages securely and efficiently. Traditional sequential mixnets, where each server mixes all messages, fail to perform at Stadium's targeted scale. Instead, Stadium requires a parallel mixing scheme, where each server processes only a fraction of the input messages, yet the messages are still mixed globally. To achieve this, most previous parallel mixing schemes [17, 19, 26, 38] use the same general pattern. Each server accepts a small fraction of messages, mixes those messages, then splits the messages among other servers. This process is repeated for some number of cycles. Although at the beginning, messages are clearly partitioned by their starting server, they are mixed globally over many cycles of splitting and mixing. The primary difference between previous parallel mixing schemes is in the chosen *mixing topology*. The mixing topology specifies the links between servers at each depth, i.e., where servers collect messages from and where they distribute messages to after shuffling. Different mixing topologies trade off between depth (number of mixing servers a message passes through), switch size (number of incoming and outgoing connections for each server), and, as we discuss next, quality of mix.

Resistance to traffic analysis. It has been shown that most previous parallel mixing schemes are susceptible to advanced traffic analysis attacks [6]. These attacks use uneven distributions between servers to probabilistically model likely paths for messages, depicted in Figure 2. Even if distributions are forced to be uniform, e.g. by padding with dummy messages, uneven distributions can be inferred if the adversary knows the input-output relation of some messages. Essentially, these attacks are able to trace back messages with some significant probability, even after many cycles of mixing.

There are a few exceptions resistant to traffic analysis; notably, the iterated butterfly topology [16] and the square topology [28] both have theoretical results for achieving near-random permutations. However these results come at the expense of relatively large depth, which is not suitable to Stadium's low latency messaging application. Instead, Stadium opts for a shallow, but vulnerable, topology. Information leakage in routing is identified and quantified with a differential privacy analysis (§7,8). Note that noise messages in Stadium serve two purposes: (1) protecting dead-drop access patterns, and (2) protecting routing traffic patterns.

Deciding how to inject noise messages into the system poses another challenge. If each server, in turn, adds and shuffles in the total required amount of noise messages (as in Vuvuzela [44]), the total processing cost of the system grows quadratically in the depth of the network; each server must process the messages added by all servers before it. Although Stadium aims for a shallow topology, the network is still too deep to accommodate the quadratic processing costs (~10-25 compared to Vuvuzela's depth 3). Instead, Stadium servers collaborate to inject noise messages across the system in one large step prior to mixing. With this strategy, servers are not mixing in their own noise messages, leaving the opportunity for malicious servers to discard noise messages before they are mixed with user messages. To ensure that noise stays in the system and the differential privacy guarantees are valid, Stadium uses several cryptographic techniques for verifiable processing of messages (§6) allowing honest servers to verify the actions of others. To optimize the relatively expensive verification workload put on honest servers, Stadium introduces a hybrid verifiable shuffling scheme to bootstrap verifiable processing of large messages onto verifiable processing of small authentication keys (§5.1).

In summary, Stadium reasons about the information leakage of parallel mixing through a differential privacy analysis. Verifiable processing is used to ensure noise messages are properly mixed with user messages, and thus, the differential privacy guarantees are upheld.



Algorithm 1: Stadium Communication Protocol

1. **Noise generation**: Servers generate noise messages. (Algorithm 4, §7)
2. **Message input**: Servers and users input messages to system. (Algorithm 5, §6.1)
3. **Mixing I**: Messages are mixed within input chain. (Algorithm 6, §6.2)
4. **Distribution**: Messages are distributed across output chains (Algorithm 7, §6.3)
5. **Mixing II**: Messages are mixed within output chain with messages from all input chains. (Algorithm 6, §6.2)
6. **Dead-drop exchange**: Messages are exchanged at dead-drops of output chain. (Algorithm 8)
7. **Message return**: Messages are reversed through mixnet to users, denoted by dashed arrows. (Algorithm 8)

Figure 3: Stadium's communication protocol through 7 processing phases.

5 DESIGN

This section presents the Stadium protocol in a chronological view of a single communication round. Figure 3 breaks down a round into 7 processing phases.

Collecting messages. In phases 1-2, messages are created and submitted to the system. The message encapsulation protocol is described in the next section (§5.1). Users each submit one message, possibly a dummy message if not communicating; servers submit a randomized number of noise messages. Stadium servers collect messages and use verifiable processing to ensure only "proper" messages are accepted. The verifiable processing of message input (§6.1) verifies submitted messages were created by the submitter (i.e., preventing replay/malleability attacks) and verifies all server-generated noise messages are included.

Mixing messages. Phases 3-5 make up Stadium's parallel mixnet. Stadium employs a 2-layer mixing topology consisting of an input mixing layer (phase 3), all-to-all distribution (phase 4), and an output mixing layer (phase 5). Since Stadium assumes a fraction of its servers to be adversary-controlled, the input and output mixing layers are not composed of single servers, but rather of small groups of servers organized into *mixchains*. Verifiable processing of the mixnet (mixing §6.2 and distribution §6.3) allows an honest server to verify messages that pass through its mixchain are uncorrupted and properly mixed. Thus, proper execution of the mixnet relies on the assumption that every mixchain contains at least one honest server; we revisit the plausibility of this assumption in §10.3.

A message's path through the mixnet consists of an *input mixchain* and an *output mixchain*. 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 among output chains, where they are again mixed—this time with messages arriving across all input chains.

Exchanging messages. The messages of an output chain are exchanged (phase 6) at the dead-drops hosted by that same output chain as described in §3. Note that this means, two communicating users must route their messages to the same output chain in order for them to be exchanged - a restriction imposed by our privacy analysis. Messages then

travel back through the mixnet to their recipient users, each mixchain reversing the permutation applied in the forward direction (phase 7). The last two phases (6-7) are not verified; messages output from the mixnet are already anonymized. Adversaries can deny service to users by not exchanging or corrupting messages, but the user's intended communication pattern remains hidden.

5.1 Message Encapsulation

Users set the path of their message through Stadium by designating an input chain, an output chain, and a dead-drop. Each mixchain has a public key, jointly established by the servers included in that chain (see Algorithm 2). Users encapsulate their message using the public keys of their selected path, using Algorithm 3. Encapsulated messages contain both the underlying message content that Alice wishes to exchange, as well as message routing metadata (e.g., the output chain selection and dead-drop id). Routing metadata needs to be properly protected for two main reasons. (1) Revealing user message routing metadata can expose user communication patterns. (2) Routing noise messages along their intended paths is necessary for the differential privacy guarantees to hold. Stadium uses verifiable processing to protect the integrity of routing metadata and uses threshold decryption to ensure routing metadata is revealed only when the mixing protocol is honestly followed.

Hybrid verifiable shuffling. Providing efficient verifiable processing for routing metadata remains a challenge. In particular, zero-knowledge shuffle proofs work on group elements (i.e., public-key encryption). Zero-knowledge shuffle proofs allow mixing servers to prove the output messages are a permutation of the input messages without revealing any knowledge about the permutation itself (preventing message corruption). Onion encrypting the metadata over multiple chains requires working in a large group size, making computation inefficient. This holds true even for the 2-layer onion encryption needed for Stadium's shallow mixing topology. Instead, Stadium handles messages in two pieces, *authentication* and *content*. The authentication piece is small (32 bytes) and is verifiably processed. It provides authentication information required to verify the integrity of the content piece.

Hybrid verifiable shuffling bootstraps verifiable shuffling of large messages onto zero-knowledge shuffle proofs of small authentication keys (extended from [33]). To form the content piece, we append a message authentication code (Encrypt-then-MAC) to the message content and then onion encrypt with the mixchain servers' public keys. To form the authentication piece, the authentication key used to produce the MAC is encrypted with the mixchain's public key. The authentication pieces are permuted by each server verifiably using zero-knowledge proofs. At the same time, each server applies the same permutation to the content pieces, stripping a layer of the hybrid encryption at each step. At the end of the mixchain, the authentication key is revealed and used to verify the correct permutations were applied to the content by checking the MAC. The message encapsulation is described in Algorithm 3 and the shuffling protocol is described in Algorithm 6. Although Stadium only uses the technique for 2 mixing layers, it extends efficiently to more.

In Stadium, the purpose of hybrid verifiable shuffling is to verifiably hold the routing metadata (both output chain selection and dead-drop id) within the content piece. Even though the integrity of the underlying communication message is also protected through hybrid verifiable shuffling, this is a non-goal as it is still susceptible to corruption during message exchange and return.

6 VERIFIABLE PROCESSING

Verifiable processing serves two main purposes in Stadium. The first is to prevent adversaries from directly learning user communication metadata, e.g. through revealing output chain selection or dead-drop id before properly mixing. The second is to prevent adversaries from partially learning user communication metadata through system-wide traffic analysis. This is accomplished by ensuring noise messages are sent along their intended routes, upholding the guarantees given by the differential privacy analysis.

In this section, we describe the three components that make up the verifiable processing pipeline (phases 2-5 in Figure 3): (1) Message input – tracking messages that enter the system, (2) Mixing – tracking messages as they are passed through a mixchain, and (3) Distribution – tracking messages as they are passed between mixchains. For each component, we list a set of properties that it aims to verify. We argue that the component successfully verifies the intended properties, and if a breach is detected, it is handled safely without exposure of user communication metadata. Complete security arguments can be found in Appendix A.

Intra-mixchain verification. The message processing and verification workloads of a server are restricted to the scope of its mixchain (with the exception of a few inexpensive operations in the verifiable distribution phase). Thus, the workload of a server scales with the fraction of messages that flow through its mixchain. Designing mixchains to operate as self-verifying entities is a crucial component to Stadium's horizontal scalability. For mixchains to self-verify themselves, each mixchain must contain at least one honest server. We evaluate the plausibility of such a configuration in §10.3.

Algorithm 2: Key Generation

1. **Communication partners share symmetric key**: Alice and Bob hold shared symmetric key. $k_{AB} \leftarrow\$ \text{KGen}(1^n)$
2. **Servers' public keys**: Each server s_i generates a public, private key pair. $(pk_{s_i}, sk_{s_i}) \leftarrow\$ \text{KGen}(1^n)$ of the form (g^x, x) for ElGamal encryption.
3. **Mixchains' public keys**: Each mixchain, $mc_i = \{s_1^i, \ldots, s_\ell^i\}$, has public, private key pair derived from the member servers' key pairs. The private key is held in shares for threshold decryption. $(pk_{mc_i}, sk_{mc_i}) \leftarrow (\prod_{s \in mc_i} pk_s, \sum_{s \in mc_i} sk_s)$

Algorithm 3: Message Encapsulation

$(auth_{input}, c_{input}) = \text{Encapsulate}(i, x, d, m)$

1. **Select message path**: A message path is designated by input chain i, output chain x, and dead-drop id d. For Alice talking to Bob, $m = \text{Enc}(k_{AB}, \text{plaintext})$.
2. **Encrypt message for output chain**: Encrypt the dead-drop id and message. $(auth_{output}, c_{output}) = \text{HybridShuffleEnc}(mc_x, d \parallel m)$
3. **Encrypt message for input chain**: Append output chain content and auth with output chain selection and encrypt. $(auth_{input}, c_{input}) = \text{HybridShuffleEnc}(mc_i, x \parallel auth_{output} \parallel c_{output})$

$(auth, c) = \text{HybridShuffleEnc}(mc, m)$

1. **Encrypt authentication piece**: Generate random group element, $a \leftarrow \text{Gen}()$. Hash group element (with round nonce) to acquire authentication key, $k_{auth} = H(a \| nonce_{round})$. Encrypt the group element with mixchain's public key, $auth \leftarrow \text{Enc}(pk_{mc}, a)$.

2a. **Add MAC to message**: Generate symmetric key, $k_m \leftarrow \text{KGen}(1^n)$. Hybrid encrypt the symmetric key with the mixchain's public key, $c \leftarrow \text{Enc}(k_m, m) \parallel \text{Enc}(pk_{mc}, k_m)$. Add MAC to the encrypted message, $c \leftarrow c \parallel \text{MAC}(c)$.

2b. **Onion encrypt content piece**: For every server $s \in mc.reverse()$: Generate symmetric key $k_s \leftarrow \text{KGen}(1^n)$, and onion encrypt $c \leftarrow \text{Enc}(pk_s, k_s) \parallel \text{Enc}(k_s, c)$. (For communicating partners, $k_s \leftarrow \text{KGen}(1^n, k_{AB})$)

Algorithm 5: Message Input

For submitting messages to mixchain i:

1. **Encapsulate message**: $(auth_{input}, c_{input}) = \text{Encapsulate}(i, x, d, m)$
2. **Create proof of knowledge**: $proof_{know} = \text{NIZK}_{know}(auth_{input}, a)$, where a is group element used to create authentication key from Algorithm 3.
3. **Submit to input chain**: Send $(auth_{input}, c_{input})$ to first server in mc_i. Send $proof_{know}$ to every server $s \in mc_i$. (For servers submitting noise messages): Group all noise messages submitted to mc_i, $\mathcal{N} = (m^{(1)} \parallel \ldots \parallel m^{(\mathcal{A}_*^i + \mathcal{B}_*^{i,*})})$. Send hash of auths to every server $s \in mc_i$: $H(\mathcal{N}_{auth})$.

For input chains:

1. **Collect candidate batch**: First server in chain collects batch and orders with noise from servers $1 \ldots m$ first, then users, $B = [\mathcal{N}^{(1)} \parallel \ldots \parallel \mathcal{N}^{(m)} \parallel m^{(u1)} \parallel \ldots \parallel m^{(un)}]$

2a. **Verify proofs of knowledge**: Every server in mixchain verifies received proofs. Removes messages with duplicate proofs. If user proof fails, remove. If noise proof fails, halt. $\forall\ auth^{(i)} \in B_{auth} : \text{Vf}_{know}(auth^{(i)}, proof_{know}^{(i)})$.

2b. **Verify noise messages included**: Every server in mixchain verifies with received hashes. $\forall\ \mathcal{N}_{auth}^{(i)} \in B_{auth}$: check $H(\mathcal{N}_{auth}^{(i)})$ against received hash.

Algorithm 4: Noise Generation

1. **Generate single access noise**: $\forall$ combinations of input chain i and output chain x, sample $\mathcal{A}_x^i \sim \text{Pois}(\lambda_1)$. Generate $\mathcal{A}_x^i$ messages sampling a random dead-drop id, d_r, for each one: $\text{Encapsulate}(i, x, d_r, 0)$.
2. **Generate double access noise**: $\forall$ combinations of input chain pairs i, j and output chain x, sample $\mathcal{B}_x^{i,j} \sim \text{Pois}(\lambda_2)$. Generate $\mathcal{B}_x^{i,j}$ pairs of messages sampling a random dead-drop id, d_r, for each pair: $\text{Encapsulate}(i, x, d_r, 0)$, $\text{Encapsulate}(j, x, d_r, 0)$.

Algorithm 6: Hybrid Verifiable Shuffle

$\pi(m^{(1)}, \ldots, m^{(n)}) = \text{HybridShuffle}((auth, c)^{(1)}, \ldots, (auth, c)^{(n)})$

1. **Permute messages**: Each server, in turn, permutes the message batch, B, and passes the permuted to batch to the next server in the mixchain.

1a. **Permute authentication pieces with proof**: Re-randomize every message using ElGamal malleability, multiplying by encryption of 1, $(g^r, g^{r \cdot pk_{mc}})$, $B_{auth}^r \leftarrow B_{auth}$. Sample random permutation π and permute B_{auth}^r. Pass $B' = \pi(B_{auth}^r)$ to next server. Send proof of permutation to every server in mixchain, $proof_{shuf} = \text{NIZK}_{shuf}(B_{auth}, B'_{auth}, \pi)$.

1b. **Permute content pieces**: For each $c = (c[0], c[1]) \in B_c$, denoting parsing the concatenation from Algorithm 3, peel off onion layer to create $c^r \in B_c^r$: $k_s \leftarrow \text{Dec}(sk_s, c[0])$, $c^r \leftarrow \text{Dec}(k_s, c[1])$. Pass $B' = \pi(B_c^r)$ to next server.

2. **Verify authentication shuffle proofs**: Each server in mixchain verifies the permutation of every other server $s \in mc$, $\text{Vf}_{shuf}(B_{auth}, B'^{(s)}_{auth}, proof^{(s)}_{shuf})$.
3. **Threshold decrypt authentication keys**: After all servers mix and all shuffle proofs verify, servers verifiably generate shares and threshold decrypt authentication keys, $[a^{(i)}] = \text{VfThreshDec}(B'_{auth})$, $[k_{auth}^{(i)}] \leftarrow [H(a^{(i)})]$.
4. **Verify content with authentication keys**: Check MAC for every $c^{(i)} \in B'_c$ using $k_{auth}^{(i)}$.

4a. **Trace message back if MAC fails**: Each server s, in reverse order, reveal the relation for failed message in π_s, provides verifiable decryption of k_s used for onion encryption. If all proofs succeed, remove message. Else, identify erring server and halt.

4b. **Threshold decrypt content**: Once all MACs verify or failed MACs are removed, servers verifiably threshold decrypt content, $[k_m^{(i)}] = \text{VfThreshDec}(B'_c[0])$, $[m^{(i)}] \leftarrow \text{Dec}(k_m, B'_c[1])$.

Algorithm 7: Verifiable Distribution

For input chain:

1. **Group messages by output chain selection**: $\forall$ output chain $i \in 1 \ldots m$, each server in input chain groups messages destined for mc_i deterministically to form $b_{out}^{(i)}$. Last server in input chain sends $b_{out}^{(i)}$ to first server of mc_i.
2. **Send hashes to output chains**: Each server in input chain sends $H(b_{auth}^{(i)})$ to every server in mc_i $\forall$ output chain $i \in 1 \ldots m$.

For output chains:

1. **Collect candidate batch**: First server in chain collects $b_{in}^{(i)}$ from each input mixchain and orders, $B = [b_{in}^{(1)} \parallel \ldots \parallel b_{in}^{(m)}]$.

2a. **Verify messages from input chains included**: Every server in mixchain verifies with received hashes. $\forall\ b_{auth}^{(i)} \in B_{auth}$: check $H(b_{auth}^{(i)})$ against received hashes.

Algorithm 8: Message Exchange and Return

1. **Dead-drop exchange**: Given array of messages $[m^{(i)}]$ and corresponding array of dead-drops $[d^{(i)}]$, exchange array indices of messages $m^{(i)}$, $m^{(j)}$ if $d^{(i)} = d^{(j)}$.
2. **Return messages through mixnet**: Reverse mixing, in reverse order, mixchain servers apply inverse permutation π^{-1} and randomize with k_s from Algorithm 6, $[m^{(i)}] \leftarrow [\text{Enc}(k_s, m^{(\pi(i))})]$. Reverse distribution by sending messages back to input mixchains based on indices of message batch. Return messages to users similarly.

Figure 4: Stadium's communication round subprotocols.

6.1 Message Input

Two types of messages are included in a message *batch* to enter a mixchain, noise and user messages. The properties that are verified in the message input protocol are as follows:

- The submitting party knows the routing metadata of the submitted message (output chain selection and dead-drop id).
- All server-submitted noise messages, for which the above property holds true, enter the mixchain.

Note that we allow user messages to be discarded, but honest noise messages must be incorporated.

To achieve the first property, Stadium requires each submitting party to attach a non-interactive zero-knowledge proof that it knows the plaintext of the message authentication piece (i.e., the authentication key) [22, 40]. Given our hybrid encapsulation technique, it is sufficient to verify knowledge of the authentication key as a proxy for the routing metadata the key is used to authenticate. The authentication key is also bound to the round number using a hash (Algorithm 3), so as to prevent replays.

In addition to attaching a proof, servers take an extra step in submitting noise messages to achieve the second property. Servers hash the set of noise messages it wishes to submit to a mixchain and send the hash to each server in the mixchain. The protocol by which servers decide how many messages to send to each mixchain is presented in the next section (§7).

The first server in the mixchain collects all received user and noise messages into a candidate message batch and sends the batch to all servers in the mixchain. Chain servers use the proofs of knowledge and noise hashes to verify the properties are upheld and remove duplicates. The full protocol is given in Algorithm 5.

6.2 Mixing

In the mixing phases (phases 3 and 5 in Figure 3), the accepted message batch is passed through the mixchain and permuted by each server in turn using the hybrid verifiable shuffling protocol (Algorithm 6). The following properties are verified:

- The output message batch (decrypted content) is a permutation of the input message batch (encrypted content), i.e., message integrity is preserved.
- The permutation is random and unknown to any adversary.

An honest server in the mixchain performs two important tasks to achieve the above properties. First, it shuffles messages using a random permutation unknown to adversaries. Second, it verifies that all other servers apply some valid permutation (i.e., do not drop or tamper with messages). For a length ℓ mixchain, each server generates one zero knowledge proof and verifies $\ell - 1$ proofs.

The authentication piece is permuted and verified with a zero-knowledge shuffle proof [3]. The content piece is permuted with the same permutation and verified at the end of the mixchain using authentication keys.

Verifying hybrid content. If the zero-knowledge shuffle proofs of the authentication pieces are all verified, mixchain servers create and combine verifiable decryption shares to threshold decrypt the authentication keys. Verifiable decryption consists of attaching a zero-knowledge proof [11] of the validity of the decryption share. The authentication key is used to verify the permutations applied to the content pieces by checking the message authentication code (MAC). Only then, do mixchain servers perform a second round of verifiable threshold decryption to extract the underlying content.

There are two possible reasons for content authentication failure: a malicious user sent an invalid message, or a malicious server modified the content. It should not be possible for a malicious user to be able to perform a DoS, so the round cannot simply be halted upon detecting a failure. Instead, servers reveal decryption proofs tracing back the path of the faulty message content piece. If all proofs succeed, the faulty message originated from a malicious user; it is dropped and the round is completed through threshold decryption. Otherwise, a malicious server will be identified and the round is halted before communication metadata is revealed.

6.3 Distribution

In the distribution phase (phase 4 in Figure 3), output chains collect messages from the input chains according to the revealed output chain selection. The collected messages are combined to form a new message batch for the second mixing phase. The following property is verified:

- Every message enters its selected output mixchain.

The verifiable distribution protocol is an extended version of noise distribution in the message input protocol, where hash checks were used to verify all noise messages were incorporated into the message batch. It provides a way for an honest server in an input chain to communicate the set of messages intended for an output chain and an honest server within the output chain to verify that the set of messages is included in the message batch.

Each server in the input chain hashes the set of messages intended for an output chain and sends the hash to every member server of that output chain. If every server in the input chain acts honestly, the calculated hashes will be equivalent. The first server of the output chain collects messages received across all the input chains into a candidate message

Observable Variable	Hidden by noise variables	Hiding noise distribution (m servers aggregate)
IO_x^i: inter-chain messages	$\mathcal{A}_x^i$ and $\forall j : \mathcal{B}_x^{i,j}$	Pois($m\lambda_1 + m^2\lambda_2$)
A_x: single-access count	$\forall i : \mathcal{A}_x^i$	Pois($m^2\lambda_1$)
B_x: double-access count	$\forall i, j : \mathcal{B}_x^{i,j}$	Pois($m^3\lambda_2$)

Table 1: Observable variables and noise.

batch. Servers on the output chain verify the candidate message batch using the received hashes. Again, it is sufficient for the input chains to hash only the authentication pieces as a proxy for the dead-drop id metadata contained within the content piece. The full protocol is given in Algorithm 7.

Inter-mixchain verification. As the total number of mixchains grows, the number of hashes a server sends increases, increasing the network costs to set up a connection with each server. This represents a small, but non-constant, processing cost for mixchains. Note however, the total hashing cost remains the same, since the underlying input (message batch) is the same size; it is simply broken into smaller divisions for hashing.

7 DIFFERENTIALLY PRIVATE ROUTING

In this section we identify the observable variables that Stadium exposes and describe how servers generate cover traffic to obscure these variables. Table 1 summarizes the observable variables and noise covering them.

7.1 Observable Variables

Adversaries may monitor the communication between all of Stadium's servers and users. Since Stadium uses verifiable processing (§6) to ensure that all messages that enter a mixchain also exit that chain, no intra-chain links leak information about user communication. The remaining are the following inter-chain 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. (When messages travel on the return path, they are already anonymized.)

To mitigate the information leaked by link (1), between the user to the input chain, users send a message at every round regardless of whether the user is active in a conversation. Therefore, observing users sending messages does not leak any information about their conversations. Stadium does not attempt to obscure the fact that a user uses the system, it only hides its users' communication patterns. We now focus on the remaining two types of links and identify three types of observable variables Stadium exposes to an adversary monitoring these links (see Table 1).

7.1.1 Inter-Chain Message Distribution. The traffic volume emitted from the input chain does not directly reveal the user's communication patterns, but attackers observing the distribution of messages to output chains may still exploit this information to infer a particular message's output chain choice, as shown in Figure 2. By inferring a message's output chain the adversary reduces the anonymity set (Alice can only communicate with others on her output chain). We define the following set of observable variables to reason about this leak of information:

Input-output chain traffic volumes, IO_x^i. The amount of traffic emitted from input chain i to output chain x. There are m^2 such variables.

7.1.2 Dead-Drop Access Counts. Adversaries that have compromised a server on an output chain can observe the dead-drops of all messages sent to that chain. In particular, an adversary can identify which messages are exchanged at a dead-drop, indicating their users are employed in a conversation, and which messages reach a dead-drop alone, indicating their users are idle. We define the following types of observable variables to reason about this leak of information:

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

Double access counts, B_x. The number of dead-drops that receive two messages output from chain x. There are m double access variables.

7.2 Noise Generation

In order to obscure the observable variables that Stadium exposes to attackers, servers inject noise messages to the system at the beginning of the round. So as not to flood the system with noise messages, noise generation responsibilities are split between (honest) servers, which add noise in one collaborative step at the beginning of each round. Verifiable processing (§6) ensures that no malicious server can remove noise later.

We define the following two categories of random variables. The single-access noise variable $\mathcal{A}_x^i$ is the number of messages that travel through input chain i and output chain x and reach a random dead-drop (such that the chance that it is used by any other message is negligible). The double-access variable $\mathcal{B}_x^{i,j}$ is the number of pairs of noise messages that reach the same dead-drop from output chain x, where one message in the pair travels through input chain i, and the other travels through input chain j.

Stadium uses the Poisson distribution for noise generation, $\mathcal{A}_x^i \sim \text{Pois}(\lambda_1)$ and $\mathcal{B}_x^{i,j} \sim \text{Pois}(\lambda_2)$. The Poisson distribution is well suited to Stadium for two main reasons. First, the additive property[1] of Poisson distributions makes it easy to reason about the collaborative noise distribution from

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

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, which occurs often when noise generation is distributed across many servers. In §8 the parameters λ_1, λ_2 are selected such that the aggregate noise (generated by all servers) provides Stadium's privacy guarantees.

The servers draw each of these variables independently and generate noise messages accordingly. For example, generating a noise message for $\mathcal{A}_x^i$ consists of creating a dummy plaintext and routing it to a random dead-drop through input chain i and output chain x. The noise message will route back to the server and will be dropped. Table 1 summarizes the noise variables hiding each observable variable.

8 PRIVACY ANALYSIS

In this section we give a sketch of the analysis for Stadium's privacy guarantees. A complete analysis is given in Appendix B. Recall our differential privacy goal (defined in Section 2) to keep any adjacent communication instances almost equally likely, where adjacency means that one user changes its traffic pattern by selecting a different output chain and/or dead-drop. This definition allows a user Alice, who is talking to Bob, to claim that she is not talking with anyone (sending a message to a dead-drop not shared with another user). Similarly, it allows Alice, who is idle, to claim she is talking to someone. Formally, Stadium is described as a mechanism $M(D)$, where each element in the input database D describes the path of one user message by the tuple (i, o, d): i is the user's selection of input chain, o is her output chain and d is the destination dead drop. (Note that i is directly observable to the adversary monitoring users, while Stadium obscures o and d.) The output of the Stadium mechanism M is a vector of observable variables IO_x^i, A_x, B_x defined in § 7.

Two inputs to Stadium, D and D', are *adjacent* if one user changes their path selection $(i, o, d) \in D$ to $(i, o', d') \in D'$. We show that the Stadium is (ϵ, δ)-differentially private, meaning for all adjacent instances D, D' and set of observable variables S:

$$\Pr[M(D) \in S] \leq e^{\epsilon} \cdot \Pr[M(D') \in S] + \delta$$

Stadium servers inject Poisson distributed noise messages to cover observable variables (§ 7 and Table 1). Our first theorem gives the differential privacy guarantees of the Poisson mechanism. Denote $\text{Pois}(\lambda; k)$ as the probability mass function of $\text{Pois}(\lambda)$ at k, $\frac{\lambda^k e^{-\lambda}}{k!}$.

Theorem 8.1. *A mechanism which adds* $\text{Pois}(\lambda)$ *noise to output* X *is* (ϵ, δ)*-differentially private with respect to changes of 1 in* X *with* $\epsilon = \ln(1 + \frac{c\sqrt{\lambda}}{\lambda})$ *and* $\delta = \frac{e^{-\lambda}(e\lambda)^{\lambda + c\sqrt{\lambda}}}{(\lambda + c\sqrt{\lambda})^{\lambda + c\sqrt{\lambda}}}$. *(Where* $c > 0$ *allows trading higher* ϵ *for lower* δ*.)*

We first analyze the information leaked through the input-output chain distribution (IO_x^i), then use that to analyze the information leaked through the output chain to dead-drop access counts (A_x, B_x). In the analysis, we consider an m-mixchain Stadium configuration.

8.1 Input Chain to Output Chain

The adversary can observe the traffic volumes from Alice's input chain i to all output chains, IO_*^i, and gain statistical information about which output chain Alice is more likely to have used, see Figure 2. The purpose of the input chains is to hide each user's selection of output chain. Intuitively, since the traffic links IO_*^i are noised, it should be almost equally likely for Alice to have travelled to output chain o or output chain o'.

We can capture this intuition formally by considering how the noise variables, $\mathcal{A}_o^i, \mathcal{B}_o^{i,*}, \mathcal{A}_{o'}^i, \mathcal{B}_{o'}^{i,*}$, that cover IO_o^i and $IO_{o'}^i$ change when Alice goes to chain o versus when Alice goes to chain o'. For fixed observations of $IO_o^i, IO_{o'}^i$, Alice switching chains corresponds to an increase of 1 and a decrease of 1 in the corresponding noise variables. Recall from the Poisson additive property, the noise covering each IO_x^i is distributed according to $\text{Pois}(m\lambda_1 + m^2\lambda_2)$. We can then apply Thm 8.1 to $IO_o^i, IO_{o'}^i$ and compose to get bounds $\bar{\epsilon}, \bar{\delta}$ for how much information the adversary learns about Alice's output chain:

$$\Pr[\text{Alice} \rightarrow o] \leq e^{\bar{\epsilon}} \cdot \Pr[\text{Alice} \rightarrow o'] + \bar{\delta}$$

8.2 Output Chain to Dead-drop

Output chains mix user messages with noise to obscure dead-drop access patterns. If an adversary knows Alice's output chain o, the dead-drop access patterns can reveal some information on the existence (or nonexistence) of Alice's communication.

Consider the two adjacent instances Alice talking to Bob and Alice not talking to anyone. Further, consider the simplification where it is known Bob goes to output chain o. Then, if Alice and Bob are talking, their messages would contribute to B_o. If instead, Alice was not talking to anyone, Bob's message would now contribute to A_o and Alice's message would contribute to $A_{o'}$ for some random choice of o'. Observing the same values in these two cases corresponds to an increase of 1 in the noise covering B_o and a decrease of 1 in the noise covering A_o and $A_{o'}$. Naively, these three variables can be composed using Thm 8.1, but a tighter bound can be found by taking advantage of the randomness of Alice's choice o'. Variables A_o and B_o will change by one, but there is not some single variable $A_{o'}$ that the adversary knows changes. Instead, it is only known that 1 variable out of the m in $\vec{A} = (A_1, \ldots, A_m)$ changes, and that variable is chosen at random. Furthermore, by removing the simplification that

Bob's output chain o is known, we get randomness over A_o and B_o as well.

We formalize this intuition in a generalization of the Poisson mechanism in Thm 8.1, which we call the *multidimensional* Poisson mechanism. This theorem captures a $\sqrt{m}$-factor of privacy savings when the change occurs in a random choice of m variables.

THEOREM 8.2. *A mechanism which adds* $\text{Pois}(\lambda)$ *noise to each variable in output* $\vec{X}$*, where* $|\vec{X}| = m$*, is* (ϵ, δ)*-differentially private with respect to a uniform random change of 1 in* $\vec{X}$ *with* $\epsilon = \ln(1 + \frac{c\sqrt{m\lambda}}{m\lambda})$ *and* $\delta = \frac{e^{-m\lambda}(em\lambda)^r}{r^r}$*. (Where* $r = m\lambda + c\sqrt{m\lambda}$ *and* $c > 0$ *is a parameter that allows trading higher* ϵ *for lower* δ*.)*

Unfortunately, this theorem cannot be directly applied since in our case, the variables that change are not uniformly random; the adversary has some small advantage, as shown in §8.1. We can consider a variation of the multidimensional Poisson mechanism where one random variable $X_i \in \vec{X}$ changes by 1, and the probability that X_i changes is p_i. In this case, we show that the worst-case ϵ, δ bounds are given when $\max_i(p_i)$ is maximized. This result intuitively makes sense, as a high $\max_i(p_i)$ means that the adversary has high probability of knowing where the change will occur. In other words, the amount of information leaked is related to the entropy of the probability change vector $\vec{p}$; lower entropy means more information leaked.

LEMMA 8.3. *Given a probability vector* $\vec{p}$*, such that* $\sum_{i=1}^{m} p_i = 1$ *and* $\forall p_i, p_i \geq p_{\min} = \frac{1-p_{\max}}{m-1}$ *for some probability* $p_{\max} > \frac{1}{m}$*. Let* $\vec{v} = v_1, \ldots, v_m$ *be values where* $v_j = \max\{v_i\}$*. Then it holds that,* $\sum_{i=1}^{m} p_i \cdot v_i \leq p_{\max} \cdot v_j + \sum_{i \neq j} p_{\min} \cdot v_i$*.*

Recall from §8.1, we showed the following constraint on the probability of the adversary learning Alice's output chain. For every pair of output chains o, o' where p_o is the probability Alice goes to output chain o, $p_o \leq e^{\bar{\epsilon}} \cdot p_{o'} + \bar{\delta}$. Additionally, $\sum_i p_i = 1$. With these constraints, we show that $p_{\max} = e^{\bar{\epsilon}} \frac{1-\bar{\delta}}{e^{\bar{\epsilon}}+m-1} + \bar{\delta}$ and $\forall\, p_i \neq p_{\max}, p_i = p_{\min} = \frac{1-\bar{\delta}}{e^{\bar{\epsilon}}+m-1}$.

Finally, we can use these "worst-case" uneven distribution probabilities to reduce the problem to a multidimensional Poisson mechanism of $m-1$ $p_{\min}$ probabilities and 1 Poisson mechanism for the $p_{\max}$ probability. Recall A_o, B_o are covered by the sum of noise variables $\mathcal{A}_o^*$ and $\mathcal{B}_o^{*,*}$ that distribute according to $\text{Pois}(m^2\lambda_1)$ and $\text{Pois}(m^3\lambda_2)$ respectively. Applying Thm 8.1 and Thm 8.2 to $\vec{A}, \vec{B}$ using Lemma 8.3 leads to the ϵ, δ differential privacy guarantees for a single round of Stadium communication. Full details of the above analysis can be found in Appendix B.

8.2.1 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 [20]. The composition of k rounds is also differentially private:

THEOREM 8.4. *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* $d > 0$ *trading higher* ϵ *for lower* δ*.*

PROOF. Direct from Theorem 3.20 in [20]. □

8.3 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 $m = 25$ - 1000 servers. Given a strict limit for δ, we search the parameters to minimize e^{ϵ}.

Figure 5 plots the number of noise messages required per round for users to communicate through 10^4 rounds with target $\delta = 10^{-4}$ tolerating $f = 25\%$ compromised servers. The time it takes to process noise messages represents a lower bound on message latency for a particular Stadium configuration. We therefore find that Stadium's latency is particularly sensitive to its privacy parameters for configurations with relatively few servers (that require significantly more noise messages). For larger server configurations, the noise workload per mixchain falls since it is split across the greater number of mixchains. Stadium operates most efficiently when the configuration is set such that the number of noise messages per mixchain is dwarfed by the number of user messages.

9 IMPLEMENTATION

We implement 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 §6) are implemented in C++. In particular, our C++ code implements Bayer and Groth's verifiable shuffling protocol [3] extended for non-interactive proofs via the Fiat-Shamir Heuristic and instantiated over Bernstein's Curve25519 elliptic curve group [5]. The shuffle depends on libNTL [41] for arithmetic. We use standard Go RPCs for networking.

Our implementation uses OpenMP and the Go runtime to parallelize our verifiable computation process. Most operations process each message in a batch independently (§6) and are therefore trivially parallelizable. Moreover, we pipeline

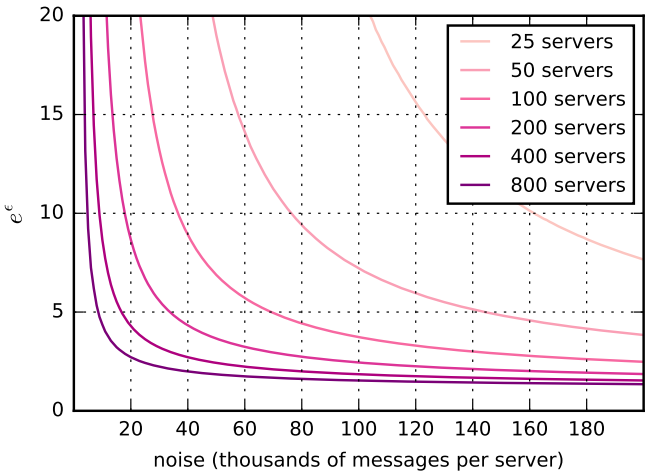


Figure 5: Number of noise messages required per server per communication round to achieve Stadium privacy guarantees with $\delta = 10^{-4}$ for 10^4 communication rounds.

steps in the verifiable computation to increase performance gains from parallelization.

We deploy and evaluate our prototype on Amazon EC2 servers. We randomly assign servers to chains, excluding straggler servers which are under heavy load. We place each server at a different index on ℓ different mixchains (where ℓ is the chain length) so that we maximize server utilization. This allows us to build an m-mixchain Stadium configuration with m servers.

Our prototype does not support client message input; instead, we simulate this by having each server generate some share of the client messages. The implementation also does not include verifiable decryption and currently lacks fault tolerance.

10 EVALUATION

We wish to answer four key questions about Stadium:

- **Horizontal scaling**: How does Stadium's latency change when more users and servers are added?
- **Security guarantees**: How does changing Stadium's ϵ and chain length parameters affect its performance?
- **Operational cost**: What resources does a deployment of Stadium require?
- **State of the art**: How does Stadium compare with Vuvuzela, and what is our design's scaling overhead?

In evaluating Stadium, Table 2 shows that our prototype efficiently supports tens of millions of users given secure system parameters at low cost to individual server operators.

Servers	Users	Latency (seconds)	Bandwidth per server	Chain length	e^ϵ, δ
100	6.3 million	43.6 ± 0.187	18 Mb/s	3	10, 0.0001
100	46 million	238 ± 0.466	88 Mb/s	3	10, 0.0001
100	46 million	504 ± 2.55	142 Mb/s	6	10, 0.0001
100	46 million	876 ± 2.32	173 Mb/s	9	10, 0.0001

Table 2: A summary of our prototype's performance characteristics.

10.1 Experiments

We benchmark Stadium on 100 Amazon `c4.8xlarge` EC2 virtual machines running Linux 4.4, each of which have 36 virtual CPU cores, 60 GB of memory, and 10 Gbps bandwidth. We measure the end-to-end message latency with a *coordinator* server, which configures the server chains and coordinates execution of a round. Individual servers also instrument Stadium's network utilization using the `collectl` tool, and these statistics are aggregated at the coordinator. Messages are short (136 bytes long) to simulate that of a minimal chat application.

For each setting of our independent variables, we run at least three trials. The maximum difference in latency between all machines across any pair of trials is within 4% of the average; the error bars are imperceptible on our plots.

We perform two experiments to quantify the performance characteristics of Stadium. Our first experiment measures the effect of chain length on system latency. We vary chain lengths from 3 to 9 for three different settings of the system's load per server for 100 servers. We discuss these effects in section 10.3.

Our second experiment synthesizes workloads of up to 50 million messages and then distributes these uniformly between configurations of 25, 50, 75, and 100 servers, configured with chain lengths of 3, 6, and 9. Setting $e^\epsilon = 10$, we subtract out noise messages corresponding to the mean of our distribution to obtain goodput values for these parameters (Figure 6). We investigate other values of ϵ in section 10.3.

Limitations. Due to time and cost constraints, we did not evaluate our system against larger configurations of servers (e.g., 500 servers). Because Stadium's latency appears to scale linearly up to 100 servers, we extrapolate its latency to larger deployments by proportionally reducing the per-server message load and interpolating as if these load-balancing trends continued past 100 servers. For instance, to obtain a projected latency of a 200-server deployment for a given load, we halve the per-server goodput, adjust noise loads, and then linearly interpolate over the 100-server configuration to obtain the desired latency. We contend in the following section that these assumptions are reasonable.

We were unable to fully complete experiments for smaller deployments of servers as individual servers lack sufficient

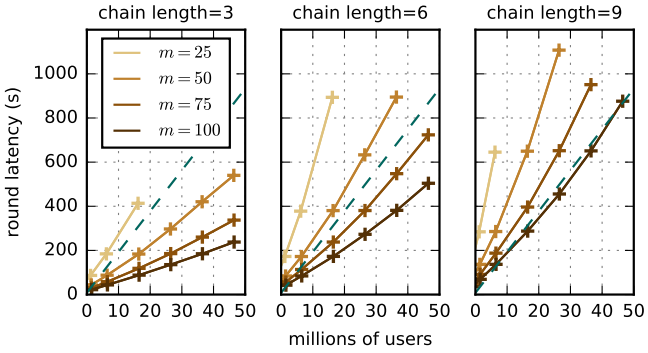


Figure 6: Stadium message latency as a function of the number of connected users given $m \in \{25, 50, 75, 100\}$ servers. For comparison, we plot Vuvuzela's best projected performance as a dashed line for chain length $\ell = 3$. At a chain length of 3 and a latency of 200s, our deployment supports 3.6× the users Vuvuzela does at 1/15th the per-server operating cost.

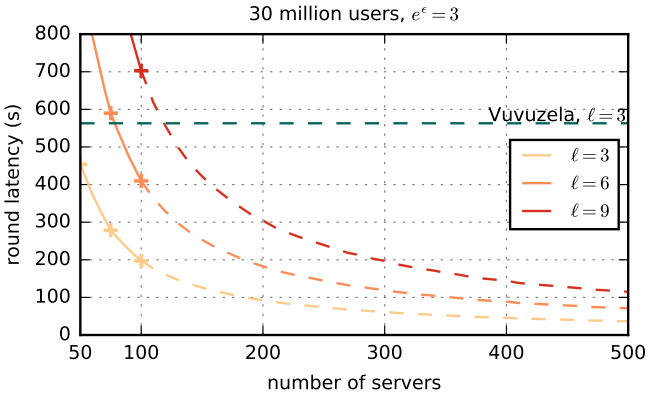


Figure 7: Stadium message latency as a function of the number of servers. We plot Stadium's projected performance past 100 servers with dashed lines (§10.1). With 500 servers, Stadium can support 30 million users with just under two minutes of latency and a chain length of 9; Vuvuzela's rounds require more than nine minutes.

memory for our implementation to support very large message batch sizes. We note that small numbers of servers are not practical for deploying Stadium.

10.2 Horizontal scaling

The end-to-end latency of Stadium grows in proportion to the number of total messages, and it shrinks in proportion to the number of total servers.

First, we can observe in Figure 6 that for a fixed number of total servers, the round latency increases near-linearly with respect to the total number of user messages, with a small quadratic term. For example, with a chain length of 6 and 50 servers, Stadium supports 16 million users at a latency of 380s. From here, increasing the number of users by 2.25× increases latency by 2.36×. With additional user messages, each mixchain receives a proportionally larger message batch to process, e.g., decrypting and verifiably shuffling these messages. Furthermore, we observe a small quadratic term which results from the message distribution phase, where each pair of mixchains must communicate. Also recall that the number of noise messages remains constant with respect to the number of users and represents an initial start-up latency that is better amortized over more users.

Figure 7 shows how round latency decreases with the number of Stadium servers. The initial points in this figure, between 50 and 100 servers, are based on our measurements (in Figure 6). To illustrate how Stadium would scale with more servers we extrapolate from our measurements based on observed trends. For instance, our measurements show that Stadium can achieve a latency of 400s with 100 servers with a chain length of 6; doubling the deployment size to 200 servers halves round latency to under 200s. This follows since for some fixed number of messages, a proportional increase in the number of servers allows Stadium to divide the real and noise messages equally among all servers (ignoring the small quadratic cost from message distribution). Additionally, increasing the number of total servers decreases the noise required per server (§8).

Even at relatively small scales, Stadium outperforms Vuvuzela in terms of latency and throughput. For example, for a chain length of 3, Vuvuzela can only support 10 million users in 200s; Stadium (with 100 servers) supports 3× as many users in the same amount of time. As we increase the total number of servers in the system, we expect that these relationships will continue to hold because our per-server noise message and user message loads will both decrease.

10.3 Security guarantees

Chain length. The length of Stadium's mixchains represents a security-performance trade-off. Stadium's verifiable processing (§6) strategy, which ensures the differential privacy guarantees are upheld, hinges on the property that every mixchain contains at least one honest verifying server. A longer chain length increases the likelihood that this property holds, but pays in performance since more servers need to handle each message.

If auxiliary information is known about the participating servers and trust assumptions of users, it may be possible to arrange servers via some policy. However, in absence of such a policy, we can randomly assign servers to mixchains

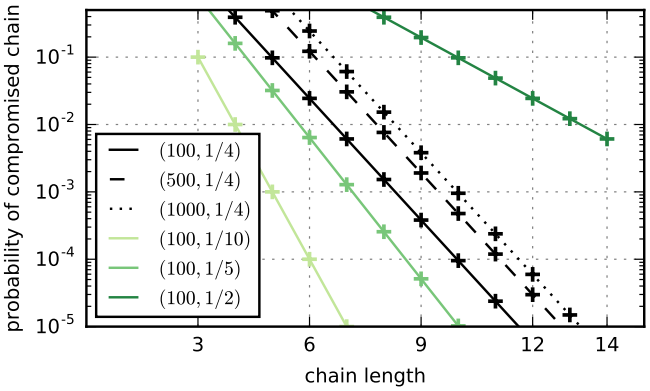


Figure 8: Probability of compromised mixchain existing with random server assignment given the fraction of compromised servers f, chain length ℓ, and number of mixchains m. (Labels in the legend denote a setting of the parameters (m, f).)

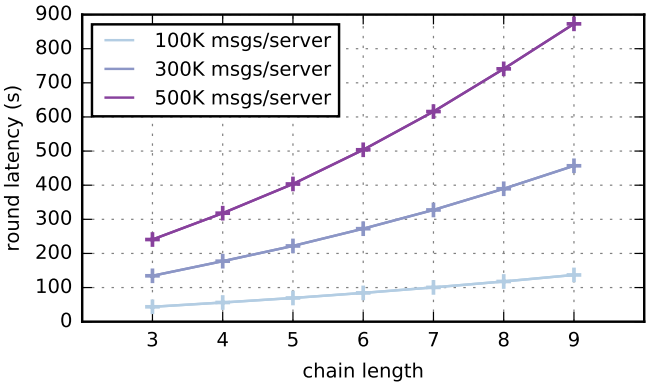


Figure 9: Chain length and message latency (m=100 servers).

and analyze the probability of one honest server per chain given some assumed upper bound fraction of compromised servers f. The probability for a specific chain to be assigned all compromised servers (with replacement) is f^ℓ where ℓ is the chain length. We can bound the probability of some chain being composed of all compromised servers with the union bound, $m \cdot f^\ell$ where m is the number of chains. This probability falls exponentially with chain length, and grows only linearly in the number of chains, as shown in Figure 8.

To evaluate the effects of chain length on performance, we perform an additional benchmark varying chain length from 3 to 9 for $m = 100$ servers. We test three different per-server loads: 100, 300, and 500 thousand messages per server. The choice of chain length has a significant effect on Stadium's latency. Indeed, Figure 9 illustrates a quadratic

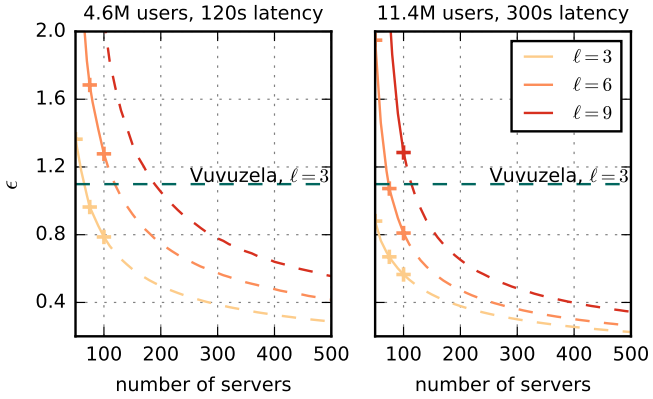


Figure 10: Stadium's privacy bound ϵ as a function of the number of servers. Adding more servers to Stadium allows it to surpass Vuvuzela in performance for both low- and medium-latency targets. (We choose goodput and latency settings which match Vuvuzela's for a clear comparison.)

relationship between the number of servers in a chain and round latency. This is expected since verifiable processing requires each server to verify a proof for every other server in the mixchain. Additionally since we pipeline servers across chains, this corresponds to a quadratic processing cost. As a result, our design is sensitive to the trust assumptions underpinning choices of chain length.

Differential privacy guarantees. We examine Stadium's privacy guarantees by evaluating changes in ϵ (bound on information revealed about communication metadata) given a fixed $\delta = 10^{-4}$ (failure probability with no privacy guarantees).

Since Stadium's latency scales down in proportion to the number of servers, we analyze our ϵ guarantees by fixing a target system goodput and latency while varying the number of servers. Observe in Figure 10 that whether we target a medium or high volume of users, our ϵ-bounds converge quickly towards a high level of privacy. For instance, with a chain length of 6, Stadium supports 4.6 million users at 120 seconds of latency with $e^\epsilon = 18.4$ with 50 users, $e^\epsilon = 5.3$ with 75 users, and $e^\epsilon = 3.5$ with 100 users. Extrapolating to 500 servers, we can achieve a small value of $e^\epsilon = 1.5$, providing a high amount of plausible deniability for Alice.

10.4 Deployment cost

Communication is the dominant factor in the cost of maintaining a server. Deploying Stadium on 100 servers with chain length of 6 requires a server to send data at about 142 Mbps (see Table 4) We achieve low per-machine bandwidth costs without significant per-user bandwidth costs:

Chain length	Bandwidth total (Mb/s)	Bandwidth per user (b/s)
3 (Stadium)	8767	189
6 (Stadium)	14246	308
9 (Stadium)	17329	374
3 (Vz.)	3984	87
6 (Vz.)	7968	173

Table 3: *Total* and *per-user* bandwidth costs of Stadium (deployed on $m = 100$ servers for 46M users) and Vuvuzela per-round. We do not pay a significant per-user bandwidth overhead (less than $5\times$) over Vuvuzela.

Chain length	Round latency (s)	Communication (GB)	Bandwidth (Mb/s)	Bandwidth vs. Vuvuzela
3	238	2.607	88	6.60%
6	504	8.978	142	10.72%
9	876	18.976	173	13.05%

Table 4: Communication and bandwidth costs of *one server* for one round of communication ($m = 100$, 46M users). Notice that our bandwidth costs are $7 - 15\times$ lower than those of Vuvuzela (at a chain length of 3).

the marginal cost of adding another user is small and insensitive to the choice of chain length. Table 3 shows that our bandwidth costs are likewise low; it costs 142 Mbps to operate Stadium with a chain length of 6. Note that we can arbitrarily drive down this cost by throttling rounds at the cost of latency.

Every round, a client must send the message itself wrapped in 2ℓ layers of onion encryption (32 bytes of overhead each) in addition to the output chain metadata (32 bytes) and a proof that the message was well-formed (32 bytes). If message size is S bytes and round latency is T seconds, then a client's bandwidth cost C is

$$C = \frac{S + 64l + 64}{T}$$

Setting a chain length of $\ell = 6$, $S = 136$ bytes and $T = 503$ seconds, clients need to send at the rate of 1.16bps. This cost scales linearly with an increasing message size or chain length.

To illustrate the feasibility of deploying Stadium, consider that of the top 300 relays in the Tor network, each offers more than 140 Mbps of bandwidth (see [43]). Using these measurements, we can estimate how much it costs to deploy a server: if bandwidth costs \$0.63/month per unit of Mbps [36], then it should cost a Stadium operator about \$110 a month to run a server for one month, given a chain length of 9.

10.5 Comparison with Vuvuzela

Like Vuvuzela, Stadium allows an operator to set the security parameters ϵ and chain length; with these two variables set, one can fix either a latency or a throughput parameter to obtain the other. Clients then submit messages into the system and receive them at fixed intervals.

Unlike Vuvuzela, an operator can add servers to Stadium, which represents a fifth variable and a fourth degree of freedom. The trust assumptions of Stadium are then slightly different from those of Vuvuzela. Vuvuzela requires users to greatly trust a few points of failure, while Stadium distributes its computation across a high number of less trustworthy servers.

Due to these differences, we conservatively pick the performance characteristics of Vuvuzela set at $\ell = 3$ for the sake of comparison. Our evaluation shows that at 100 servers and chain lengths of 3, 6, and 9, Stadium is competitive with Vuvuzela in terms of latency and privacy at large scales. In addition, whether we desire low latency, high throughput, strong security, or all of these characteristics, we can incrementally add servers to reach that point. Furthermore, Vuvuzela's requirement of 1.3 Gbps of bandwidth limits deployment to large organizations (no Tor relay provides more than 1 Gbps of bandwidth [43]) while a modest requirement of 142 Mbps allows individuals to scale out Stadium.

11 RELATED WORK

Anonymous communication systems. The study of online anonymous communication dates back to Chaum's work on mixnets [8] and DC-nets [9]. Systems based on these primitives [10, 17, 24, 25, 27, 31] share similar design principles of layering encryption, then batching, permuting, and forwarding messages. Similar to the layering encryption of mixnets, onion routing [42] allows for specific subsets of mixservers to be selected, amenable to peer-to-peer and distributed systems. Systems based on these principles achieve high throughput due to the relatively inexpensive cost of their operations. Tor [18], the most popular anonymity system in use, is an overlay network of onion routers. Tor supports its millions of users with a horizontally scaling design, where capacity can be bolstered through the addition of volunteer providers – a shared design goal with Stadium.

Recent systems have focused on protecting against strong, active adversaries with the ability to monitor and inject traffic at a global scale. The scalability of many of these systems are limited by vertically scaling designs, where providers must run more powerful infrastructure to support more users. Riposte [14] for point-to-point communication and Talek [12] for publish-subscribe are based on the cryptographic technique of private information retrieval (PIR). Dissent [15, 46] is an anonymous messaging system based on DC-nets. These systems are restricted from horizontal scalability by computationally expensive PIR techniques and communication intensive broadcast requirements.

Vuvuzela [44], the current largest-scale system that protects against strong adversaries, is a mixnet-based private messaging system that achieves high throughput, in part, by relaxing the provable privacy guarantees from cryptographic indistinguishability to differential privacy. Stadium is also mixnet-based and uses a similar differential privacy definition, but is designed to scale horizontally with added providers, support a higher throughput, and cut operating costs of providers.

More recently, there have been a few systems that aim for the same goals of provable privacy against strong adversaries *and* horizontal-scalability as Stadium. Pung [1] uses PIR techniques for private messaging. Although Pung is horizontally scalable, it does not aim to achieve the same levels of throughput as Stadium. Instead, Pung focuses on the threat model of all untrusted providers. In contrast, Stadium's privacy is dependent on some providers faithfully executing the protocol and verifying others. Atom [32] is an anonymous publishing system that uses verifiable processing techniques very similar to Stadium's. However, with differing applications and privacy goals, Stadium and Atom reach different mixnet design points. In particular, Atom uses a random permutation network to achieve strong anonymity sets, while Stadium relaxes anonymity with differential privacy to reason about information leakage of a shallow two-layer network, greatly reducing latency at the cost of perfect mixing. Lastly, Loopix [37] is an asynchronous messaging system based on onion routing that achieves some provable properties through random Poisson delays.

Mixnets. Stadium builds on two classic threads of mixnet research, parallel mixing and verifiable mixing. Parallel mixnets [17, 19, 26, 38] were suggested as a way to horizontally scale traditional sequential mixnets. Stadium adopts ideas from this literature and provides a differential privacy analysis to bound the information leakage of message routing within the mixnet from traffic analysis [6].

Verifiable shuffles make up the majority of Stadium's message processing. Verifiable shuffles were originally proposed to allow globally verifiable e-voting [3, 13, 23, 35], another privacy related scenario; but they have typically been considered prohibitively expensive for low-latency applications such as messaging. However, with recent cryptographic advances [3] in shuffle efficiency and, more importantly, parallelization, Stadium shows that they are nearing practicality. Finally, Stadium combines efficiently engineered verifiable shuffles, hybrid verifiable shuffling, and an intra-mixchain verification strategy to allow verification workloads of the complete mixnet to scale.

We explored the use of randomized partial checking (RPC) [29] as a more robust verification alternative to zero knowledge proofs. RPC allows a small probability that an adversary will directly learn the path of a message through an honest server. Even though this probability is small, it grows too much for our privacy budget over many servers and many rounds. Other techniques using dummy/trap messages, which allow a small probability for an adversary to replace a message, may be more amenable to our privacy analysis [32].

12 CONCLUSION

Stadium is a point-to-point messaging system that provides metadata and data privacy while scaling its work efficiently across hundreds of *low-cost* providers operated by different organizations. In contrast to previous strong anonymity systems, Stadium can be deployed incrementally using small providers and does not require a small centralized anytrust set. We show that Stadium can scale to support 4× more users than previous systems using servers that cost an order of magnitude less to operate.

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. We would also like to thank our reviewers and our shepherd Michael Walfish. This work was supported by NSF awards CNS-1413920 and CNS-1414119, and by Google.

REFERENCES

[1] Sebastian Angel and Srinath T. V. Setty. 2016. Unobservable Communication over Fully Untrusted Infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.* 551–569. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/angel

[2] Kevin S. Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas C. Sicker. 2007. Low-resource routing attacks against Tor. In *Proceedings of the 2007 ACM Workshop on Privacy in the Electronic Society, WPES 2007, Alexandria, VA, USA, October 29, 2007.* 11–20. https://doi.org/10.1145/1314333.1314336

[3] Stephanie Bayer and Jens Groth. 2012. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In *EUROCRYPT (Lecture Notes in Computer Science)*, Vol. 7237. Springer, 263–280. http://dx.doi.org/10.1007/978-3-642-29011-4

[4] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby (Eds.). ACM, 62–73. http://dl.acm.org/citation.cfm?id=168588

[5] Daniel J. Bernstein. 2006. *Curve25519: New Diffie-Hellman Speed Records.* Springer Berlin Heidelberg, Berlin, Heidelberg, 207–228. https://doi.org/10.1007/11745853_14

[6] Nikita Borisov. 2005. An Analysis of Parallel Mixing with Attacker-Controlled Inputs. In *Privacy Enhancing Technologies (Lecture Notes in Computer Science)*, George Danezis and David M. Martin Jr (Eds.), Vol. 3856. Springer, 12–25. http://dx.doi.org/10.1007/11767831_2

[7] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a distance: website fingerprinting attacks and defenses.

In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. 605–616. https://doi.org/10.1145/2382196.2382260

[8] David Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 24, 2 (1981), 84–88. https://doi.org/10.1145/358549.358563

[9] David Chaum. 1988. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *J. Cryptology* 1, 1 (1988), 65–75. https://doi.org/10.1007/BF00206326

[10] David Chaum, Farid Javani, Aniket Kate, Anna Krasnova, Joeri de Ruiter, and Alan T. Sherman. 2016. cMix: Anonymization by High-Performance Scalable Mixing. *IACR Cryptology ePrint Archive* 2016 (2016), 8. http://eprint.iacr.org/2016/008

[11] David Chaum and Torben Pryds Pedersen. 1992. Wallet Databases with Observers (Extended Abstract). In *Advances in Cryptology—CRYPTO '92 (Lecture Notes in Computer Science)*, Ernest F. Brickell (Ed.), Vol. 740. Springer-Verlag, 1993, 89–105.

[12] Raymond Cheng, William Scott, Bryan Parno, Arvind Krishnamurthy, and Thomas Anderson. 2016. *Talek: a Private Publish-Subscribe Protocol*. Technical Report. University of Washington.

[13] Michael E. Clarkson, Stephen Chong, and Andrew C. Myers. 2007. Civitas: A Secure Remote Voting System. In *Frontiers of Electronic Voting, 29.07. - 03.08.2007*. http://drops.dagstuhl.de/opus/volltexte/2008/1296

[14] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An Anonymous Messaging System Handling Millions of Users. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. 321–338. https://doi.org/10.1109/SP.2015.27

[15] Henry Corrigan-Gibbs and Bryan Ford. 2010. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*. 340–350. https://doi.org/10.1145/1866307.1866346

[16] Artur Czumaj and Berthold Vöcking. 2014. Thorp Shuffling, Butterflies, and Non-Markovian Couplings. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*. 344–355.

[17] George Danezis, Roger Dingledine, and Nick Mathewson. 2003. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *2003 IEEE Symposium on Security and Privacy (S&P 2003), 11-14 May 2003, Berkeley, CA, USA*. 2–15. https://doi.org/10.1109/SECPRI.2003.1199323

[18] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*. 303–320. http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html

[19] Roger Dingledine, Vitaly Shmatikov, and Paul F. Syverson. 2004. Synchronous Batching: From Cascades to Free Routes. In *Privacy Enhancing Technologies, 4th International Workshop, PET 2004, Toronto, Canada, May 26-28, 2004, Revised Selected Papers*. 186–206. https://doi.org/10.1007/11423409_12

[20] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407. http://dx.doi.org/10.1561/0400000042

[21] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. 332–346. https://doi.org/10.1109/SP.2012.28

[22] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO (Lecture Notes in Computer Science)*, Andrew M. Odlyzko (Ed.), Vol. 263. Springer, 186–194. http://dx.doi.org/10.1007/3-540-47721-7_12

[23] Jun Furukawa and Kazue Sako. 2001. An Efficient Scheme for Proving a Shuffle. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*. 368–387. https://doi.org/10.1007/3-540-44647-8_22

[24] Nethanel Gelernter, Amir Herzberg, and Hemi Leibowitz. 2016. Two Cents for Strong Anonymity: The Anonymous Post-office Protocol. *IACR Cryptology ePrint Archive* 2016 (2016), 489. http://eprint.iacr.org/2016/489

[25] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. 2003. *Herbivore: A Scalable and Efficient Protocol for Anonymous Communication*. Technical Report. Cornell University.

[26] Philippe Golle and Ari Juels. 2004. Parallel mixing. In *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*. 220–226. https://doi.org/10.1145/1030083.1030113

[27] Ceki Gülcü and Gene Tsudik. 1996. Mixing Email with Babel. In *1996 Symposium on Network and Distributed System Security, (S)NDSS '96, San Diego, CA, February 22-23, 1996*. 2–16. https://doi.org/10.1109/NDSS.1996.492350

[28] Johan Håstad. 2006. The square lattice shuffle. *Random Struct. Algorithms* 29, 4 (2006), 466–474.

[29] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. 2002. Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*. 339–353. http://www.usenix.org/publications/library/proceedings/sec02/jakobsson.html

[30] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul F. Syverson. 2013. Users get routed: traffic correlation on Tor by realistic adversaries. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. 337–348. https://doi.org/10.1145/2508859.2516651

[31] Dogan Kesdogan, Jan Egner, and Roland Büschkes. 1998. Stop-and-Go-MIXes Providing Probabilistic Anonymity in an Open System. In *Information Hiding, Second International Workshop, Portland, Oregon, USA, April 14-17, 1998, Proceedings*. 83–98. https://doi.org/10.1007/3-540-49380-8_7

[32] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. 2017. Atom: Horizontally scaling strong anonymity. In *Symposium on Operating Systems Principles*.

[33] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. 2016. Riffle: An Efficient Communication System With Strong Anonymity. *PoPETs* 2016, 2 (2016), 115–134. http://www.degruyter.com/view/j/popets.2015.2016.issue-2/popets-2016-0008/popets-2016-0008.xml

[34] David Lazar and Nickolai Zeldovich. 2016. Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*. 571–586. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/lazar

[35] C. Andrew Neff. 2001. A verifiable secret shuffle and its application to e-voting. In *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001*. 116–125. https://doi.org/10.1145/501983.502000

[36] W. Norton. 2010. *Internet Transit Prices - Historical and Projected*. Technical Report. http://drpeering.net/white-papers/Internet-Transit-Pricing-Historical-And-Projected.php

[37] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*. 1199–1216. https://www.usenix.org/conference/

usenixsecurity17/technical-sessions/presentation/piotrowska

[38] Charles Rackoff and Daniel R. Simon. 1993. Cryptographic Defense Against Traffic Analysis. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing (STOC '93)*. ACM, New York, NY, USA, 672–681. https://doi.org/10.1145/167088.167260

[39] Alan Rusbridger. 2013. The Snowden Leaks and the Public. The New-York Review of Book. (Nov. 2013).

[40] Claus-Peter Schnorr. 1989. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. 239–252. https://doi.org/10.1007/0-387-34805-0_22

[41] Victor Shoup. 2016. NTL: A Library for doing Number Theory. http://www.shoup.net/ntl/. (2016).

[42] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. 1997. Anonymous Connections and Onion Routing. In *1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, Oakland, CA, USA*. 44–54. https://doi.org/10.1109/SECPRI.1997.601314

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

[44] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*. 137–152. https://doi.org/10.1145/2815400.2815417

[45] Tao Wang and Ian Goldberg. 2013. Improved website fingerprinting on Tor. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*. 201–212. https://doi.org/10.1145/2517840.2517851

[46] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. 2012. Dissent in Numbers: Making Strong Anonymity Scale. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*. 179–182. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/wolinsky

A VERIFIABLE PROCESSING SECURITY ARGUMENTS

A.1 Preliminaries

In this section, we will show certain invariants on the message batch hold during processing. To ensure invariants are held, a number of cryptographic protocols are used for verifiable processing. We start by outlining the details of them here. For clarity, we will index servers with i and mixchains with j.

A.1.1 Threshold decryption. Each server s_i generates a public, private key pair. $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{KGen}(1^n)$ of the form (g^{x_i}, x_i) for ElGamal encryption. Furthermore, each mixchain, $\mathsf{mc}_j = \left\{\mathsf{s}_1^{(j)}, \ldots, \mathsf{s}_l^{(j)}\right\}$, has public key derived from the member servers' key pairs. The private key is held in shares for threshold decryption.

$$\mathsf{pk}_j = \prod_{\mathsf{s}_i \in \mathsf{mc}_j} \mathsf{pk}_i = g^{\sum_i x_i}$$

$$c = \mathsf{Enc}(\mathsf{pk}_j, m) = (g^r, m \cdot \mathsf{pk}_j^y) = (g^r, m \cdot g^{r \sum_i x_i})$$

Then each server i can produce a decryption share of c as follows:

$$\mathsf{DecShare}(\mathsf{sk}_i, c) = (g^r)^{x_i} = g^{r x_i}$$

The server's secret key, x_i, cannot be extracted by discrete log assumption. All shares can be combined to decrypt c:

$$\mathsf{ThreshDec}(c) = \frac{m \cdot g^{r \sum_i x_i}}{\prod_i \mathsf{DecShare}(\mathsf{pk}_i, c)} = \frac{m \cdot g^{r \sum_i x_i}}{\prod_i g^{r x_i}} = \frac{m \cdot g^{r \sum_i x_i}}{g^{r \sum_i x_i}} = m$$

A.1.2 Proof of knowledge of discrete log. This protocol, $\mathsf{NIZK}_{know}(g^r, g)$, allows a prover to generate a proof, proof_{know}, that they know r that can be verified with protocol $\mathsf{Vf}_{know}(g^r, g, \mathsf{proof}_{know})$. We use the Schnorr protocol [40] for this purpose, which is made non-interactive using the Fiat-Shamir heuristic [22].

A.1.3 Proof of equality of discrete logs (verifiable decryption). This protocol, $\mathsf{NIZK}_{dec}((g, g^a), (h, h^a))$, allows a prover to generate a proof, proof_{dec}, that they know a that can be verified with protocol $\mathsf{Vf}_{dec}((g, g^a), (h, h^a), \mathsf{proof}_{dec})$. This is used for verifiable decryption when a server decrypts by raising random value $(g^r)^{\mathsf{sk}} = g^{rx}$. The server must then prove they used their actual secret key x, by proving the equality of discrete logs of $((g, \mathsf{pk} = g^x), (g^r, g^{rx}))$. We use the Pedersen-Chaum protocol [11] again extended with the Fiat-Shamir heuristic [22].

A.1.4 Proof for shuffle permutation (verifiable shuffle). A protocol, $\mathsf{NIZK}_{shuf}(B, B', \pi)$, to allow a prover to generate a proof, proof_{shuf}, of the correctness of a shuffle that can be verified with protocol $\mathsf{Vf}_{shuf}(B, B', \mathsf{proof}_{shuf})$. The protocol verifies that two message batches B, B' are permutations of each other. Specifically, for some valid permuation π:

$$B = \{(g^{r_1}, m_1 \cdot g^{r_1 x}), \ldots, (g^{r_n}, m_n \cdot g^{r_n x})\}$$

$$B' = \left\{(g^{r'_1}, m_{\pi(1)} \cdot g^{r'_1 x}), \ldots, (g^{r'_n}, m_{\pi(n)} \cdot g^{r'_n x})\right\}$$

We use the Bayer Groth shuffle [3] for this protocol.

A.2 Message Input

The following properties are verified:

- The submitting party knows the routing metadata of the submitted message (output chain selection and dead-drop id).
- All server-submitted noise messages, for which the above property holds true, enter the mixchain.

Proof. Submitting parties provide a proof of knowledge of the authentication key. The content is encrypted with an authenticated encryption scheme using this key (Encrypt-then-MAC). Shown in the Mixing section (A.3), if the content is not correctly authenticated, servers can trace back the message to find if the submitting party is at fault, and then reject the message. This protocol allows us to use the authentication key proof of knowledge as a proxy for proof of knowledge for the content.

Assume then, that if a proof of knowledge verifies correctly, then it was created by a party that knows the routing metadata of the message. We then need to show that submitting parties cannot reuse messages and associated proofs generated by other parties.

(1) **Resubmit message from a previous round**: The authentication key is derived from the round number so authentication of the MAC will not succeed.
(2) **Resubmit message from current round**:
   (a) **Resubmit message to different mixchain**: The authentication key is encrypted under a different mixchain's public key. Thus, when decryption occurs, the derived authentication key will be incorrect and authentication of the MAC will not succeed.
   (b) **Resubmit message to same mixchain**: Servers in the mixchain remove (deduplicate) messages with duplicate proofs (keeps one copy of the message). Since the proof of knowledge used is non-malleable, resubmitted messages will be removed.

Thus, messages and proofs that are reused are caught by the system, so the only way for a message to successfully process is if the submitting party knows the routing metadata.

Next, we show the second invariant holds. To submit a batch of noise messages to a mixchain, a server also sends a hash of the batch to every server in the mixchain. Thus, since the candidate message batch is ordered in a deterministic fashion by the first server in the mixchain, each member server of the mixchain checks all the noise hashes. If a noise hash fails, the server does not accept the candidate batch, and audits which of the first server or the noise-generating server erred. If all noise hashes succeed, then that means all the noise messages each noise-generating server submitted are part of the candidate batch. □

A.3 Mixing

The following properties are verified:

- The output message batch (decrypted content) is a permutation of the input message batch (encrypted content), i.e., message integrity is preserved.
- The permutation is random and unknown to any adversary.

Proof. Servers perform verifiable shuffles of the message batch in sequence. Each time, verifying that the mixing server takes the correct authentication input (output of the previous server) and produces a valid authentication output (the verifiable shuffle proof succeeds, i.e., a valid permuation was applied). This ensures that the encrypted authentication batch outputted from the mixchain is a valid permuation of the authentication batch inputted to the mixchain. Next, servers use verifiable decryption to generate shares for threshold decryption. Servers can verify using the verifiable decryption proofs that the shares they receive are correct. Thus, we can be confident the decrypted authentication keys are correct.

Now, we need to show the decrypted message content is correct. Servers use the authentication keys to authenticate the MACs of the content. If MACs succeed, it means that both the proper permutations were applied to the content, and that the content was properly onion-decrypted so the values are correct. There are two possible reasons for content MAC authentication failure: a malicious user sent an invalid message, or a malicious server modified the content.

(1) A malicious user sent a malformed message.
(2) A malicious server modified the message.

Servers reveal decryption proofs tracing back the path of the faulty message content piece, in order to prove that they correctly peeled off the onion encryption layer of the content. Since just the path of the faulty message is revealed, the mixing of other messages is unaffected. To reveal the path, each server, in reverse turn, reveals the permutation relation only of the index they claim was the faulty message, along with a verifiable decryption proof for the ephemeral symmetric key of the index. Other servers verify the permutation relation revealed is correct by replaying the server's rerandomization, i.e., peeling off the onion layer with the revealed symmetric key and checking if the result matches.

The symmetric key is verifiably decrypted so it known to be correct for the index given by the server. For the symmetric key to decrypt to the proper value, the creater of the message in the specified index must have known the decrypted value. Thus, it is possible for a server to forge a different path for a message that it knows the decryption of, but not for messages where the decryption is unknown. Even if it forges a path for a message it knows, other honest server proofs will still succeed. As long as permutation relations are not allowed to be reused, all faulty messages will still be caught.

If all proofs succeed, the faulty message originated from a malicious user; it is dropped and the round is completed through a second round of threshold decryption. Otherwise, a malicious server will be identified and the round is halted before communication metadata is revealed. The second round of threshold decryption also uses verifiable decryption, so the decrypted symmetric keys are correct. The symmetric keys are then known to all servers in the mixchain, and the content can be decrypted.

Next, we show the permutation applied by the mixchain is random and unknown to any adversary. First we show the messages pass through an honest server. We assume that each mixchain contains an honest server, so we only need to show that every server in the mixchain mixes the message batch. Mixchain servers cannot bypass a server in the mixchain since every server must supply a verifiable shuffle proof. If the inputs and outputs of the verifiable shuffle proof do not line up with the adjacent proofs, the round will be halted. Additionally, an honest server will only provide decryption shares to an output batch that has valid verifiable shuffle proofs from all mixchain servers. Thus, every server applies a permutation to the message batch.

Next, we need to show the permuation is is random and unknown to any adversary. If just one server applies a random and unknown permutation, then the property holds for the whole mixchain. Let's show this for the permutation applied by the honest server.

(1) **The permutation is random**: The honest server follows the protocol and picks a random permutation π.
(2) **The permuation is unknown**:
  (a) **The permutation is not revealed in the forward direction**:
    (i) Authentication piece: The permutation applied to the authentication pieces is hidden since the messages are re-randomized to a group element using ElGamal's malleability property.
    (ii) Content piece: The permutation applied to the content piece is hidden since the output messages consist of a symmetric decryption of the input messages using an unknown symmetric key.
  (b) **The permutation is not revealed in the reverse direction**: The reverse permutation applied to the exchanged content is hidden since the messages are encrypted using an unknown symmetric key. The message in the reverse direction is different from the message in the forward direction, so using the same symmetric key is not a problem.

Thus, the permutation applied by the honest server is both random and unknown to the adversary. □

A.4 Distribution

The following properties are verified:

- Every message enters its selected output mixchain.

Proof. We have already shown the decryption of the input chain content is correct. Thus, every server in the input chain knows the correct output chain destination for each message in the message batch. Each server groups the messages deterministically by output chain destination, and hashes the batch destined for each output chain. If all servers behave honestly, the hash for each output chain destination matches.

Output chain servers receive hashes from every server of every input chain. If they receive different hashes from servers of the same input chain, it means the servers in the input chain disagree on the output of their chain and an audit should begin.

The first server of the output chain receives message batches from the last server of every input chain and organizes them deterministically into a candidate message batch. Like in the message input case, the delivered messages from each input chain are verified with the received hashes. If a hash verification fails, the server does not accept the message batch, and audits which of the first server of the output chain or the last server of the failed input chain erred. If all verification succeed, then it means the input chain successfully agreed on which messages to send to each output chain, and the output chain agreed that all messages were correctly incorporated into the message batch. □

B PRIVACY PROOFS

B.1 Poisson Mechanism

Theorem B.1. *A mechanism which adds* $\text{Pois}(\lambda)$ *noise to output* X *is* (ϵ, δ)*-differentially private with respect to changes of 1 in* X *with* $\epsilon = \ln(1 + \frac{c\sqrt{\lambda}}{\lambda})$ *and* $\delta = \frac{e^{-\lambda}(e\lambda)^{\lambda+c\sqrt{\lambda}}}{(\lambda+c\sqrt{\lambda})^{\lambda+c\sqrt{\lambda}}}$*. (Where* $c > 0$ *allows trading higher* ϵ *for lower* δ*.)*

Denote $\text{Pois}(\lambda, x)$ as the probability mass function of $\text{Pois}(\lambda)$ at x, $\frac{\lambda^x e^{-\lambda}}{x!}$.

Proof. Using the Poisson probability mass function, it follows that we require:

$$\begin{aligned}\text{Pois}(\lambda, x-1) &= \frac{\lambda^{x-1}e^{-\lambda}}{(x-1)!} \\ &= \left(\frac{x}{\lambda}\right)\frac{\lambda^{x}e^{-\lambda}}{x!} \\ &= \left(\frac{x}{\lambda}\right)\cdot \text{Pois}(\lambda, x)\end{aligned}$$

When $x \leq \lambda + c\sqrt{\lambda}$, we find:

$$\text{Pois}(\lambda, x-1) \leq \left(\frac{\lambda + c\sqrt{\lambda}}{\lambda}\right)\cdot \text{Pois}(\lambda, x)$$

When $x > \lambda + c\sqrt{\lambda}$, the probability difference can be bounded using the Poisson tail bound to form the additive δ term. The Poisson tail bounds for $\text{Pois}(\lambda)$:

$$\begin{aligned}\Pr[X \geq x] &\leq \frac{e^{-\lambda}(e\lambda)^x}{x^x} && x > \lambda \\ \Pr[X \leq x] &\leq \frac{e^{-\lambda}(e\lambda)^x}{x^x} && x < \lambda\end{aligned}$$

Thus,

$$\text{Pois}(\lambda, x-1) \leq e^{\epsilon}\cdot \text{Pois}(\lambda, x) + \delta$$

$$e^{\epsilon} = \frac{\lambda + c\sqrt{\lambda}}{\lambda}; \quad \delta = \frac{e^{-\lambda}(e\lambda)^{\lambda + c\sqrt{\lambda}}}{(\lambda + c\sqrt{\lambda})^{\lambda + c\sqrt{\lambda}}}$$

□

B.2 Uneven Distribution Probabilities

Denote $IO^i_* = \vec{\gamma} = (\gamma_1, \ldots, \gamma_m)$. We use the Poisson mechanism to show that from observing IO^i_* for Alice's input chain i, the probability Alice goes to output chain o, $\Pr\left[A \to o \;\middle|\; IO^i_* = \vec{\gamma}\right]$, is close to the probability Alice goes to output chain o', $\Pr\left[A \to o' \;\middle|\; IO^i_* = \vec{\gamma}\right]$. Recall that each IO^i_x is covered by noise distributed according to $\text{Pois}(m\lambda_1 + m^2\lambda_2)$. Say $\lambda = m\lambda_1 + m^2\lambda_2$.

$$\begin{aligned}\Pr\left[IO^i_* = \vec{\gamma} \;\middle|\; A \to o\right] &= \text{Pois}(\lambda, \gamma_o - 1)\,\text{Pois}(\lambda, \gamma_{o'}) \prod_{k \neq o, o'} \text{Pois}(\lambda, \gamma_k) \\ &\leq e^{2\epsilon}\cdot \text{Pois}(\lambda, \gamma_o)\,\text{Pois}(\lambda, \gamma_{o'} - 1) \prod_{k \neq o, o'} \text{Pois}(\lambda, \gamma_k) + 2\delta \\ &\leq e^{2\epsilon}\cdot \Pr\left[IO^i_* = \vec{\gamma} \;\middle|\; A \to o'\right] + 2\delta\end{aligned}$$

Where ϵ, δ are the result of applying Thm 8.1 with distribution $\text{Pois}(m\lambda_1 + m^2\lambda_2)$. Call $(2\epsilon, 2\delta) = (\bar{\epsilon}, \bar{\delta})$.
We can use Bayes rule to say:

$$\Pr\left[A \to o \;\middle|\; IO^i_* = \vec{\gamma}\right] = \frac{\Pr\left[IO^i_* = \vec{\gamma} \;\middle|\; A \to o\right]\Pr[A \to o]}{\Pr\left[IO^i_*\right]}$$

Since $\Pr\left[IO^i_*\right]$ is just a constant and $\Pr[A \to o] = \Pr[A \to o'] = 1/m$, we can say the bounds translate over to:

$$\Pr\left[A \to o \;\middle|\; IO^i_* = \vec{\gamma}\right] \leq e^{\bar{\epsilon}}\cdot \Pr\left[A \to o' \;\middle|\; IO^i_* = \vec{\gamma}\right] + \bar{\delta}$$

B.3 Multidimensional Poisson Mechanism

Theorem B.2. *A mechanism which adds* $\text{Pois}(\lambda)$ *noise to each variable in output* $\vec{X}$*, where* $|\vec{X}| = m$*, is* (ϵ, δ)*-differentially private with respect to a uniform random change of 1 in* $\vec{X}$ *with* $\epsilon = \ln(1 + \frac{c\sqrt{m\lambda}}{m\lambda})$ *and* $\delta = \frac{e^{-m\lambda}(em\lambda)^r}{r^r}$*. (Where* $r = m\lambda + c\sqrt{m\lambda}$ *and* $c > 0$ *is a parameter that allows trading higher* ϵ *for lower* δ*.)*

Proof. For some vector of values, $X = (\alpha_1, \ldots, \alpha_m)$. Then we want to show:

$$\frac{1}{m}\sum_i \text{Pois}(\lambda, \alpha_i - 1)\prod_{j\neq i}\text{Pois}(\lambda, \alpha_j) \leq e^{\epsilon}\prod_j \text{Pois}(\lambda, \alpha_j) + \delta$$

We note that if each variable is covered with $\text{Pois}(\lambda)$, the sum of variables, $\sum_i \alpha_i$, is covered by $\text{Pois}(m\lambda)$. Assume:

$$\sum_i \alpha_i < m\lambda + c\sqrt{m\lambda}$$

$$\begin{aligned}
\frac{1}{m}\sum_i \text{Pois}(\lambda, \alpha_i - 1)\prod_{j\neq i}\text{Pois}(\lambda, \alpha_j) &= \frac{1}{m}\sum_i \frac{\lambda^{\alpha_i - 1}e^{-\lambda}}{(\alpha_i - 1)!}\prod_{j\neq i}\text{Pois}(\lambda, \alpha_j) \\
&= \frac{1}{m}\sum_i \frac{\alpha_i}{\lambda}\prod_j \text{Pois}(\lambda, \alpha_j) \\
&= \left(\frac{1}{m\lambda}\sum_i \alpha_i\right)\prod_j \text{Pois}(\lambda, \alpha_j) \\
&\leq \left(\frac{m\lambda + c\sqrt{m\lambda}}{m\lambda}\right)\prod_j \text{Pois}(\lambda, \alpha_j)
\end{aligned}$$

Similarly, with $m\lambda - c\sqrt{m\lambda} < \sum_i \alpha_i$, we get:

$$\begin{aligned}
\frac{\prod_j \text{Pois}(\lambda, \alpha_j)}{\frac{1}{m}\sum_i \text{Pois}(\lambda, \alpha_i - 1)\prod_{j\neq i}\text{Pois}(\lambda, \alpha_j)} &= \frac{\prod_j \text{Pois}(\lambda, \alpha_j)}{\left(\frac{1}{m\lambda}\sum_i \alpha_i\right)\prod_j \text{Pois}(\lambda, \alpha_j)} \\
&= \frac{m\lambda}{\sum_i \alpha_i} \\
&\leq \frac{m\lambda}{m\lambda - c\sqrt{m\lambda}}
\end{aligned}$$

Again, we let δ be the failure probability using the Poisson tail bounds for $\text{Pois}(m\lambda)$:

$$\begin{aligned}
\Pr\left[\sum_i \alpha_i \leq m\lambda - c\sqrt{m\lambda}\right] &\leq \frac{e^{-m\lambda}(em\lambda)^{m\lambda - c\sqrt{m\lambda}}}{(m\lambda - c\sqrt{m\lambda})^{m\lambda - c\sqrt{m\lambda}}} \\
\Pr\left[\sum_i \alpha_i \geq m\lambda + c\sqrt{m\lambda}\right] &\leq \frac{e^{-m\lambda}(em\lambda)^{m\lambda + c\sqrt{m\lambda}}}{(m\lambda + c\sqrt{m\lambda})^{m\lambda + c\sqrt{m\lambda}}}
\end{aligned}$$

□

B.3.1 More than one variable changes. We also need an analysis for where two random variables change (representing two single access variables changing).

$$\begin{aligned}\frac{\prod_k \text{Pois}(\lambda, \alpha_k)}{\frac{1}{m^2-m}\sum_{i,j}\text{Pois}(\lambda,\alpha_i-1)\,\text{Pois}(\lambda,\alpha_j-1)\prod_{k\neq i,j}\text{Pois}(\lambda,\alpha_k)} &= \frac{\prod_k \text{Pois}(\lambda, \alpha_k)}{\frac{1}{m^2-m}\sum_{i,j}\frac{\alpha_i\alpha_j}{\lambda^2}\prod_k \text{Pois}(\lambda,\alpha_k)} \\ &= \frac{(m^2-m)\lambda^2}{\sum_{i,j}\alpha_i\alpha_j} \\ &= \frac{(m^2-m)\lambda^2}{(m\lambda - c\sqrt{m\lambda})^2}\end{aligned}$$

This is less than the square of the single variable change, so the epsilons seem to compose linearly with the number of variables changing. The delta remains the same. Bounds hold when $i = j$.

B.4 Incorporating Uneven Distribution Probabilities

Instead of $1/m$ probability for each chain i, we have $\Pr[A \to i]$. We know that for all i, j:

$$\Pr[A \to i] \leq e^{\bar{\epsilon}} \cdot \Pr[A \to j] + \bar{\delta}$$

Since we know that the minimum probability is $< 1/m$, the naive way to bound this probability then is to replace

$$\Pr[A \to i] \leq \frac{e^{\bar{\epsilon}}}{m} + \bar{\delta}$$

However this is not a very tight bound, since we are essentially multiplying by this $\bar{\epsilon}$ which does not scale well with the number of servers m (depends on m^2).

We are using these probabilities in the form of a summation:

$$\sum_i \left(\Pr[A \to i] \times \text{Pois}(\lambda, \alpha_i - 1) \prod_{j \neq i} \text{Pois}(\lambda, \alpha_j) \right)$$

Since we know that these probabilities sum to 1, it seems wasteful to bound every single one by above. Instead, we'd like to consider what distribution of probabilities maximizes the privacy loss with respect to some constraints. Namely,

$$\sum_i \Pr[A \to i] = 1 \qquad \text{(probs sum to 1)}$$

$$\forall i, j \quad \Pr[A \to i] \leq e^{\bar{\epsilon}} \cdot \Pr[A \to j] + \bar{\delta} \qquad \text{(from uneven distribution)}$$

Unconstrained, we see that the most information is leaked when all the probability mass is on one variable. Thus, we show the most information is leaked in the constrained problem when the maximum probability mass possible is placed on one variable following the constraints. Or in other words, the probability vector is of minimum entropy.

$$(m-1)p_{\min} + e^{\bar{\epsilon}}p_{\min} + \bar{\delta} = 1$$

$$p_{\min} = \frac{1-\bar{\delta}}{e^{\bar{\epsilon}} + m - 1}$$

$$p_{\max} = e^{\bar{\epsilon}} \cdot \frac{1-\bar{\delta}}{e^{\bar{\epsilon}} + m - 1} + \bar{\delta}$$

Lemma B.3. *Given a probability vector $\vec{p}$, such that $\sum_{i=1}^m p_i = 1$ and $\forall p_i, p_i \geq p_{\min} = \frac{1-p_{\max}}{m-1}$ for some probability $p_{\max} > \frac{1}{m}$. Let $v_1, \ldots, v_m$ be values such that $v_j = \max\{v_i\}$. Then it holds that, $\sum_{i=1}^m p_i \cdot v_i \leq p_{\max} \cdot v_j + \sum_{i \neq j} p_{\min} \cdot v_i$.*

Proof. Since $p_i \geq p_{\min}$, for every probability p_i there exists $\epsilon_i \geq 0$ such that $p_i = p_{\min} + \epsilon_i$.

$$\begin{aligned}
\sum p_i \cdot v_i &= p_j \cdot v_j + \sum_{i \neq j} (p_{\min} + \epsilon_i) \cdot v_i \\
&= p_j \cdot v_j + \sum_{i \neq j} p_{\min} \cdot v_i + \sum_{i \neq j} \epsilon_i \cdot v_i \\
&\leq (p_j + \sum_{i \neq j} \epsilon_i) \cdot v_j + \sum_{i \neq j} p_{\min} \cdot v_i && (v_j = \max(\vec{v})) \\
&= p_{\max} \cdot v_j + \sum_{i \neq j} p_{\min} \cdot v_i && (p_{\max} = p_j + \sum_{i \neq j} \epsilon_i)
\end{aligned}$$

□

We can adjust the multidimensional analysis in the following way:

$$\frac{\sum_i p_i \operatorname{Pois}(\lambda, \alpha_i - 1) \prod_{k \neq i} \operatorname{Pois}(\lambda, \alpha_k)}{\prod_k \operatorname{Pois}(\lambda, \alpha_k)} \leq \frac{p_{\max} \operatorname{Pois}(\lambda, \alpha_j - 1) \prod_{k \neq j} \operatorname{Pois}(\lambda, \alpha_k) + \sum_{i \neq j} p_{\min} \operatorname{Pois}(\lambda, \alpha_i - 1) \prod_{k \neq i} \operatorname{Pois}(\lambda, \alpha_k)}{\prod_k \operatorname{Pois}(\lambda, \alpha_k)}$$

We can apply Thm 8.1 to the first term of the numerator and Thm 8.2 to the second term. Call the results of these theorem applications ϵ_1, δ_1 and ϵ_2, δ_2 respectively. We then get:

$$\begin{aligned}
\frac{\sum_i p_i \operatorname{Pois}(\lambda, \alpha_i - 1) \prod_{k \neq i} \operatorname{Pois}(\lambda, \alpha_k)}{\prod_k \operatorname{Pois}(\lambda, \alpha_k)} &\leq \frac{p_{\max} \operatorname{Pois}(\lambda, \alpha_j - 1) \prod_{k \neq j} \operatorname{Pois}(\lambda, \alpha_k) + \sum_{i \neq j} p_{\min} \operatorname{Pois}(\lambda, \alpha_i - 1) \prod_{k \neq i} \operatorname{Pois}(\lambda, \alpha_k)}{\prod_k \operatorname{Pois}(\lambda, \alpha_k)} \\
&\leq p_{\max} \cdot e^{\epsilon_1} + (m-1) p_{\min} \cdot e^{\epsilon_2}
\end{aligned}$$

The failure probabilities are composed similarly:

$$\delta = p_{\max} \cdot \delta_1 + (m-1) p_{\min} \cdot \delta_2$$

B.5 Adjacent Instances

Now we will show how to compose together the above analysis with respect to Stadium's observable variables. Recall the observable variables are IO^*_*, A_*, B_*. We already showed how the IO^*_* variables leak information about the which output chain users go to. We now use that result along with the multidimensional Poisson analysis to reason about the leakage of information from A_*, B_*. Denote $\vec{\alpha}, \vec{\beta}$ as the noise covering A_*, B_*.

B.5.1 Alice talking to Bob.

$$\begin{aligned}
\Pr\left[M(D) = \left(\vec{\alpha}, \vec{\beta}\right)\right] &= \sum_i \Pr\left[M(D) = \left(\vec{\alpha}, \vec{\beta}\right) \,\middle|\, A, B \to i\right] \cdot \Pr[A, B \to i] \\
&= \sum_i \Pr[A, B \to i] \left(\operatorname{Pois}(\lambda_2, \beta_i - 1) \prod_{k \neq i} \operatorname{Pois}(\lambda_2, \beta_k) \prod_k \operatorname{Pois}(\lambda_1, \alpha_k) \right)
\end{aligned}$$

Following the constraints:

$$\begin{aligned}
&\sum_i \Pr[A, B \to i] = 1 && \text{(probs sum to 1)} \\
&\forall i, j \quad \Pr[A, B \to i] \leq e^{2\bar{\epsilon}} \cdot \Pr[A, B \to j] + 2\bar{\delta} && \text{(from uneven distribution)}
\end{aligned}$$

B.5.2 Alice not talking.

$$\begin{aligned}
\Pr\left[M(D) = \left(\vec{\alpha}, \vec{\beta}\right)\right] &= \sum_{i,j} \Pr\left[M(D) = \left(\vec{\alpha}, \vec{\beta}\right) \,\middle|\, A \to i, B \to j\right] \cdot \Pr[A \to i] \Pr[B \to j] \\
&= \sum_{i,j} \Pr[A \to i] \Pr[B \to j] \left(\operatorname{Pois}(\lambda_1, \alpha_i - 1) \operatorname{Pois}(\lambda_1, \alpha_j - 1) \prod_{k \neq i,j} \operatorname{Pois}(\lambda_1, \alpha_k) \prod_k \operatorname{Pois}(\lambda_2, \beta_k) \right)
\end{aligned}$$

Following the constraints:

$$\sum_i \Pr[A \to i] = 1 \qquad \text{(probs sum to 1)}$$

$$\sum_i \Pr[B \to i] = 1 \qquad \text{(probs sum to 1)}$$

$$\sum_{i,j} \Pr[A \to i] \Pr[B \to j] = 1 \qquad \text{(implied from above)}$$

$$\forall i, j \quad \Pr[A \to i] \le e^{\bar{\epsilon}} \cdot \Pr[A \to j] + \bar{\delta} \qquad \text{(from uneven distribution)}$$

$$\forall i, j \quad \Pr[B \to i] \le e^{\bar{\epsilon}} \cdot \Pr[B \to j] + \bar{\delta} \qquad \text{(from uneven distribution)}$$

$$\forall i, j, k, l \quad \Pr[A \to i] \Pr[B \to j] \le e^{2\bar{\epsilon}} \cdot \Pr[A \to k] \Pr[B \to l] + 2\bar{\delta} \qquad \text{(implied from above)}$$

B.5.3 Comparison. Using the multidimensional Poisson analysis, we can compare the probability of each instance to:

$$\prod_k \text{Pois}(\lambda_1, \alpha_k) \text{Pois}(\lambda_2, \beta_k)$$

Consider Alice talking to Bob, and some ϵ_1, δ_1 from the multidimensional Poisson analysis:

$$\begin{aligned}
&\prod_k \text{Pois}(\lambda_1, \alpha_k) \sum_i \Pr[A, B \to i] \left(\text{Pois}(\lambda_2, \beta_i - 1) \prod_{k \neq i} \text{Pois}(\lambda_2, \beta_k) \right) \\
&\qquad \le \prod_k \text{Pois}(\lambda_1, \alpha_k) \cdot e^{\epsilon_1} \cdot \left(\prod_k \text{Pois}(\lambda_2, \beta_k) \right) + \delta_1 \\
&\qquad = e^{\epsilon_1} \cdot \left(\prod_k \text{Pois}(\lambda_1, \alpha_k) \text{Pois}(\lambda_2, \beta_k) \right) + \delta_1
\end{aligned}$$

Now consider Alice not talking, and some ϵ_2, δ_2 from the multidimensional Poisson analysis:

$$\begin{aligned}
&\prod_k \text{Pois}(\lambda_1, \alpha_k) \text{Pois}(\lambda_2, \beta_k) \\
&\qquad \le \prod_k \text{Pois}(\lambda_2, \beta_k) \cdot \left[e^{\epsilon_2} \left(\sum_{i,j} \Pr[A \to i] \Pr[B \to j] \text{Pois}(\lambda_1, \alpha_i - 1) \text{Pois}(\lambda_1, \alpha_j - 1) \prod_{k \neq i,j} \text{Pois}(\lambda_1, \alpha_k) \right) + \delta_2 \right] \\
&\qquad \le e^{\epsilon_2} \left(\sum_{i,j} \Pr[A \to i] \Pr[B \to j] \text{Pois}(\lambda_1, \alpha_i - 1) \text{Pois}(\lambda_1, \alpha_j - 1) \prod_{k \neq i,j} \text{Pois}(\lambda_1, \alpha_k) \prod_k \text{Pois}(\lambda_2, \beta_k) \right) + \delta_2
\end{aligned}$$

We can compose these two bounds to get:

$$\Pr[\text{ Alice talking to Bob }] \le e^{\epsilon_1 + \epsilon_2} \cdot \Pr[\text{ Alice not talking }] + \delta_1 + \delta_2$$

Composing these bounds does not follow trivially, but is due to our use of δ as a failure probabilitiy where assumptions do not hold. This allows us to compose the multiplicative ϵ factors in the normal fashion, and use a union bound on the failure probabilities, δ.

We can similarly show the privacy bounds for the other direction using the opposite bounds from the multinomial Poisson analysis.