Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy

Saba Eskandarian, *Stanford University;* Henry Corrigan-Gibbs, *MIT CSAIL;*
Matei Zaharia and Dan Boneh, *Stanford University*

https://www.usenix.org/conference/usenixsecurity21/presentation/eskandarian

This paper is included in the Proceedings of the
30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the
30th USENIX Security Symposium
is sponsored by USENIX.

Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy

Saba Eskandarian
Stanford University

Henry Corrigan-Gibbs
MIT CSAIL

Matei Zaharia
Stanford University

Dan Boneh
Stanford University

Abstract

Existing systems for metadata-hiding messaging that provide cryptographic privacy properties have either high communication costs, high computation costs, or both. In this paper, we introduce Express, a metadata-hiding communication system that significantly reduces both communication and computation costs. Express is a two-server system that provides cryptographic security against an arbitrary number of malicious clients and one malicious server. In terms of communication, Express only incurs a constant-factor overhead per message sent regardless of the number of users, whereas previous cryptographically-secure systems Pung and Riposte had communication costs proportional to roughly the square root of the number of users. In terms of computation, Express only uses symmetric key cryptographic primitives and makes both practical and asymptotic improvements on protocols employed by prior work. These improvements enable Express to increase message throughput, reduce latency, and consume over 100× less bandwidth than Pung and Riposte, dropping the end to end cost of running a realistic whistleblowing application by 6×.

1 Introduction

Secure messaging apps and TLS protect the confidentiality of data in transit. However, transport-layer encryption does little to protect sensitive communications *metadata*, which can include the time of a communications session, the identities of the communicating parties, and the amount of data exchanged. As a result, state-sponsored intelligence gathering and surveillance programs [21, 28, 36], particularly those targeted at journalists and dissidents [51, 57], continue to thrive – even in strong democracies like the United States [7, 8]. Anonymity systems such as Tor [31], or the whistleblowing tool SecureDrop [10, 55], attempt to hide communications metadata, but they are vulnerable to traffic analysis if an adversary controls certain key points in the network [30, 37, 38].

A host of systems can hide metadata with cryptographic security guarantees (e.g., Riposte [24], Talek [19], P3 [39], Pung [4], Riffle [44], Atom [43], XRD [45]). Unfortunately, these systems generally use heavy public-key cryptographic tools and incur high communication costs, making them difficult to deploy in practice. Another class of systems provides a differential privacy security guarantee (e.g., Vuvuzela [58], Alpenhorn [47], Stadium [56], Karaoke [46]). These systems offer high throughput and very low communication costs, but their security guarantees degrade with each round of communication, making them unsuitable for communication infrastructure that must operate over a long period of time.

This paper presents Express, a metadata-hiding communication system with cryptographic security that makes both practical and asymptotic improvements over prior work. Express is a two-server system that provides cryptographic security against an arbitrary number of malicious clients and up to one malicious server. This security guarantee falls between that of Riposte [24], which provides security against at most one malicious server out of three total, and Pung [4], which can provide security even in the single-server setting where the server is malicious. Express only uses lightweight symmetric cryptographic primitives and introduces new protocols which allow it to improve throughput, reduce latency, consume over 100× less bandwidth, and cost 6× less to operate compared to these prior works.

Express architecture. To receive messages via Express, a client registers *mailboxes* with the servers, who collectively maintain the contents of all the mailboxes. After registration, the mailbox owner distributes the *address* of a mailbox (i.e., a cryptographic identifier) to each communication peer via some out-of-band means. Given the address of a mailbox, any client can use Express to upload a message into that mailbox, without revealing to anyone except the mailbox owner which mailbox the client wrote into. Mailbox owners can fetch the contents of their mailboxes at any time with any frequency they wish, and *only* the owner of a mailbox can fetch its contents.

Crucially, Express hides which client wrote into which mailbox but *does not* hide which client *read* from which mailbox. This requires mailbox owners to check their mailboxes at a

fixed frequency, although there need not be any synchronization between the rates that different owners access their mailboxes. As we will discuss, this form of metadata privacy fits well with our main application: whistleblowing.

Technical overview. We now sketch the technical ideas behind the design of Express. As in prior work [24], Express servers hold a table of mailboxes secret-shared across two servers; clients use a cryptographic tool called a *distributed point function* [33] to write messages into a mailbox without the servers learning which mailbox a client wrote into [24, 49]. This basic approach to private writing leaves two important problems unsolved: handling read access to mailboxes and dealing with denial of service attacks from malicious users.

The first contribution of Express is to allow mailbox reads and writes to be asynchronous. This allows Express clients to contact the system with any frequency they like, regardless of other clients' behavior. In contrast, prior systems such as Riposte, Pung, and Vuvuzela [4, 24, 58] require *every* client to write before *any* client can read, so the whole system is forced to operate in synchronized rounds. We are able to allow read/write interleaving in Express with a careful combination of encryption and rerandomization. At a high level: any client in Express can read from any mailbox, but each read returns a fresh re-randomized encryption of the mailbox contents that only the mailbox owner can decrypt. In this way, even if an adversary reads the contents of all mailboxes between every pair of client writes, the adversary learns nothing about which honest client is communicating with which honest client.

The second major challenge for messaging systems based on secret sharing [17, 23–25, 27, 61] is to protect against malicious clients, who may corrupt the functioning of the system by submitting malformed messages. Since no server has a complete view of the message being written by each client, servers cannot immediately tell if a message is well-formed, e.g., whether it modifies only one mailbox or overwrites the contents of many mailboxes with garbage, destroying real messages that may have been placed in them. Express protects against such denial-of-service attacks using a new *auditing* protocol. In a system with n mailboxes, Express's auditing protocol requires only $O(\lambda)$ communication between parties, for a fixed security parameter λ, as well as $O(1)$ client side computation (in terms of AES evaluations and finite field operations). The analogous scheme in Riposte required $\Omega(\lambda\sqrt{n})$ communication and $\Omega(\sqrt{n})$ client computation [24], and additionally required a third non-colluding server. In practice, our new auditing scheme reduces overall computation costs for the client by 8× for a deployment with one million registered mailboxes.

In addition to defending against malformed messages aimed at corrupting the whole database of mailboxes, Express must protect against targeted attacks. A malicious client could potentially send a correctly-formed message containing random content to a single mailbox in hopes of overwriting any content written to that mailbox by an honest client. We defend against this by assigning *virtual addresses* to each mailbox. Each mailbox is accessed via a 128-bit virtual address, regardless of the actual number of mailboxes registered. The servers store and compute only over the number of actually registered mailboxes, not the number of virtual mailboxes. However, since virtual addresses are distributed at random over an exponentially large address space, a malicious client cannot write to a mailbox unless it knows the corresponding address. Section 4 describes our protections against malicious clients in detail.

Evaluation application. We evaluate Express as a system for whistleblowers to send messages to journalists while hiding their communications metadata from network surveillance. In this application, a journalist registers a mailbox for each source from which she wishes to receive information. The journalist then communicates her mailbox address to the source via, for example, a one-time in-person meeting. Thereafter, the source can privately send messages to the journalist by dropping them off in the journalist's Express mailbox. In this way, we can implement a cryptographically metadata-hiding variant of the SecureDrop system [10].

To provide whistleblowers with any reasonable guarantee of privacy, Express must provide its users with a degree of plausible deniability in the form of cover traffic. Otherwise, merely contacting the Express servers would automatically incriminate clients. As we will demonstrate, Express's low client computation and communication costs mean that an Express client implemented in JavaScript and embedded in a web page can generate copious cover traffic. Browsers that visit a cooperative news site's home page can opt-in to generate cover traffic for the system by running a JavaScript client in the backgound – thereby increasing the anonymity set enjoyed by clients using Express to whistleblow – without negatively impacting end-users' web browsing experience. We discuss this and other considerations involved in using Express for whistleblowing, e.g., how a journalist can communicate a mailbox address to a source, in Section 6.

We implement Express and evaluate its performance on message sizes of up to 32KB, larger than is used in the evaluations of Pung [4], Riposte [24] and Vuvuzela [58]. Recent high-profile whistleblowing events such as the whistleblower's report to the US intelligence community's inspector general [6] (25.3KB) or last year's anonymous New York Times op-ed [5] (9KB) demonstrate that messages of this length are very relevant to the whistleblowing scenario. We also compare Express's performance to Pung [4] and Riposte [24], finding that Express matches or exceeds their performance, and conclude that Express reduces the dollar cost of running a metadata-hiding whistleblowing service by 6× compared to prior work (see Figure 8). On the client side, Express's computation and communication cost are both *independent* of the number of users, at about 20ms client computation and 5KB communication overhead per message, enabling our new strategies for efficiently generating cover traffic. This represents over 100× bandwidth savings compared to Riposte [24] and over 7,000× savings compared to Pung for one million users. Although Vuvuzela

operates under a very different security model, we compare the two systems qualitatively in our full evaluation, which appears in Section 7.

In summary, we make the following contributions:

- The design and security analysis of Express, a metadata-hiding communication system that significantly reduces both communication and computation costs compared to prior work.
- A new auditing protocol to blindly detect malformed messages that is both asymptotically and practically more efficient than that of Riposte [24] while also removing the need for a third server to perform audits.
- An implementation and evaluation of Express that demonstrates the feasibility of our approach to metadata-hiding whistleblowing. Our open-source implementation of Express is available online at: `https://github.com/SabaEskandarian/Express`.

2 Design Goals

This section introduces the architecture of Express and describes our security goals.

An Express deployment consists of two servers that collectively maintain a set of *locked mailboxes*. Each locked mailbox implements a private channel through which one client can send messages to another who has the secret cryptographic key to unlock that mailbox.

To use Express, a client wishing to receive messages first registers a mailbox and gets a *mailbox address*. From then on, any client who has been given the mailbox address can write messages to that mailbox, and the owner of that mailbox can check the mailbox for messages whenever it wants. We discuss how clients can communicate mailbox addresses to each other via a *dialing* protocol in Section 6.2.

We consider an attacker who controls one of the two Express servers, any number of Express clients, and the entire network. The main security property we demand is that, after an honest client *writes* a message into a mailbox, the attacker learns nothing about which mailbox the client wrote into. This corresponds to an anonymity guarantee where the sender of a given message cannot be distinguished among the set of all senders in a given time interval. We also require that an attacker who controls any number of malicious clients cannot prevent honest clients from communicating with each other. In other words, we protect against malicious clients mounting in-protocol denial-of-service attacks. We do not aim to protect against DoS attacks by malicous servers, nor against network-level DoS attacks, but we will describe how clients can incorporate straightforward checks to detect tampering by malicious servers.

2.1 Express API

Express allows clients to register mailboxes, read the contents of mailboxes they register, and privately write to others' mailboxes. Clients interact with the servers via the following operations:

Mailbox registration. A client registers a new mailbox by sending the Express servers distinct *mailbox keys*. The servers respond with a *mailbox address*. We say that a client "owns" a given mailbox if it holds the mailbox's keys and address.

Mailbox read. To read from a mailbox, the client sends the mailbox's address to the Express servers. The servers respond with the locked (i.e., encrypted) mailbox contents, which the client can decrypt using its two mailbox keys together.

Mailbox write. To write to a mailbox, a client sends a specially-encoded *write request* to the Express servers that contains an encoding of both the address of the destination mailbox and the message to write into it. No single Express server can learn either the destination address or message from the write request.

2.2 Security Goals

Based on the demands of our application to whistleblowing, Express primarily aims to provide privacy guarantees for *writes* and not for reads. For example, Express hides who whistleblowers send messages to, but it does not hide the fact that journalists check their mailboxes. Below we describe Express's core security properties, which we formalize when proving security in Appendix A.

Metadata-hiding. We wish to hide who a given client is writing to from everyone except the recipient of that client's messages. To this end, our *metadata-hiding* security guarantee requires that for each write into an Express mailbox, no adversary who controls arbitrarily many clients and one server can determine which mailbox that write targeted unless the adversary owns the target mailbox.

We formalize this requirement in Appendix A, where we show that an adversary can *simulate* its view of honest clients' requests before seeing them, which proves that the adversary learns nothing from requests that it can't generate on its own, except necessary information such as the time the write occurred and which client initiated it. In particular, this means the adversary does not learn the mailbox into which a request writes, although it does learn that a write has occurred. A malicious server can stop responding to requests or corrupt the contents of users' mailboxes, but we require that even an actively malicious server cannot break our metadata-hiding property.

Soundness. Express must be resilient to malformed messages sent by malicious clients. This means no client can write to a mailbox it has not been authorized to access, even if it deviates arbitrarily from our protocol. We capture this requirement via a *soundness game* in Appendix A, where we also prove that no adversary can win the soundness game in Express with greater than negligible probability in a security parameter.

2.3 Design Approaches

As there are many potential approaches to metadata-hiding systems, we now briefly sketch high-level decisions made regarding the goals of Express.

Deployment scenario. Express's primary deployment scenario is as a system for whistleblowing, where a source leaks a document or tip to a journalist. In this setting, unlike prior work, Express does not require the system to run in synchronous rounds. This is the deployment scenario on which we will focus the exposition of the Express system. However, since this is a one-directional communication setting (the source can send leaks to the journalist but not have an ongoing conversation), Express can also be used as a standard messaging protocol where clients, e.g., sources and journalists, send messages back and forth to each other. In this setting, similar to prior work, messaging in Express would progress in time epochs, with a server-determined duration for each round.

Differential vs cryptographic privacy. Express belongs to a family of systems that provide cryptographic security guarantees. In contrast, a number of systems (e.g., Vuvuzela, Stadium, Karaoke [46, 56, 58]) provide differentially private security. The difference between the two types of systems lies in the amount of private metadata the systems leak to an adversary. Cryptographic security means that no information leaks – the adversary learns nothing, even after observing many rounds of communication, about which clients are communicating with each other. In contrast, systems providing the differential privacy notion of security allow some quantifiable leakage of metadata. Thus, with differential privacy-based systems, an attacker can – after a number of communication rounds – learn who is communicating. In contrast, the security of Express does not degrade, even after many rounds of interaction. Thus, although differentially private systems offer faster performance, cryptographic security is preferable for frequently used privacy-critical applications.

Distributing trust. There are two potential approaches to deployment of metadata-hiding systems. One approach envisions a grass-roots deployment model where large numbers of people or organizations decide to participate to run the system, and trust is distributed among the servers with tolerance for some fraction behaving maliciously. The approach taken by Express (and the works to which we primarily compare it [4, 24]) envisions a commercial infrastructure setting where only a small number of participants (e.g., for our example use case, the Wall Street Journal and the Washington Post) are needed to deploy the system with its full security guarantees. Given equal performance and security against an equal fraction of malicious servers, it is of course preferable to distribute trust over a larger number of parties. Thus designs that split trust between a small number of parties can be seen as one point on a tradeoff between having many parties that undergo some light vetting versus having few parties that undergo heavier vetting before being included as servers in the system.

2.4 Limitations

We now discuss some limitations of Express to aid in determining which scenarios are best-suited to an Express deployment.

The most important limitation to consider when deciding whether to deploy Express is the issue of censorship. As mentioned above, Express relies on distributing trust among two servers. Thus, if traffic to either server is blocked, the system can no longer be accessed. Since we envision Express being deployed by major news organizations, Express would not be appropriate for use in countries with a history of blocking traffic to such organizations. This is true of any system that distributes trust over a small number of servers (or has easily identifiable traffic). However, there is a need to prevent surveillance even in countries with relatively open access to the internet. It is in this setting that Express can be an effective approach to metadata-hiding communication.

Express allows mailbox owners to access their mailboxes and retrieve messages with whatever frequency they desire when being used for one-way communication, but they must check mailboxes at regular intervals in order to maintain security because Express does not hide which mailbox a given read accesses. If a mailbox owner changes her mailbox-checking pattern based on the contents of messages received, this may leak something about who is sending her messages. Note that although this implies that mailbox owners should regularly check their mailboxes, it does not impose any restrictions on the frequency with which any owner checks her mailboxes – it is not a fixed frequency required by the system and can be different for each mailbox owner. This is in contrast with prior works, which fix a system-wide frequency with which clients must contact the servers or require clients to always remain online. Clients sending messages through Express but not also receiving messages (e.g., whistleblowers sending tips or documents) do not need to regularly contact the system.

Another reason for mailbox owners to check their mailboxes regularly is that messages in Express are written into mailboxes by adding, not concatenating, the message contents to the previous contents of the mailbox. It is thus possible for a second message sent to the same mailbox to overwrite the original contents, causing the content to be clobbered when someone eventually reads it. This risk can be easily mitigated, however, because each mailbox is for one client to send messages to one other client, and servers zero-out the contents of mailboxes after they are read to make space for new messages. Looking ahead to our application, messages can be a leak of a single document, where more than one message is not required. If a journalist expects to receive many messages from the same source before she has a chance to read and empty the contents of a mailbox, one way to handle this situation is to register several mailboxes for the same source, so each message can be sent to a different mailbox. This way, as long as a journalist checks and empties her mailboxes before they have all been used, no messages will be overwritten.

While Express's soundness property prevents in-protocol denial of service attacks by malicious clients, a malicious Express server can launch a denial of service attack by overwriting mailboxes with garbage. This attack will prevent communication

through Express, but it can at least be detected. We discuss how clients can add integrity checks to their messages to achieve authenticated encryption over Express in Section 5. This means that a client receiving a garbage message will know that the message has been corrupted by a malicious server.

Finally, like all systems providing powerful metadata-hiding guarantees, Express must make use of cover traffic to hide information about which users are really communicating via Express. Although necessary, cover traffic allows metadata-hiding systems to protect even against adversaries with strong background knowledge about who might be communicating with whom by providing plausible deniability to clients sending messages through Express. We further discuss cover traffic in Section 6.1.

3 Express Architecture

This section describes the basic architecture of Express. Section 4 shows how to add defenses to protect against disruptive clients, and Section 5 states the full Express protocol. Section 6 discusses how to use Express for whistleblowing, including how a mailbox owner communicates a mailbox address to senders and how to increase the number of Express users by deploying it on the web.

The starting point for Express is a technique for privately writing into mailboxes using distributed point functions [24, 33, 49]. We review how DPFs can be used for private writing in Section 3.1. A private writing mechanism alone, however, does not suffice to allow metadata-hiding communication. We must also have a mechanism to handle access control so that only the mailbox owner can access the contents of a given mailbox. We discuss a lightweight cryptographic access control system in Section 3.2, where we also explain how this combination of private writing and controlled reading enables metadata hiding without synchronized rounds.

3.1 Review: Private Writing with DPFs

We briefly review the technique used in Riposte [24] for allowing a client to privately write into a database, stored in secret-shared form, at a set of servers.

A naïve approach. In Express, two servers – servers A and B – collectively hold the contents of a set of *mailboxes*. In particular, if there are n mailboxes in the system and each mailbox holds an element of a finite field $\mathbb{F}$, then we can write the contents of all mailboxes in the system as a vector $D \in \mathbb{F}^n$. Each server holds an additive secret share of the vector D: that is, server A holds a vector $D_A \in \mathbb{F}^n$ and server B holds a vector $D_B \in \mathbb{F}^n$ such that $D = D_A + D_B \in \mathbb{F}^n$.

Once a client registers a mailbox, another client with that mailbox's address can send messages or documents to the mailbox, which the mailbox owner can check at his or her convenience. Although Express can support mailboxes of different sizes, size information can be used to trace a message from its sender to its receiver, so Express clients must pad messages, either all to the same size or to one of a few pre-set size options.

To write a message $m \in \mathbb{F}$ into the i-th mailbox naïvely, the Express client could prepare a vector $m \cdot \mathbf{e}_i \in \mathbb{F}^n$, where $\mathbf{e}_i$ is the ith standard-basis vector (i.e., the all-zeros vector in $\mathbb{F}^n$ with a one in coordinate i). The client would then split this vector into two additive shares w_A and w_B such that $w_A + w_B = m \cdot \mathbf{e}_i$, and send one of each of these "write-request" vectors to each of the two servers. The servers then process the write by setting:

$$D_A \leftarrow D_A + w_A \in \mathbb{F}^n \qquad D_B \leftarrow D_B + w_B \in \mathbb{F}^n,$$

which has the effect of adding the value $m \in \mathbb{F}$ into the contents of the ith mailbox in the system.

The communication cost of this naïve approach is large: updating a single mailbox requires the client to send n field elements to each server.

Improving efficiency via DPFs. Instead of sending such a large message, the client uses *distributed point functions* (DPFs) [14, 15, 33] to compress these vectors. DPFs allow a client to split a point function f, in this case a function mapping indices in the client's vector to their respective values, into two *function shares* f_A and f_B which individually reveal nothing about f, but whose sum at any point is the corresponding value of f. More formally, let $f_{i^*,m} \colon [N] \to \mathbb{F}$ be a point function that evaluates to 0 at every point $i \in [N]$ except that $f(i^*) = m \in \mathbb{F}$. A DPF allows a client holding $f_{i^*,m}$ to generate shares f_A and $f_B : [N] \to \mathbb{F}$ such that:

(i) an attacker who sees only one of the two shares learns nothing about i^* or m, and

(ii) for all $i \in [N]$, $f_{i^*,m}(i) = f_A(i) + f_B(i) \in \mathbb{F}$.

Moreover, in addition to supporting messages $m \in \mathbb{F}$, the latest generation of DPFs [15] allow for any message $m \in \{0,1\}^*$. When using these DPFs with security parameter λ, each function share (f_A and f_B) has bitlength $O(\lambda \log N + |m|)$. In addition to general improvements in efficiency over prior DPFs, our choice of DPF scheme will enable new techniques that we introduce in Section 4.

In essence, the client can use DPFs to *compress* the vectors w_A and w_B, which reduces the communication cost to $O(\lambda \cdot \log N + \log|\mathbb{F}|)$ bits, when instantiated with a pseudorandom function [35] using λ-bit keys. Upon receiving f_A and f_B the servers can evaluate them at each point $i \in [n]$ to recover the vectors w_A and w_B and update D_A and D_B as before.

3.2 Hiding Metadata without Synchronized Rounds

Private writing alone does not suffice to provide metadata-hiding privacy. In order to achieve this, we also need to control *read* access to mailboxes. Otherwise, a network adversary who controls a single client could read the contents of all mailboxes between each pair of writes and learn which client's message modified which mailbox contents, even if messages are encrypted. Prior works such as Pung [4] or Riposte [24] prevent this attack by operating in batched rounds in which

many clients write messages before any client is allowed to read. The key feature that allows Express to hide metadata without relying on synchronized rounds is that a message can only be read by the mailbox owner to whom it is sent. Express can make messages available to mailbox owners immediately as long as (1) the messages remain inaccessible to an attacker who does not own the mailbox whose contents have been modified and (2) the attacker cannot tell which mailbox has been modified if it does not own the modified mailbox. Thus, all we need to successfully hide metadata without rounds is a mechanism for access control that satisfies these two requirements. While an adversary who continuously reads from all mailboxes could then still learn *when* a write occurs, it would learn nothing about which mailbox contents were modified as a result.

Express includes a lightweight cryptographic approach to access control that relies on symmetric encryption, does not require the servers to undertake any user authentication logic when serving read requests, and enables useful implementation optimizations. A client registering a mailbox uploads keys k_A and k_B to servers A and B respectively, and the servers encrypt stored data using the respective key for each mailbox, decrypting before making modifications and re-encrypting after. The re-encryption ensures that the contents of every mailbox are rerandomized after each write, so an attacker attempting to read the contents of a mailbox for which it does not have both keys learns nothing from reading the encrypted contents of the mailbox, including whether or not those contents have changed. This property still holds even if only one of the two servers carries out the re-encryption, so its security is unaffected if a malicious server does not encrypt or re-encrypt mailboxes. Our implementation encrypts mailbox contents in counter mode, so re-encryption simply involves subtracting the encryption of the previous count and adding in the new one. Since these operations are commutative, we can implement an optimization where re-encryption is not done on every write but only when a read occurs after one or more writes. This makes our approach – which requires only symmetric encryption – more efficient than a straightforward one based on public key encryption, e.g., where the contents of each mailbox are encrypted under the owner's public key when a read is requested.

4 Protecting Against Malicious Clients

The techniques in Section 3 suffice to provide privacy if all clients behave honestly, but they are vulnerable to disruption by a malicious client. In the scheme described thus far, one malicious client can corrupt the state of the two servers with a single message. To do so, the malicious client sends DPF shares f_A and f_B to the servers that expand into vectors w_A and w_B such that $w_A + w_B = v \in \mathbb{F}^n$, where v is non-zero at many (or even all) coordinates. A client who submits such DPF key shares can, with one message to the servers, write into *every* mailbox in the system, corrupting whatever actual messages each mailbox may have held.

Express protects against this attack with an *auditing* protocol that checks to make sure $(w_A + w_B) \in \mathbb{F}^n$ is a vector with at most one non-zero component. In other words, the servers check that each write request updates only a single mailbox. Any write request that fails this check can be discarded to prevent it form corrupting the contents of D_A and D_B. Riposte [24], a prior work that also audits DPFs to protect against malicious clients, uses a three-server auditing protocol that requires communication $\Omega(\lambda\sqrt{n})$ and client computation $\Omega(\sqrt{n})$ for a system with n mailboxes, where λ is a security parameter. However, their protocol takes advantage of the structure of a particular DPF construction that is less efficient than the one used by Express. Applying their protocol to the more efficient DPFs used in Express would require client communication and computation $\Omega(\lambda n)$ and $\Omega(n)$ respectively as well as the introduction of an additional non-colluding server. This linear bandwidth consumption *per write* would create a communication bottleneck in Express and increase client-side computation costs significantly. Moreover, adding a third server – and requiring that two out of three servers remain honest to guarantee security – would dramatically reduce the practicality of the Express system. To resolve this issue, we introduce a new auditing protocol that drops client computation (in terms of AES evaluations and finite field operations) to $O(1)$ and communication to $O(\lambda)$ while simultaneously *eliminating the need for a third server* to perform audits. We describe our two-party auditing protocol in Section 4.1.

Although auditing ensures that DPFs sent by clients must be well-formed, an attacker targeting Express has a second avenue to disrupting the system. Instead of attempting to corrupt the entire set of mailboxes – an attack prevented by the auditing protocol – a malicious client can write random data to *only one mailbox* and corrupt any message a source may send to a journalist over that mailbox. Although this attack is easily detectable when a journalist receives a random message, it still allows for easy disruption of the system and cannot be blocked by blind auditing because the disruptive message is structured as a legitimate write.

We defend against this kind of targeted disruption with a new application of *virtual addressing*. At a high level, we assign each mailbox a unique 128-bit virtual address and modify the system to ensure that writing into a mailbox requires knowing the mailbox's virtual address. In this way, a malicious user cannot corrupt the contents of an honest user's mailbox, since the malicious user will not be able to guess the honest user's virtual address. We discuss this defense and its implications for other components of the system in Section 4.2.

4.1 Auditing to Prevent Disruption

This section describes our auditing protocol. We begin with a rough outline of the protocol before stating the security properties required of it and then explaining the protocol in full detail. At a high level, our auditing protocol combines the verifiable DPF protocol of Boyle et al. [15], which only provides security

against *semi-honest* servers, with *secret-shared non-interactive proofs* (SNIPs) first introduced by the Prio system [23] (and later improved and generalized by Boneh et al. [11]) to achieve security against *fully malicious* servers. We explain each of these ideas and how we combine them below.

Let the vectors w_A and $w_B \in \mathbb{F}^n$ be the outputs that servers A and B recover after evaluating $f_A(i)$, $f_B(i)$, for $i \in [n]$. Note that even DPFs that output a message in $\{0,1\}^*$ begin with an element of a λ-bit field $\mathbb{F}$ and expand it, so for the purposes of our auditing protocol, we can assume that every DPF output is an element of $\mathbb{F}$. We say that $w = w_A + w_B \in \mathbb{F}^n$ is a *valid* write-request vector if it is a vector in $\mathbb{F}^n$ of Hamming-weight at most one. The goal of the auditing protocol is to determine whether a given write-request vector is valid.

The observation of Boyle et al. [15] is that the following n-variate polynomial equals zero with high probability over the random choices of $r_1, ..., r_n$ if and only if (1) there is at most one nonzero w_i and (2) $m = w_i$ for the nonzero value of w_i

$$f(r_1, ..., r_n) = (\Sigma_{i \in [n]} w_i r_i)^2 - m \cdot (\Sigma_{i \in [n]} w_i r_i^2).$$

This polynomial roughly corresponds to taking a random linear combination of the elements of w – using randomness shared between the two servers – and checking that the square of the linear combination and the sum of the terms of the linear combination squared are the same. Using the fact that it is easy to compute linear functions on secret-shared data, the two sums in the equation above can be computed non-interactively by servers A and B. Boyle et al. suggest using a multiparty computation between the servers to compute the remaining multiplications and check whether this polynomial in fact equals zero, thus determining whether the DPF is valid.

The problem with this approach is that it is only secure against *semi-honest* servers. A malicious server can deviate from the protocol and potentially learn which entry of w is non-zero. For example, suppose a malicious server A is interested in knowing whether a write request modifies an index i^*. It runs the auditing protocol as described, but it replaces its value w_{Ai^*} with a randomly chosen value w'_{Ai^*}. If $w_{Ai^*} + w_{Bi^*} = 0$, i.e., i^* was not the nonzero index of w, this modification will cause the audit to fail because the vector w' that includes w'_{Ai^*} instead of w_{Ai^*} no longer has hamming weight one. Thus the malicious server learns that the write request would not have modified index i^*. On the other hand, if $w_{Ai^*} + w_{Bi^*} \neq 0$, i.e., i^* was the nonzero index of w, the inclusion of w'_{Ai^*} still results in a vector w' of hamming weight one, and the auditing protocol passes. Thus the malicious server can detect whether or not the write request modifies index i^* by observing whether or not auditing was successful after it tampers with its inputs.

To prevent this attack we make use of a SNIP proof system [11, 23]. In a SNIP, a client sends each server a share of an input w and an arithmetic circuit $\mathsf{Verify}()$. The client then uses a SNIP proof to convince the servers, who only hold shares of w but may communicate with each other, that $\mathsf{Verify}(w) = 1$. An important property of a SNIP proof system is that it provides security against *malicious* servers. That is, even a server who deviates from the protocol cannot abuse a SNIP to learn more about w. SNIP proofs require computation and communication linear in the number of multiplications between secret values in the statement being proved. Our approach is to instantiate the DPF verification protocol of Boyle et al. [15] *inside* of a SNIP to protect it from potentially malicious servers. Since the Boyle et al. verification protocol only requires two multiplications between shared values, the squaring and the multiplication by m, this results in a constant-sized SNIP (i.e. size $O(\lambda)$).

Properties of auditing protocol. Before describing our protocol in detail, we recall the completeness, soundness, and zero-knowledge properties we require of the auditing protocol (adapted from those of Riposte's auditing protocol [24]).

- **Completeness.** If all parties are honest, the audit always accepts.
- **Soundness against malicious clients.** If w is not a valid write request (i.e., the client is malicious) and both servers are honest, then the audit will reject with overwhelming probability.
- **Zero knowledge against malicious server.** Informally: as long as the client is honest, an active attacker controlling at most one server learns nothing about the write request w, apart from the fact that it is valid. That is, for any malicious server there exists an efficient algorithm that simulates the view of the protocol execution with an honest second server and an honest client. The simulator takes as input only the public system parameters and the identity of the malicious server.

Our auditing protocol. Our auditing protocol proceeds as follows. We assume that data servers A and B share a private stream of random bits generated from a pseudorandom generator with a seed r. In practice, the servers generate the random seed by agreeing on a shared secret at setup and using a pseudorandom generator to get a new seed for each execution of this protocol. We will describe the protocol using a SNIP as a black box and give details on how to instantiate the SNIP in Appendix B.

At the start of the protocol, server A holds r and $w_A \in \mathbb{F}^n$ and server B holds r and $w_B \in \mathbb{F}^n$, both generated by evaluating the DPF shares sent by the client at each registered mailbox address. The client holds the index i^* at which w is non-zero as well as the values of w_A and w_B at index i^*, which it computes from the function shares f_A and f_B that it sent to the servers.

1. *Servers derive proof inputs.*
   The servers begin by sending the random seed r used to generate their shared randomness to the client.

   Next, they compute shares m_A and m_B of m, the value of w at its non-zero entry, which is simply the sum of all the elements of w_A or w_B respectively because all but one entry of w should be zero. That is, the servers compute

$$m_A \leftarrow \Sigma_{i \in [n]} w_{Ai} \quad \text{and} \quad m_B \leftarrow \Sigma_{i \in [n]} w_{Bi}.$$

Then servers A and B use their shared randomness r to generate a random vector $r = (r_1, ..., r_n) \in \mathbb{F}^n$ and then compute the vector of squares $R = (r_1^2, ..., r_n^2) \in \mathbb{F}^n$. After this, they compute shares of the "check" values $c = \langle w, r \rangle$ and $C = \langle w, R \rangle$:

$$c_A \leftarrow \langle w_A, r \rangle \in \mathbb{F}, \quad C_A \leftarrow \langle w_A, R \rangle \in \mathbb{F}$$
$$c_B \leftarrow \langle w_B, r \rangle \in \mathbb{F}, \quad C_B \leftarrow \langle w_B, R \rangle \in \mathbb{F}$$

Here the notation $\langle x, y \rangle$ represents the inner product between vectors $x, y \in \mathbb{F}^n$, defined as $\Sigma_{i=1}^n x_i y_i$.

At this point, the servers hold values m_A, c_A, C_A and m_B, c_B, C_B respectively.

2. *Client derives proof inputs.*
Since the client knows the seed r, the index i^*, and the values of w_A and w_B at index i^* (and as a consequence the value of $m = w_{Ai^*} + w_{Bi^*} \in \mathbb{F}$), the client can compute the random values r^*, r^{*2} that will be multiplied by the i^*th entries of w_A and w_B. Since all the values other than the i^*th entry of w are zero, the client need not compute them. Thus the client computes the check values $c^* = r^* \cdot (w_{Ai^*} + \cdot w_{Bi^*})$ and $C^* = r^{*2} \cdot (w_{Ai^*} + \cdot w_{Bi^*})$. Note that this allows the client to compute the check values in only $O(1)$ time even though the servers must do $O(n)$ work to find them.

3. *Proof computation and verification.*
To complete the proof, the client prepares a SNIP proof $\pi = (\pi_A, \pi_B)$, sends π_A to server A, and sends π_B to server B. The servers then verify the proof, communicating with each other as needed. The SNIP proves that

$$c^2 - m \cdot C = 0$$

where $c \leftarrow c_A + c_B$ and $C \leftarrow C_A + C_B$.

The soundness property of the SNIP proof guarantees that the servers will only accept the proof if the statement is true, and the zero-knowledge property of the proof guarantees that as long as one server is honest, the servers learn nothing from receiving the SNIP proof that they did not know before receiving it (even if one server is fully malicious). Note that this statement only involves two multiplications: $c \cdot c$ and $m \cdot C$.

We sketch the instantiation of the proofs used in our auditing protocol as well as the security analysis of the full auditing protocol in Appendix B. Full details and a security proof for the SNIP proof system itself can be found in the Prio paper [23] and the follow-up work of Boneh et al. [11].

4.2 Preventing Targeted Disruption

We now describe how Express prevents a targeted attack where a malicious client writes random data to a single mailbox to corrupt its contents. Express servers assign each mailbox a 128-bit *virtual address* and ensure that a client can only write to a mailbox if it knows the corresponding virtual address.

To implement this, the Express servers maintain an array of n *physical* mailboxes, but they also maintain an array of 2^λ *virtual* mailboxes, where $\lambda \approx 128$ is a security parameter. The two data servers assign a unique virtual address to each physical mailbox, and they collectively maintain a mapping – a *page table* – that maps each active virtual address to a physical mailbox. Since the virtual addressing scheme's only goal is to prevent misbehavior by malicious clients, the servers both hold the contents of the page table (i.e., the list of active virtual addresses and their mapping to physical addresses) in the clear. The virtual-address space (around 2^{128} entries) is vastly larger than the number of physical mailboxes (around 2^{20}, perhaps), so the vast majority of the virtual-address space goes unused.

When a client registers a new mailbox, the servers both allocate storage for a new physical mailbox, assign a new random virtual address to this physical mailbox, and update their page tables. The address can either be chosen by one server and sent to the other or generated separately by each server using shared randomness. The servers then return the virtual and physical addresses for the mailbox to the client. As mentioned above, a mailbox owner must communicate its address to others to receive messages. We describe how this can be achieved when we discuss dialing in Section 6.2. The contents of the tables stored at the servers are shown in Figure 1.

When preparing a write request, the client prepares DPF shares f_A and $f_B : 2^\lambda \to \mathbb{F}$ as if it were going to write in to the exponentially large address space. However, instead of evaluating shares at every $i \in [2^\lambda]$, the Express servers only evaluate f_A and f_B at the currently active virtual addresses. In this way, the number of DPF evaluations the servers compute remains linear in the number of registered mailboxes, even though clients send write requests as if the address space were exponentially large. A client who does not know the address for a given mailbox has a chance negligible in λ of guessing the correct virtual address. Note that this technique is only possible because Express uses a DPF whose share sizes are *logarithmic* in the function domain size. Using virtual addresses with older square-root DPFs would result in infeasibly large message sizes and computation costs.

Although virtual addressing, when combined with auditing, does fully resolve the issue of disruptive writes, it does not fully abstract away physical addresses. Our auditing protocol critically relies on the client knowing the index of the mailbox it wants to write to among the set of all mailboxes. As such, a client preparing to send a message must be informed of both the virtual and physical addresses of the mailbox it wishes to write to. Fortunately, the size of a physical address is much smaller than that of a virtual address (about 20 bits compared to 128 bits for a virtual address), so communicating both addresses at once adds little cost to only sending the virtual address.

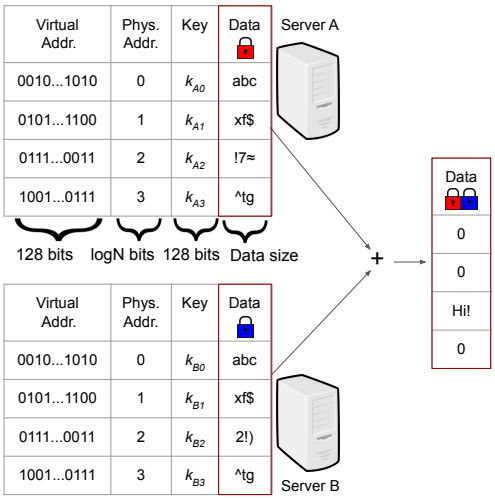

Figure 1: Contents of the tables held by servers in Express. Each server stores the conversion from virtual to physical addresses and a distinct key for each mailbox. Combining data from the two servers allows a user holding both keys for a given mailbox to read its contents.

5 Full Express Protocol

This section summarizes the full Express protocol described incrementally in Sections 3 and 4. We will describe the protocol in full but refer to the steps of the auditing protocol as described in Section 4.1 to avoid repeating the protocol spelled out in detail there. We prove security in Appendix A. After describing the protocol, we describe how clients can add message integrity to their Express messages.

We assume that a mailbox owner has already set up a mailbox with virtual address v and physical address p and communicated (p,v) to another client. We discuss options for communicating p and v to other clients ("dialing") in Section 6.2. We also assume that the mailbox owner holds mailbox keys k_A and k_B, which it has sent to servers A and B respectively, and the client has a message m that it wants to send. Server A holds vectors V of virtual addresses, K_A of keys, and D_A of mailbox contents, each of length n. Server B likewise holds V, K_B and D_B. Each entry of D_A and D_B is encrypted in counter mode under the corresponding key in K_A or K_B. Figure 1 shows the information held by servers A and B for each mailbox.

Sending a message.

1. The client generates DPF shares f_A and f_B of the point function $f_{v,m} : [2^\lambda] \to \{0,1\}^{|m|}$. It sends f_A to A and f_B to B.

2. A and B evaluate $w_A \leftarrow (f_A(V_1),...,f_A(V_n))$ and $w_B \leftarrow (f_B(V_1),...,f_B(V_n))$. They use their shared randomness to generate a seed r to be used in the auditing protocol, send it to the client, and prepare the server inputs to the SNIP.

3. The client prepares the client inputs to the SNIP and generates the corresponding proof $\pi = (\pi_A, \pi_B)$. It sends π_A to server A and π_B to server B.

4. The servers verify the SNIP proof π, and they abort if the verification fails.

5. Servers A and B decrypt each D_{Ai} with K_{Ai} and each D_{Bi} with key $K_{Bi}, i \in [n]$. Next, they set $D_{Ai} \leftarrow D_{Ai} + w_{Ai}$ and $D_{Bi} \leftarrow D_{Bi} + w_{Bi}$ before re-encrypting the new values of D_{Ai} and D_{Bi} under the same keys (with new nonces).

Checking a mailbox.

1. The mailbox owner sends (p,v) to servers A and B to request to read from the mailbox at physical address p.

2. Servers A and B check that virtual address v corresponds to physical address p and then send D_{Ap} and D_{Bp} as well as the nonce used for the encryption of each value. Then they set the values of D_{Ap} and D_{Bp} to fresh encryptions of 0 under K_{Ap} and K_{Bp} respectively, emptying the mailbox. Since only the mailbox owner and whoever wrote into a mailbox know p and v, and the virtual address space for v is huge, clients cannot read or delete the contents of each other's mailboxes.

3. The mailbox owner decrypts the values of D_{Ap} and D_{Bp} it received with keys k_A and k_B to get messages m_{Ap} and m_{Bp}. It outputs message $m \leftarrow m_{Ap} + m_{Bp}$.

Complexity. Table 2 shows the communication and computational complexity of sending a message in Express for the client and the servers. We measure computational complexity in terms of AES evaluations and field operations separately to better capture the computation being carried out by each party. The complexities reported are the sum of costs due to DPF evaluation, re-encryption, and auditing.

Client communication includes sending a DPF whose shares are functions with domain size 2^λ, resulting in DPFs of size $O(\lambda^2 + |m|)$. As discussed in Section 4.1, the auditing protocol involves the client sending a proof of size $O(\lambda)$.

Cryptographic costs on the client include generating DPF shares and evaluating the DPF at one point, both of which cost $O(\lambda + |m|)$. The server, on the other hand, must evaluate the DPF at each address and also generate the random vectors needed for the auditing protocol. The number of field operations for each party come directly from the costs incurred during the auditing protocol.

Message integrity. The core Express protocol does not protect message integrity, so a malicious server could undetectably corrupt the contents of a mailbox. This can be remedied in a straightforward way by using MACs. Given that the clients writing to and reading from a mailbox share a secret to establish an address, they could instead use a master secret to derive (e.g., via a hash) a mailbox address and a MAC key. Messages written to Express could then be MACed before being split

	Client	Servers
Communication	$O(\lambda^2 + \|m\|)$	$O(\lambda)$
AES Evaluations	$O(\lambda + \|m\|)$	$O(n(\lambda + \|m\|))$
Field Operations	$O(1)$	$O(n)$

Table 2: Complexity of processing a single write in Express with n mailboxes, message size |m|, and security parameter λ. Communication measures bits sent only.

into shares via a DPF. Since a MAC-then-encrypt approach provides authenticated encryption when the encryption is done in counter mode [12] (as we do), Express with MACed messages provides authenticated encryption on the messages.

6 Using Express for Whistleblowing

Having described the core Express system itself, this section covers two important considerations involved in using Express for whistleblowing: plausible deniability for whistleblowers and agreeing on mailbox addresses.

First, in order to provide meaningful security in practice, Express must hide both the recipient of a given client's message as well as *whether* a client is really communicating with a journalist. We discuss how to provide plausible deniability for Express clients in Section 6.1. Second, to set up their communication channel, a journalist and whistleblower must agree on a mailbox address through which they will communicate. This can be done either in person or via a dialing protocol as described in Section 6.2.

6.1 Plausible Deniability

We now turn to the goal of hiding whether or not a client is really communicating with a journalist. If Express were only to be used by journalists and their sources, it would fundamentally fail to serve its purpose. Although no observer could determine which journalist a given message was sent to, the mere fact that someone sent a message using Express reveals that she must be a source for some journalist. In order to provide plausible deniability to whistleblowers, other, non-whistleblowing users must send messages through the system as well.

One solution for this problem, first suggested in the Conscript system [26], is to have cooperative web sites embed Javascript in their pages that generates and submits dummy requests. For example, the New York Times home page could be modified such that each time a consenting user visits (or for every nth consenting user that visits), Javascript in the page directs the browser to generate a request to a special write-only Express dummy address that the servers maintain but for which each server generates its own encryption key not known to any user. Since no user has the keys to unlock this address, messages written to it can never be retrieved, and Express's metadata-hiding property guarantees that messages sent to the dummy address are indistinguishable from real messages sent to journalists. This enables creating a great deal of cover traffic and gives clients who really are whistleblowers plausible deniability, as long as communication patterns between users and the Express servers are the same for real and cover traffic. Moreover, only one large organization needs to implement this technique for all news organizations who receive messages through Express to benefit from the cover traffic. The exact quantity of cover traffic required to provide the appropriate level of protection for whistleblowers using Express is ultimately a subjective decision, but the Express metadata-hiding guarantee implies that a whistleblower sending a message through Express cannot be distinguished among the set of all users sending messages through Express, be they real messages or cover messages.

Express is particularly well-suited to this approach for two reasons: aligned incentives and low client side costs. First, participating news organizations all have web sites and a natural incentive to direct cover traffic to the Express system. Even if only one or a few organizations among them are willing to risk adding dummy traffic scripts to their pages, everyone benefits. In fact, even the same organizations who are willing to host the Express servers could add the dummy scripts to their own news websites to ensure adequate cover traffic. Second, as demonstrated in Section 7, Express's extremely low client computation and communication requirements lend themselves particularly well to this approach, since the client can easily run in the background on a web browser, even in computation or data-restricted settings such as mobile devices. We empirically evaluate a JavaScript version of the Express client in Section 7.2 and find it imposes very little additional cost on the browser.

Using in-browser JavaScript to give users plausible deniability raises a number of security and ethical concerns. We defer to the Conscript paper [26] for an extensive discussion of the security and ethical considerations involved and note that it is also possible to generate cover traffic for Express using a standalone client, as is common in other systems.

6.2 Dialing

In order to use Express, a journalist and source must agree on the mailbox address which the source will use to send messages to the journalist. Journalists who make initial in-person contact with sources could, for example, distribute business cards with mailbox addresses on them in QR code form.

Journalists and sources could also use a more expensive *dialing* protocol to share an initial secret before moving to Express to more efficiently communicate longer or more frequent messages. One approach to dialing that can conveniently integrate with Express is to use an improved version of the Riposte [24] system as a dialing protocol. Riposte offers a public broadcast functionality that progresses in fixed time epochs, where anyone can announce a message to the world. Since journalists can easily post their public keys online, e.g., next to their name at the bottom of articles they write, anyone wishing to connect with a particular journalist can send a mailbox address (and perhaps some introductory text) encrypted under that journalist's public

key with no other identifying information. A client run by a journalist can download all Riposte messages sent in a day and identify those encrypted under that journalist's public key. The journalist can then register any mailbox addresses sent to it and communicate with whoever sent the messages via Express. This requires mailbox owners (in this case, the journalist) to choose virtual addresses instead of the servers, but the probability of colliding addresses is low because the virtual address space is large. Using this approach to dialing gives Express users the ability to bootstrap from a single message in a dialing system with fixed-duration rounds to as many messages as they want in a system which processes messages asynchronously.

Since Riposte has a similar underlying architecture to Express, a number of the techniques used in Express could be used to make it a more effective dialing protocol. Most importantly, instead of using Riposte's DPFs and auditing protocol, which are less efficient and require a third non-colluding server, the dialing protocol can use a Riposte/Express hybrid approach where the DPF and auditing protocol are those of Express. This means that the dialing protocol relies on the same trust assumptions as the main protocol, and it can even be deployed on the same servers.

Integrity in the dialing protocol can be ensured in a way similar to the main protocol as well. Instead of sending only a mailbox address, clients send a secret from which a mailbox address and MAC key can be derived, and the encrypted message is then MACed using that key. To ensure that servers can't tamper with or erase messages by changing their state after seeing that of the other server, they are required to publish and send each other commitments to (hashes of) the message shares they hold before publishing the actual databases of messages.

7 Implementation and Evaluation

We implement Express with the underlying cryptographic operations (DPFs, auditing) in C and the higher level functionality (servers, client) in Go. We use OpenSSL for cryptographic operations in C and base our DPF implementation in part on libdpf [18], which is in turn based on libfss [59, 60]. We also re-implemented the client-side computations involved in sending a write request in JavaScript for the whistleblowing application, using the SJCL [53, 54] and TweetNaCl.js [1] libraries for crypto operations. We implement the DPF construction [15] and the auditing protocol using the field $\mathbb{F}_p$ of integers modulo the prime $p = 2^{128} - 159$, since these field elements have a convenient representation in two 64-bit words. Our implementation does not include the client-side integrity checks described in Section 5, but these checks can be added by clients with no impact on server-side code or performance.

We evaluate Express on three Google Cloud instances (two running the servers and a third to simulate clients) with 16-core intel Xeon processors (Haswell or later) with 64GB of RAM each and 15.6 Gbps bandwidth. We run all three in the same datacenter to minimize network latency and focus comparisons to other systems on computational costs since we begin our evaluation by considering communication separately. We evaluate the JavaScript implementation of the whistleblowing client on a laptop with an Intel i5-2540M CPU @ 2.60GHz and 4GB of RAM running Arch Linux and the Chromium web browser. All experiments use security parameter $\lambda = 128$.

We compare Express to Riposte [24] and Pung [4], two prior works that also provide cryptographic metadata-hiding guarantees, albeit in slightly different settings. We choose to compare to these systems because, like Express, they also provide cryptographic security guarantees and only rely on a small number of servers to provide their security guarantees. Riposte requires 3 servers, of which two must be honest (a stronger trust assumption than Express) whereas Pung requires only a single server which can potentially be malicious (a weaker trust assumption). We rerun the original implementations of Riposte and Pung on the same cloud instances used to evaluate Express. Our evaluation results do not distinguish between real and dummy messages because the two are identical from a performance perspective.

We find that Express reduces communication costs by orders of magnitude compared to Riposte and Pung, with clients using over 100× less bandwidth than Riposte and over 4000× less bandwidth than Pung when sending a message in the presence of one million registered mailboxes. On the client implemented in C/Go, Express requires 20ms of computation to send a write request, even in the presence of one million registered mailboxes, and our JavaScript client performs similarly, requiring 51ms for the same task.

We compare the performance of our auditing protocol to the prior protocol proposed by Riposte [24]. Despite making a weaker trust assumption and requiring only two servers, our protocol reduces client computation time by several orders of magnitude, resulting in audit compute time of under 5 *micro*seconds regardless of the number of registered mailboxes and reducing overall client compute costs by 8× compared to an implementation that uses Riposte's auditing protocol.

On the server side, we show that Express's throughput and latency costs are better than prior work. We also calculate the dollar cost of running each system to send one million messages and find that Express costs 6× less to operate than Riposte, the second cheapest system. Throughout our experiments we generally compare to prior work on message sizes comparable to or larger than those used in their original evaluations. Since the recent whistleblower's report to the US intelligence community's inspector general contained 25.3KB of text [6] and last year's widely reported anonymous op-ed in the New York Times contained about 9KB of text [5], we make sure to evaluate Express on 32KB messages as well.

7.1 Communication Costs

Figures 3 and 4 show communication costs for each party when sending a 160 Byte message and compares to costs in Riposte [24] and Pung [4]. We use a smaller message size than

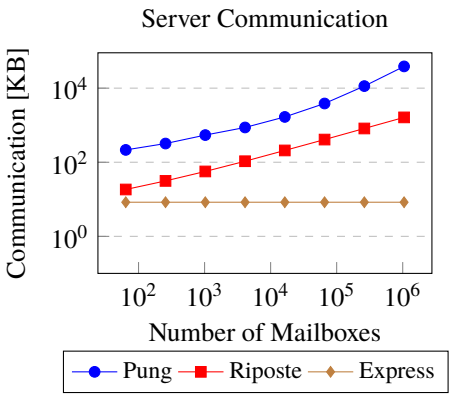

Figure 3: Server communication costs when sending 160 Byte messages, including both data sent and received. Riposte also requires an auditing server whose costs are not depicted.

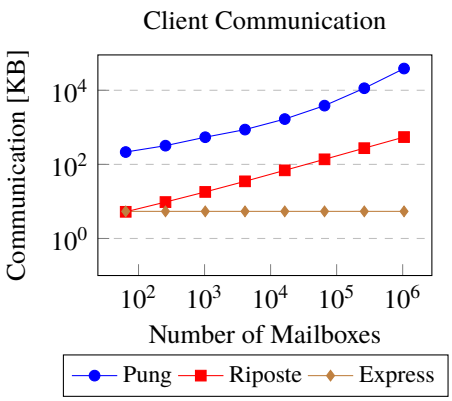

Figure 4: Client communication costs when sending 160 Byte messages, including both data sent and received. Express requires significantly less communication than prior work.

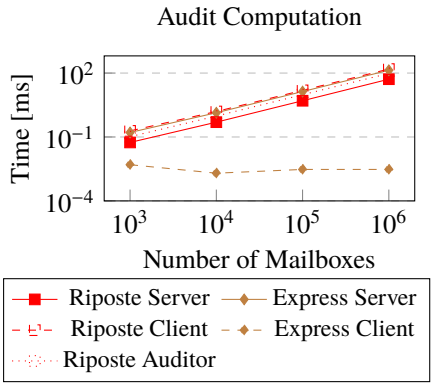

Figure 5: Our auditing protocol dramatically reduces computation costs for the client while server-side costs remain comparable to prior work, where audit computation time is dwarfed by DPF evaluation anyway.

in our subsequent experiments to focus on measuring the role of the DPF and auditing in communication costs. Communication costs always increase linearly with the size of the messages being sent. Express's communication costs are *constant* regardless of the number of mailboxes, compared to asymptotically $\sqrt{n}$ in Riposte, the system with the next lowest costs. For 2^{14} mailboxes, Express has 8.34KB of communication by the server and 5.39KB by the client for each write. The corresponding costs in Riposte are 208KB and 69KB, respectively, representing communication reductions of 25× on the server side and 13× on the client. Riposte additionally requires a third auditing server which incurs 13.8KB of communication, whereas Express has no such requirement. For about one million (2^{20}) mailboxes, Express requires 101× less communication than Riposte on the client side and 195× less on the server side. The communication reduction compared to Pung in this setting is 4,631× on the server side and 7,161× on the client side, reflecting the high cost of providing security with only one server as Pung does. Our communication savings come from using log-sized DPFs that write into a large but fixed-size virtual address space for write requests and from our new auditing protocol whose communication costs do not increase with the number of mailboxes.

7.2 Client Costs

Client computation time in both our native C/Go and in-browser Javascript implementations remains constant as the number of mailboxes on the server side increases: since the client always prepares a DPF to be run on the 2^{128}-sized virtual address space, the cost of preparing the DPF does not grow with the number of mailboxes, and the client-side auditing cost is constant as well. To send a 1KB message, our client takes 20ms in C/Go and 51ms in Javascript. Combined with the low client communication costs in Figures 3 and 4, this shows that an Express client can easily be deployed as background Javascript in a web page to create cover traffic, as explained in Section 6.1.

To further explore performance implications of an Express client being embedded on a major news site, we measured the page load times of the New York Times, Washington Post, and Wall Street Journal websites. On average, these pages took 5.4, 3.4, and 2.2 seconds to load completely (over a 50MBit/sec connection), so the computation costs of our client in the browser are less than 3% of current page load times and can occur in the background without impacting user experience. We also measured the sizes of the three websites (without caching) at 4.9MB, 9.1MB, and 8.2MB, respectively. Our JavaScript implementation with dependent libraries takes 72.5KB of space, so adding our code would increase a site's size by less than 1.5%.

Auditing. In addition to enabling improved communication efficiency, as seen above, our auditing protocol dramatically reduces computation costs for the client. Figure 5 shows the computation costs of our auditing protocol as compared to the protocol used in Riposte [24], which we re-implemented for the purpose of this experiment. Unlike Riposte, where client and server computation costs for auditing are comparable, our protocol runs in $O(1)$ time on the client, taking less than *5 microseconds* regardless of how many mailboxes are registered on the servers. This is about 55,000× less than the client computation cost for auditing in Riposte for one million mailboxes and translates to overall client computation on our system running 8× faster than it would if it were using the Riposte auditing protocol. In addition to the asymptotic improvement, our protocol uses only hardware-accelerated AES evaluations, whereas Riposte's auditing protocol involves a mix of AES evaluations and more costly SHA256 hashes.

Our auditing protocol's performance is comparable to Riposte on the server side, but it does not require a third auditing server as Riposte does. The performance bottleneck on the servers is DPF evaluations, not auditing, so server side performance improvements in auditing would only result in negligible

improvements in end-to-end performance. As we will see, Express outperforms Riposte's overall throughput despite not significantly changing server side auditing costs.

7.3 Server Performance

We now measure the performance of Express on the server-side. We measure the total throughput of the system, the latency between when a client sends a message and when the mailbox owner can read it, and the cost in dollars of running Express.

Throughput. We compare Express's throughput to Riposte [24]. Figure 7 shows the comparison between Express and Riposte for 1KB messages, where throughput is measured as the number of writes the servers can process per unit time. Express's throughput is 1.4-6.3× that of Riposte in our experiments, and Express's throughput when handling 32KB messages is comparable to Riposte when handling only 1KB messages for up to about 50,000 mailboxes. Both systems are ultimately computation-bound by the number of DPF evaluations required to process writes. The graph shows the high throughput of each system drops significantly as they shift from being communication-bound to being computation-bound by DPF evaluations for increasingly large numbers of mailboxes.

Like Express, Riposte uses DPFs to write messages across two servers. Unlike Express, Riposte requires a third party to audit user messages and must run its protocol in rounds to provide anonymity guarantees to its users. The rounds are necessary for Riposte's anonymous broadcast setting because all messages are public, so if messages were revealed after each write, the author of a message would clearly be whoever connected to the system last. In contrast, Express messages can be delivered immediately without waiting for a round to end.

Another difference between Express and Riposte is that Riposte relies on a probabilistic approach based on hashing for users to decide where to write with their DPF queries. This means that there is a chance messages will collide when written to the same address, rendering all colliding messages unreadable. We evaluated Riposte with parameters set to allow a failure rate of 5%, meaning that 1 in 20 messages would be corrupted by a collision and not delivered, even after Riposte's collision-recovery procedure. Express's virtual address system avoids this issue because the space of virtual addresses is so large that collisions would only occur with negligible probability.

Latency. Since Express does not require any synchronization between clients and the Express servers, the latency of a write request consists only of the time for the servers to process the request and for the mailbox owner to read the message. Figure 6 shows how latency for processing a single write request scales as the number of mailboxes increases for various mailbox sizes. After about 10,000 mailboxes, or even 1,000 mailboxes for larger message sizes, message processing becomes bound by the latency of computing AES for each DPF evaluation, so total latency increases linearly with the number of DPFs that must be evaluated (one per mailbox).

In prior metadata-hiding communication systems, message delivery latency depends on a deployment-specified round duration. As such, it is difficult to directly compare latency in Express to prior work. We can, however, compare to the computation time on the servers to process one message and deliver it to its recipient. For example, Riposte's "latency" under this metric is simply the time to process a DPF write and then run an audit. A more interesting comparison is to see how Express's server-side costs compare to a different architecture, such as the single-server PIR-based approach of Pung [4].

Since Pung [4] uses fast writes and more expensive reads whereas Express has fast reads but expensive writes, we run both systems with a write followed by a read, as required by Pung's messaging use case. As shown in Figure 6, Express outperforms Pung by 1.3-2.6× when run with 100-1,000,000 mailboxes for 1KB messages. When we increase the message size to 10KB, we find that Pung is $2-2.9\times$ slower than Express and closely matches Express's performance on 32KB messages. Note that the comparison to Pung is not quite apples to apples because Pung operates in a stricter single-server security setting.

Total system cost. Having measured Express's throughput and latency, we now turn to the question of Express's cost in dollars (USD). Our evaluation focuses on the dollar cost of running the infrastructure required for Express in the cloud and excludes human costs such as paying engineers to deploy and maintain the software. The primary non-human costs in running Express, as with any metadata-hiding system, come from running the necessary servers and passing data through them. Using the data from our evaluation thus far, we estimate the price of running Express to send one million messages using public Google Cloud Platform pricing information. We calculate the cost of running the system as the cost of hosting the Express servers for the length of time required to process one million messages plus the data passed between the servers and back to the client (data passing into Google cloud instances from clients outside is free). We price the instances according to costs for various regions in the US and Canada and calculate data charges using the prices for data transfer between regions in the US and Canada (for communication between servers) or with the public internet (for communication with clients).

The results of this estimation process appear in Figure 8, where we carry out similar calculations for Pung and Riposte. As depicted in the figure, processing one million messages with Express costs 5.9× less than Riposte, the closest prior work measured, in the presence of 100,000 mailboxes. The high cost of running Pung comes from its communication costs, where data egress charges far outweigh the cost of hosting the system. The data egress cost of sending one million messages in Pung with 262,144 registered mailboxes exceeds $\$1,000$. On the other hand, Express and Riposte incur smaller data costs, \$0.05 per million messages in Express and \$4.21 per million messages in Riposte with one million registered mailboxes. The large gap in cost between Express and Riposte comes from hosting the servers themselves. Express's higher throughput means it can process one million messages more quickly than

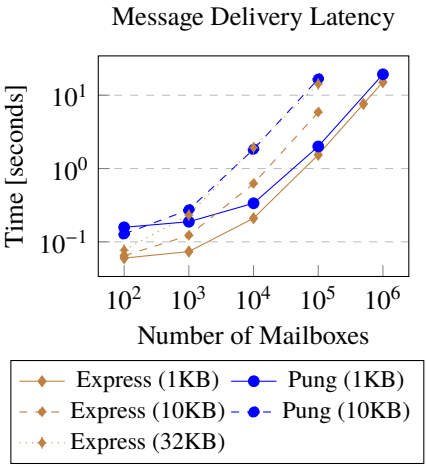

Figure 6: Message delivery latency in Express and Pung for various message sizes. Express outperforms Pung by 1.3 – 2.6× for 1KB messages and by 2.0 – 2.9× for 10KB messages. Pung's performance for 10KB messages is comparable to Express's performance for 32KB messages.

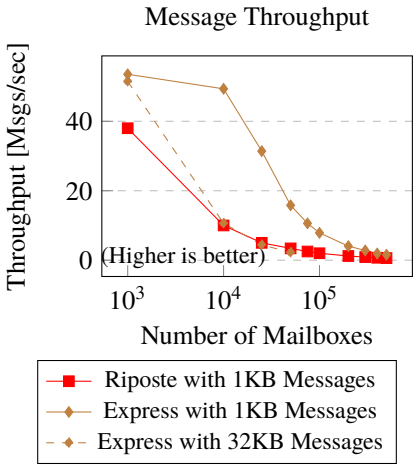

Figure 7: Express's throughput is 1.4-6.3× that of Riposte for 1KB messages. Even with 32KB messages, Express's throughput is still comparable to Riposte on 1KB messages. For large numbers of mailboxes, both systems are computation-bound by the number of DPF evaluations required to process writes.

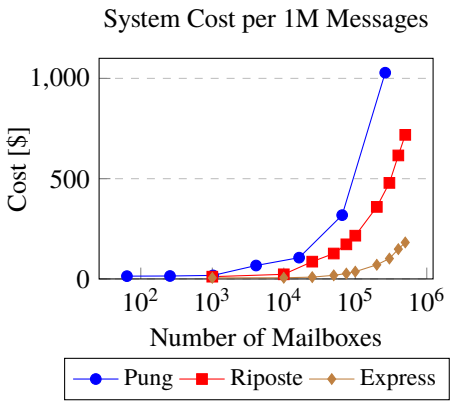

Figure 8: Dollar costs to run end-to-end metadata hiding systems with cryptographic security guarantees. Prices are based on Google Cloud Platform public pricing information for compute instances and data egress. Processing one million messages in Express in the presence of 100,000 registered mailboxes costs 5.9× less than the next cheapest system.

Riposte, and the fact that it requires only two servers, compared to three in Riposte, means that the cost per hour of running Express is approximately 2/3 that of running Riposte. Hosting costs per 24 hours, excluding data costs, are $11.75 for Pung, $37.25 for Riposte, and $24.68 for Express, corresponding to the number of servers each system needs (including cost differences for hosting servers in different regions).

Comparison to differential privacy systems. As described in Section 2.3, systems based on differential privacy (DP) exchange gradual metadata leakage over time for stronger performance. Although this fundamental difference in security properties makes it difficult to do a direct comparison to DP systems such as Vuvuzela [58], Stadium [56], and Karaoke [46], we will attempt here to roughly compare Express to published performance results for Vuvuzela and Karaoke. Vuvuzela operates with the same distributed trust model as Express, with a small number of servers, whereas Karaoke is designed for use in a setting with many servers. See Section 2.3 for a discussion of these two approaches to distributing trust.

One further difference to keep in mind when comparing existing DP systems to Express (as well as the systems we have compared Express to thus far) is that costs in Riposte, Pung, and Express increase in the number of mailboxes registered, while costs in existing DP-based systems increase in the number of *users* registered. This means that a fully connected communication graph on N users would require N^2 mailboxes in Express but would not require additional cost in DP systems beyond that of N users and the high volume of traffic required for all of them to talk to each other. Fortunately, in most messaging systems, each user only has a small number of active contacts relative to the total number of users on the platform, so this difference should not cause harm in practice.

Vuvuzela's end-to-end latency to deliver a 256 byte message for the lowest security setting on which it was evaluated hovers around 8 seconds for 10,000 users and 20 seconds for one million users [58]. By comparison, Express takes 210ms to write and then read a larger 1KB message when there are 10,000 mailboxes and 15 seconds when there are one million mailboxes. The higher latency in Vuvuzela is due to cover traffic messages sent before a message can be delivered.

Karaoke operates using a variable number of servers, and its end-to-end latency to deliver a 256 byte message hovers around 6 seconds for one million users and 100 servers when up to 20% of servers are malicious [46]. However, Karaoke's latency approximately triples when moving from providing security against 20% malicious servers to 50% malicious servers, which more closely matches the one-out-of-two security provided by Express. Since Karaoke's evaluation was also conducted on more powerful machines than ours, we conclude that latency is roughly comparable between Express and Karaoke.

On the other hand, not requiring cryptographic security allows DP solutions to achieve higher throughput than cryptographic systems. As such, they can process messages faster and at lower cost than Express. However, in addition to the difference in security guarantees, they achieve their low price by pushing the true cost of operating the system onto clients. To send and receive messages, clients must *always* remain online.

8 Related Work

The most widely used anonymity system in use today is without a doubt Tor [31], which relies on onion routing. Secure-

Drop [10, 55] is a widely used Tor-based tool to allow sources to anonymously connect with journalists to give tips. Although our work focuses on hiding metadata and not on preserving anonymity, anonymity systems are often used even when clients only wish to hide metadata. Although a number of works precisely model and analyze the security offered by Tor [9, 41, 42], it is unfortunately vulnerable to traffic analysis attacks if a passive adversary controls enough of the network [30, 37, 38]. A recent impossibility result suggests that this limitation may be necessary for broad classes of anonymity systems [29].

Cryptographic security. Express belongs to a broad family of works which aim to give cryptographic guarantees regarding anonymity and metadata-hiding properties. One category of works in this area include systems based on mix-nets [25, 27, 34, 44, 50, 52, 61] which involve all users in a peer to peer system participating in shuffling messages [16, 17]. Later work has added verifiability to this model [44] and outsourced the shuffling to a smaller set of servers [52, 61]. Most recently, mixing techniques have been extended to support large numbers of users in Atom [43] and XRD [45]. Systems in this line of work suffer from high latency due to the need to run many shuffles and require participation by a large number of servers run by different operators to achieve security.

An important difference between Express and mixnets relates to tradeoffs in anonymity and latency. Since a user's anonymity set is based on the number of messages being shuffled together, a mixnet operator must choose between a high-latency setting with a large anonymity set or a lower latency setting with a smaller anonymity set. For example, if 1,000 messages are sent through the system in one hour, a mixnet that wants an anonymity set size of 1,000 must wait an hour before it can deliver messages, whereas Express can achieve the same anonymity set but deliver messages immediately. A mixnet's anonymity set is restricted to the number of messages included in the mixing, which in turn depends on the desired latency, leading to an inherent tradeoff between anonymity and latency [29]. Express messages, on the other hand, are in some sense mixed with all the prior messages sent through the system. This means that while a mixnet may have to compromise on anonymity set size to meet a given latency goal, Express does not.

Another class of cryptographic messaging solutions use private information retrieval techniques [3, 15, 20, 33, 48, 49] to render reads or writes into a database of mailboxes private and target a variety of use cases [4, 13, 19, 22, 24, 39, 40]. Express falls into this category. Riposte [24] and, more recently, Blinder [2], provide anonymous broadcast mechanisms using DPFs [33], and Talek [19] offers a private publish-subscribe protocol. P3 [39] deals with privately retrieving messages with more expressive search queries. Pung [4] operates in a single-server setting and therefore requires weaker trust assumptions than Express, but as we show in Section 7, has higher costs than Express as well.

Differential privacy. Another class of works make differential privacy guarantees [32] instead of cryptographic guarantees. These systems typically achieve better performance but at the cost of setting a *privacy budget* that dictates how much privacy the system will provide. These works include Vuvuzela [58], Alpenhorn [47], Stadium [56], and Karaoke [46].

9 Conclusion

We have presented Express, a metadata-hiding communication system that requires only symmetric key cryptographic primitives while providing near-optimal communication costs. In addition to order of magnitude improvements in communication cost, Express reduces the dollar cost of running a metadata-hiding communication system by 6× compared to prior work. Our implementation is open source and available online at `https://github.com/SabaEskandarian/Express`.

Acknowledgments

We would like to thank Dima Kogan, Alex Ozdemir, the anonymous reviewers, and our shepherd, Esfandiar Mohammadi, for their thoughtful comments.

This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Infosys, NEC, and VMware—as well the NSF under CAREER grant CNS-1651570. The work was additionally funded by NSF, DARPA, a grant from ONR, and the Simons Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or the National Science Foundation.

References

[1] Tweetnacl.js. `https://github.com/dchest/tweetnacl-js`.

[2] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder: Mpc based scalable and robust anonymous committed broadcast. Cryptology ePrint Archive, Report 2020/248, 2020.

[3] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *IEEE Symposium on Security and Privacy, SP*, 2018.

[4] Sebastian Angel and Srinath T. V. Setty. Unobservable communication over fully untrusted infrastructure. In *OSDI*, 2016.

[5] Anonymous. I am part of the resistance inside the trump administration. `https://www.nytimes.com/2018/09/05/opinion/trump-white-house-anonymous-resistance.html`, 2018.

[6] Anonymous. Whistleblower complaint to us intelligence community inspector general. `https://www.documentcloud.org/documents/6430351-Whistleblower-Complaint.html`, 2019.

[7] AP. Gov't obtains wide ap phone records in probe. *Associated Press*, 2013.

[8] AP. Times says justice seized reporter's email, phone records. *Associated Press*, 2018.

[9] Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. Anoa: A framework for analyzing anonymous communication protocols. *J. Priv. Confidentiality*, 2016.

[10] Charles Berret. Guide to securedrop. `https://www.cjr.org/tow_center_reports/guide_to_securedrop.php`, 2016.

[11] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *CRYPTO*, 2019.

[12] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography (version 0.5, Chapter 9)*. 2017. https://cryptobook.us.

[13] Nikita Borisov, George Danezis, and Ian Goldberg. DP5: A private presence service. *PoPETs*, 2015(2):4–24, 2015.

[14] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT*, 2015.

[15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *ACM CCS*, 2016.

[16] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

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

[18] Weikeng Chen. libdpf. https://github.com/weikengchen/libdpf, 2018.

[19] Raymond Cheng, Will Scott, Bryan Parno, Irene Zhang, Arvind Krishnamurthy, and Thomas Anderson. Talek: a Private Publish-Subscribe Protocol. Technical Report UW-CSE-16-11-01, University of Washington Computer Science and Engineering, Seattle, Washington, Nov 2016.

[20] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[21] David Cole. We kill people based on metadata. *New York Review of Books*, 2014.

[22] David A. Cooper and Kenneth P. Birman. Preserving privacy in a network of mobile computers. In *IEEE Symposium on Security and Privacy, SP*, 1995.

[23] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI*, 2017.

[24] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *IEEE Symposium on Security and Privacy, SP*, 2015.

[25] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *ACM CCS*, 2010.

[26] Henry Corrigan-Gibbs and Bryan Ford. Conscript your friends into larger anonymity sets with javascript. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*, pages 243–248, 2013.

[27] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In *USENIX Security*, 2013.

[28] Cora Currier. Planned nsa reforms still leave journalists reason to worry. *Columbia Journalism Review*, 2014.

[29] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two. In *IEEE Symposium on Security and Privacy, SP*, 2018.

[30] Roger Dingledine. One cell is enough to break tor's anonymity, 2009.

[31] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.

[32] Cynthia Dwork. Differential privacy. In *ICALP*, 2006.

[33] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT*, 2014.

[34] Sharad Goel, Mark Robson, Milo Polte, and Emin Gun Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[35] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *CRYPTO*, 1984.

[36] Glenn Greenwald. Nsa collecting phone records of millions of verizon customers daily. *The Guardian*, 2013.

[37] Amir Houmansadr and Nikita Borisov. The need for flow fingerprints to link correlated network flows. In *PETS*, 2013.

[38] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul F. Syverson. Users get routed: traffic correlation on tor by realistic adversaries. In *ACM CCS*, 2013.

[39] Lea Kissner, Alina Oprea, Michael K. Reiter, Dawn Xiaodong Song, and Ke Yang. Private keyword-based push and pull with applications to anonymous communication. In *ACNS*, 2004.

[40] Lea Kissner, Alina Oprea, Michael K. Reiter, Dawn Xiaodong Song, and Ke Yang. Private keyword-based push and pull with applications to anonymous communication. In *ACNS*, 2004.

[41] Christiane Kuhn, Martin Beck, Stefan Schiffner, Eduard A. Jorswieck, and Thorsten Strufe. On privacy notions in anonymous communication. *PoPETs*, 2019.

[42] Christiane Kuhn, Martin Beck, and Thorsten Strufe. Breaking and (partially) fixing provably secure onion routing. In *IEEE Symposium on Security and Privacy, SP*, 2020.

[43] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: Horizontally scaling strong anonymity. In *SOSP*, 2017.

[44] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An efficient communication system with strong anonymity. *PoPETs*, 2016(2):115–134, 2016.

[45] Albert Kwon, David Lu, and Srinivas Devadas. XRD: scalable messaging system with cryptographic privacy. *CoRR*, abs/1901.04368, 2019.

[46] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *OSDI*, 2018.

[47] David Lazar and Nickolai Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *OSDI*, 2016.

[48] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR : Private information retrieval for everyone. *PoPETs*, 2016(2):155–174, 2016.

[49] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *STOC*, 1997.

[50] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system. In *USENIX Security*, 2017.

[51] Julie Posetti. *Protecting Journalism Sources in the Digital Age*. UNESCO, 2017.

[52] Len Sassaman, Bram Cohen, and Nick Mathewson. The pynchon gate: a secure method of pseudonymous mail retrieval. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*, pages 1–9, 2005.

[53] Emily Stark, Michael Hamburg, and Dan Boneh. Stanford javascript crypto library. https://github.com/bitwiseshiftleft/sjcl, 2009.

[54] Emily Stark, Michael Hamburg, and Dan Boneh. Symmetric cryptography in javascript. In *ACSAC*, 2009.

[55] Aaron Swartz. Securedrop. https://securedrop.org/, 2013.

[56] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *SOSP*, 2017.

[57] United Nations High Commissioner for Human Rights. The right to privacy in the digital age, 2018.

[58] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai

Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *SOSP*, 2015.

[59] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. libfss. `https://github.com/frankw2/libfss`, 2017.

[60] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *NSDI*, 2017.

[61] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI*, 2012.

A Security Arguments

This appendix formalizes and proves the soundness and metadata-hiding security properties described in Section 2.

Soundness. We formalize soundness as follows.

Definition 1 (Soundness)**.** We define the following soundness game $\mathsf{SOUND}[\lambda]$ played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ who simulates the behavior of servers A and B. Both $\mathcal{A}$ and $\mathcal{C}$ are given λ as input.

- Setup. Challenger $\mathcal{C}$ creates an initially empty list I of compromised mailbox indices. Adversary $\mathcal{A}$ requests creation of a number of mailboxes N of its choosing. There are two ways in which it may create a mailbox:
  1. Adversary $\mathcal{A}$ performs the role of a user interacting with the servers to create a new mailbox. Challenger $\mathcal{C}$ adds this mailbox to I.
  2. Adversary $\mathcal{A}$ instructs $\mathcal{C}$ to create a mailbox where $\mathcal{C}$ plays the role of both the user and the servers, saving the user's state (and in particular, the mailbox keys) at the end of the registration process.
- Queries and Corruptions. Adversary $\mathcal{A}$ sends requests to the servers, controlled by $\mathcal{C}$. At any time, it may send $\mathcal{C}$ a mailbox index i, at which point $\mathcal{C}$ will send the saved state of the user who registered mailbox i and add i to list I.
- Output. Challenger $\mathcal{C}$ performs a read on each registered mailbox. If $|I| < N$ and any mailbox outside of the list I contains nonzero contents, the adversary wins the game.

We say a messaging scheme is *sound* if no PPT adversary can win the soundness game above with greater than negligible probability in the security parameter λ.

Claim. *The Express scheme is sound.*

Proof. The soundness proof follows closely from the soundness of our auditing protocol. For each write request sent to the Express servers, we consider two cases: where the write modifies one mailbox and where the write modifies more than one mailbox. If a write modifies more than one mailbox, then it will not be applied to the database of mailboxes, except with negligible probability in λ, by the soundness property of the auditing protocol. This means that we must only consider writes that modify a single mailbox. The adversary does not know the virtual addresses of mailboxes outside of I, but it only wins the soundness game if it produces a DPF that writes to the address of a mailbox outside of I. This can only occur with probability $2^{-\lambda}$ (for $\lambda = 128$ in our instantiation of the protocol), which is also negligible. Thus an adversary can only win the soundness game with probability negligible in λ. □

Metadata-hiding. We can formalize the definition of metadata-hiding by requiring that there exists an efficient *simulator* algorithm Sim that, given the list ℓ of honest clients who connect with the servers, produces an output which is computationally indistinguishable from the view of an adversary $\mathcal{A}$ who controls any number of users and one server while processing requests from the remaining honest users, subject to the restriction that the recipients of the messages from honest users are never among those controlled by $\mathcal{A}$. More specifically, ℓ should include which client connects, time of connection, and size of message transmitted for each connection made to the compromised server. Given this information, the client can simulate the content of the messages sent by the honest client.

This definition satisfies our intuitive notion of metadata-hiding because it means that for each message, the server learns nothing about who the message is sent to, as everything it learns could be simulated before it even sees the request. This information would be contained in the content of the honest client's messages, which are not given to the simulator. We sketch a proof of the metadata-hiding security argument below. The proof relies on the zero-knowledge property of the auditing protocol, the privacy of the DPFs used, and the security of the encryption used for access control.

Claim (Informal)**.** *There exists an algorithm* Sim *that, given the list* ℓ *of honest client connections to the Express servers, simulates the view of an adversary* $\mathcal{A}$ *who controls one Express server and any number of clients, subject to the restriction that the recipients of the honest clients' messages are never among those controlled by* $\mathcal{A}$*.*

Proof (sketch). Sim simulates write requests from honest users and the process of auditing them by invoking the simulator implied by the zero-knowledge property of the auditing protocol. Note that this in turn uses the simulator implied by the definition of DPF privacy to generate DPF shares. Moreover, whenever malicious users request to read the contents of mailboxes, the simulated honest server(s) returns encryptions of zero.

The proof that this simulator gives the adversary $\mathcal{A}$ a view indistinguishable from interaction with a real honest server and honest users is fairly straightforward. First, since the adversary knows the virtual addresses of honest users' mailboxes, as well as one of the two keys needed to read the contents of those mailboxes (if it has compromised one of the servers), it can send read requests for the contents of honest mailboxes. However, since the adversary does not see the second key to any honest users' mailboxes, we invoke the semantic security of the encryption scheme used to protect honest mailbox contents to show that the messages returned from read requests to an honest server are indistinguishable from encryptions of zero.

From here, just as in the case of soundness, the proof follows from the security of the auditing scheme. From the zero-knowledge property of the auditing scheme, we know that the view of either server in the auditing protocol can be simulated. But the view of each server in Express's auditing protocol is the same as the view of that server in the overall protocol, since the server's view only consists of its shares of the proof input (in the compressed form of a DPF share from which it derives the actual inputs) and the proof messages themselves. □

B SNIPs and Analysis of Auditing Protocol

This appendix sketches the instantiation of the proofs used in our auditing protocol as well as the analysis of the auditing protocol. Full details and a security proof for this proof system can be found in the Prio paper [23]. We include the instantiation of the proof here for completeness, including some improvements described in the follow-up work of Boneh et al. [11].

The size of a SNIP proof is linear in the number of multiplication gates in the arithmetic circuit representing the statement to be proved. In our case, there are 2 multiplications. The client numbers the gates as 1 and 2. The idea of the proof is to create three polynomials f, g, and h such that f, g represent the left and right inputs of each gate and h to the outputs of each gate. f is the polynomial defined by the points $(0, r_f), (1, c), (2, m)$, and g is the polynomial defined by the points $(0, r_g), (1, c), (2, C)$, where r_f and r_g are random values chosen by the client. Observe that the servers already hold shares of each point used to define f and g except the random values r_f and r_g, shares of which must be included in the SNIP proof.

Next, h is defined as the polynomial representing the expected outputs of each multiplication gate, or the product $f \cdot g$. Since each of f and g will be of degree 2, h will be of degree 4. The client can compute h from f and g and must send shares of the description of h to each server as part of the proof.

Since the servers now have shares of the inputs and outputs of each multiplication from f, g, and h, they only need to check that $f \cdot g = h$ to be convinced that this relationship holds among their inputs. They do this by evaluating each polynomial at a random point t and checking equality. To compute the product $f(t) \cdot g(t)$, the servers simply evaluate their shares of each function and publish the result. This reveals nothing about f or g except their value at the point t.

The Prio paper [23] and the improvements of Boneh et al. [11] give full proofs of completeness, soundness, and zero-knowledge for this protocol. As a minor optimization, instead of sending one proof as described above, we send two separate SNIPs, one for each of the two multiplications. This results in a slightly larger proof size but simplifies the polynomial multiplications because the polynomials f, g become linear and h becomes quadratic. The security properties of the protocol are unchanged by this modification.

Analysis. Having described the relevant building blocks, we now sketch the analysis of our full auditing protocol. The security properties of our auditing scheme follow directly from those of the two protocols we combine to build it (which we do not re-prove here). Completeness follows directly from the completeness of the verifiable DPF protocol of Boyle et al. as well as the completeness of SNIPs.

Likewise, soundness follows directly from the soundness of these two building blocks, with soundness error equal to the sum of the soundness error of the DPF verification protocol and the SNIP. We prove the following claim.

Claim. *If the servers begin the auditing protocol holding vectors $w_A \in \mathbb{F}^n$ and $w_B \in \mathbb{F}^n$ such that $w = w_A + w_B \in \mathbb{F}^n$ is a vector of Hamming-weight* greater than *one, then the audit will reject, except with error probability $\epsilon = O(1/|\mathbb{F}|)$.*

By taking $\mathbb{F}$ to be a field of size 2^λ, for security parameter λ, we can make the error probability ϵ negligibly small in λ.

The claim is true because the auditing protocol will only accept a false proof if (1) the difference $c^2 - mC = 0$ for a w that has more than one non-zero entry, or (2) the soundness of the SNIP fails to enforce that only inputs satisfying this relationship will be accepted. But the probability of (1) is negligible in $|\mathbb{F}|$ by the security of the DPF verification protocol of Boyle et al. [15], and the probability of (2) is negligible in $|\mathbb{F}|$ by the soundness of SNIPs [11, 23]. By a union bound, the soundness error of the overall protocol is at most the sum of the soundness errors of the verifiable DPF protocol and the SNIPs.

To prove the zero-knowledge property, we must show that there exists a simulator algorithm Sim that can produce outputs whose distribution is computationally indistinguishable from the view of the servers in an execution of the Express auditing protocol where the sum $w_A + w_B$ corresponds to a vector with a single non-zero entry. This algorithm will interact with a potentially malicious adversary $\mathcal{A}$ who plays the role of the server whose view is being simulated. This proves the security of the protocol because it shows that an adversary can learn anything it would learn from actually participating in the protocol by running Sim on its own.

The construction of Sim and subsequent proof of security follow almost directly from the original proof of security for SNIPs used in Prio [23]. To see why, observe that the view of each server in the auditing protocol consists of the server's DPF share, the server's share of the proof, and any messages sent between the servers during the proof. The only difference between this and the standard SNIP simulator is that the server's inputs are compressed in the form of DPF shares instead of being stated explicitly as the vector w_A or w_B. In essence, the DPF can be thought of as an efficient way to encode the server's inputs to the proof. To bridge this difference between our protocol and the original SNIP, we make one small change to the SNIP simulator. The original SNIP simulator samples the server's input share at random. Our modified SNIP simulator will sample the server's input shares using simulated DPF shares instead. Since the proof of zero-knowledge is otherwise identical, we defer to the prio paper for the full proof [23].