Spectrum: High-Bandwidth Anonymous Broadcast with Malicious Security

Zachary Newman
MIT CSAIL
zjn@mit.edu

Sacha Servan-Schreiber
MIT CSAIL
3s@mit.edu

Srinivas Devadas
MIT CSAIL
devadas@csail.mit.edu

Abstract

We present Spectrum, a high-bandwidth, metadata-private file broadcasting system with malicious security guarantees. In Spectrum, a small number of publishers broadcast to many subscribers via two or more non-colluding servers. Subscribers generate indistinguishable cover traffic, hiding which users are publishers, for full metadata privacy.

Spectrum builds on prior work that uses DC-nets for anonymous broadcast. Existing anonymous broadcast systems do not optimize for a setting where there are fewer publishers compared to subscribers – a common situation in real-world broadcasts. To prevent disruption by malicious clients sending malformed requests, we develop a blind authentication protocol that allows servers to reject malicious requests. We also ensure security against malicious servers deviating from protocol and potentially colluding with clients. Our techniques for providing malicious security are applicable to other systems for anonymous broadcast and may be of independent interest.

We implement and evaluate Spectrum. Compared to the state-of-the-art in cryptographic anonymous communication systems, Spectrum is 3–140× faster (and commensurately cheaper). Deployed on two commodity servers, Spectrum allows publishers to share 500 MB in 1h 24m with an anonymity set of 10,000 (for a total cost of about $1.93). This corresponds to an anonymous upload of a full-length 720p documentary movie.

1 Introduction

Free and democratic society depends on an informed public, which sometimes depends on whistleblowers shedding light on misdeeds and corruption. Over the last century, whistleblowers have exposed financial crimes and government corruption [50, 59, 67], risks to public health [30, 40], Russian interference in the 2016 U.S. presidential election [50, 60], presidential misconduct [11, 32, 56, 71], war and human rights crimes [3, 26, 79], and, of note to computer security researchers, digital mass surveillance by U.S. government agencies [12]. Political philosophers debate [2, 23] the ethics of whistleblowing, but agree it often has a positive impact.

Motivation for this work. Whistleblowers take on great personal risks in bringing misdeeds to light. The luckiest enjoy legal protections [80] or financial reward [81]. But many face exile [12], incarceration [39, 60, 66], or risk their lives [79]. More recently, political activist Alexei Navalny was detained and sentenced to prison following the release of documents accusing Russian president Vladimir Putin of corruption and embezzlement [72].

To mitigate these risks, many whistleblowers turn to technology to protect themselves [36]. Secure messaging apps Signal [17] and SecureDrop [4] have proven to be an important resource to whistleblowers and journalists [31, 76]. Encryption does its job, even against the NSA [83]—but it may not be enough to protect from powerful adversaries.

Since the Snowden revelations, governments and the press have focused on *metadata*. The source, destination, and timing of encrypted data can leak information about its contents. Prosecutors used SFTP metadata in the case against Chelsea Manning [88]. Newer technology is still vulnerable: a federal judge found Natalie Edwards guilty on evidence of metadata from an encrypted messaging app [39]. To protect whistleblowers and protect against powerful adversaries, systems must be designed with metadata privacy in mind.

Many academic and practical metadata-hiding systems provide solutions to this problem for some applications. Tor [25] boasts a distributed network of 6,000 nodes and 2 million daily active users (the only such system with wide usage). Tor is fast enough for web browsing, but de-anonymization attacks identify users with a high success rate based on observed traffic [5, 8, 29, 37, 42, 55, 58]. Moreover, the effectiveness of de-anonymization attacks increases with the size of the traffic pattern. Whistleblowers using Tor to upload large files can be more easily de-anonymized compared to casual web users for this reason.

Some recent academic research systems [1, 18, 28, 43–45,

47, 78, 82] address the problem of hiding metadata in anonymous communication, providing precise security guarantees for both direct messaging and "Twitter"-like broadcast applications. However, a limitation of all existing systems is that they are designed for low-bandwidth content, incurring impractical latencies with large messages (see Section 6).

Contributions. Spectrum is the first anonymous broadcast system supporting high-bandwidth broadcasts with security against actively malicious clients and servers. We do so by designing for the many-subscriber and few-publisher setting, which reflects the real-world usage of broadcast platforms. Spectrum scales proportionally to the number of *broadcasts* in the system rather than the total number of users: the primary bottleneck and cause for high-latency in prior work.

This paper contributes:

1. Design of Spectrum, a system for high-bandwidth metadata-private broadcasting with strong robustness and privacy guarantees in a malicious security setting,
2. A general extension we call BlameGame that can be used to "upgrade" anonymous broadcasting protocols for security against de-anonymization attacks from malicious servers,
3. An open-source implementation of Spectrum which we extensively evaluate and compare to existing anonymous broadcasting and communication systems.

Limitations. Spectrum shares some limitations with other metadata-private systems:

1. Spectrum provides anonymity among honest online users and requires all users to contribute cover messages to a broadcast (to perfectly hide network metadata). Thus, subscribers must *upload* as much data as a publisher to provide anonymity for the publisher.
2. Spectrum achieves peak performance with exactly two servers. Instantiating with more than two servers requires using less (concretely) efficient cryptographic primitive: a seed-homomorphic PRG [7].

Other metadata-private systems (e.g., [1, 18, 19, 43, 45, 47, 78]) also provide anonymity within the set of online users and sending fixed-size messages. Unfortunately, it is a necessary cost to pay for strong privacy; if only one user uploads a very large message, the network metadata alone is sufficient to de-anonymize them. However, we show how to amortize the practical impact of this limitation by only requiring *one* cover message per subscriber, even in the case of multiple publishers simultaneously using the system (Section 4).

Paper organization. We formalize the setting and functionality in Section 3. We describe the protocol in Section 4. Section 5 presents the implementation and evaluation. Section 6 surveys related work.

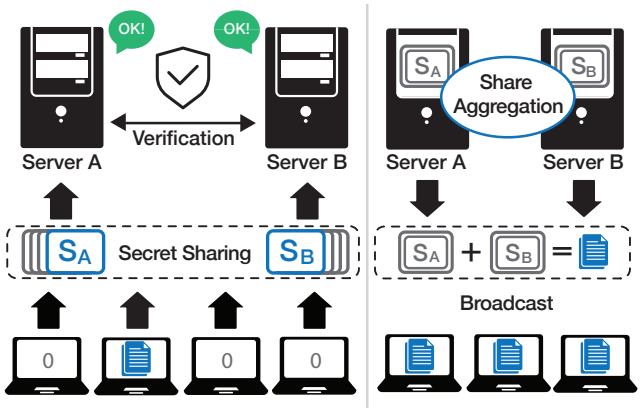

Figure 1: In Spectrum, users upload secret shares to the servers. Servers validate and combine these shares to recover the broadcast message while hiding its provenance.

2 Anonymous Broadcast

In this section we describe high-level techniques for anonymous broadcast.

The setting. In an anonymous broadcast, one or more users (*publishers*) share a *message* (e.g., a file) in a manner that prevents an adversary from learning its source, even with full view of the entire network. In Spectrum, passive users generate cover traffic (indistinguishable dummy messages) to increase the size of the *anonymity set* – the users who plausibly could have originated the broadcast message. We call these passive users *subscribers* as we expect that most users in the system that are not broadcasting are instead consuming broadcasts. We will use the term *client* to mean the program via which a user communicates with servers.

DC-nets. A *Dining Cryptographer* network (DC-net) [15] allows for anonymous broadcast. DC-nets commonly use secret-sharing to obscure the source of data in the network. As in prior work [1, 18, 28], we instantiate a DC-net with two (or more) servers and many clients.

Toy Protocol. Consider a setting with two (non-colluding) servers ServerA and ServerB and two or more clients, where one of the clients wishes to share a file. In a two-server DC-net, the ith client samples a random bit string r_i and sends secret share $r_i \oplus m_i$ to ServerA and secret share r_i to ServerB. Servers can recover m_i by combining their respective shares:

$$m_i = (m_i \oplus r_i) \oplus (r_i).$$

If exactly one of N clients shares a message $m_i = \boldsymbol{m}$ while all other clients share $m_i = 0$, the servers can recover $\boldsymbol{m}$ (without

learning which client sent $m_i = \boldsymbol{m}$) by aggregating:

$$\boldsymbol{m} = \overbrace{\bigoplus_i^N (r_i \oplus m_i)}^{\text{ServerA}} \oplus \overbrace{\bigoplus_i^N r_i}^{\text{ServerB}} = \overbrace{\bigoplus_i^N (\cancel{r_i \oplus r_i}) \oplus \bigoplus_i^N m_i}^{\text{(commutativity of xor)}}$$
$$= \underbrace{0 \oplus \ldots \oplus \boldsymbol{m} \oplus \ldots \oplus 0}_{\text{origin of } \boldsymbol{m} \text{ is hidden}}.$$

This simple scheme protects client anonymity given that each server obtains a uniformly random value. Only the combined shares reveal the resulting message. However, user can undetectably *disrupt* the broadcast by non-zero shares. The main challenge in metadata-private broadcasting is ensuring correctness in the presence of such disruptors [1, 18, 28, 43]. Indeed, preventing disruption by malicious clients is the primary cause of high-latency incurred by prior work [1, 18, 43].

Moreover, we show that the request auditing process in existing systems [1, 18, 28] is susceptible to an active attack by a malicious server (Appendix A). Specifically, we show that such a server can *undetectably* de-anonymize a broadcaster with some probability by intentionally causing a request audit to fail, excluding their messages.

Anonymous Broadcast in Spectrum. Spectrum builds on the above DC-net construction to improve efficiency, prevent disruption, and prevent malicious servers from de-anonymizing users through active attacks. We present Spectrum with two servers and any number of publishers and subscribers. We generalize Spectrum to more than two servers in Section 4.5. In both cases, privacy for Spectrum holds provided at least one of the servers does not collude with any other server, and privacy is guaranteed even in the face of active attacks. We formalize the threat model in Section 3.1.

3 System Overview

Spectrum consists of two or more broadcast servers and many clients. Each client is either a *publisher* or a *subscriber*. Figure 1 outlines a deployment with two servers, one publisher, and three subscribers. In this scenario, the publisher secret-shares a message with the servers while the subscribers secret-share dummy cover messages to prevent network metadata from de-anonymizing the publisher. In Spectrum, all clients prove that the provided messages do not disrupt the broadcast (i.e., that they are behaving according to protocol). We achieve this by designing a blind auditing protocol through which servers ensure that each client's message is well-formed and does not disrupt any broadcast. Servers do not learn whether a client is a subscriber or publisher when auditing secret-shares of the provided message. We describe this in Section 3.2.

3.1 Threat Model

In Spectrum, an adversary may control an arbitrary subset of clients and servers. Only one of the servers needs to be honest (not colluding with the adversary) in order to guarantee anonymity for all honest clients. On the other hand, all clients are completely untrusted by the servers and are assumed to deviate from protocol in arbitrary ways or collude with the adversary.

Guarantees. If any subset of malicious clients deviate from protocol, both anonymity and system availability must remain intact for all *honest* clients.

If a subset of corrupted servers arbitrarily deviate from protocol, anonymity for all honest clients is guaranteed but availability of the system as a whole is not. Likewise, all other system failures may disrupt availability but must not cause deanonymization of any honest client.

Assumptions. As with prior work [1, 18, 43], we assume that the adversary has full view of the network but does not interfere with network traffic. To hide network traffic *contents*, we make black-box use of public key infrastructure (e.g., TLS [63]) for encrypted communication between clients and servers; such infrastructure is widely deployed. Finally, we rely on the discrete logarithm assumption for the purpose of protecting against colluding servers [27].

3.2 Blind Message Authentication

The main idea behind Spectrum is a form of blind request authentication that can be efficiently verified by the servers to reject malicious clients. We adapt the Carter-Wegman MAC [13, 85] to serve as a secret-shared "authentication tag" accompanying the message shares in the DC-net broadcast of Section 2. The tag proves that a client's request is either: (1) an authenticated broadcast message (i.e., $m_i = \boldsymbol{m}$); or (2) a cover request issued by a subscriber (i.e., $m_i = 0$). Crucially, the servers must not learn whether (1) or (2) is true when auditing the tag for correctness.

Carter-Wegman MAC. Let $\mathbb{F}$ be any finite field (e.g., integers modulo a prime p) of sufficiently large size in a security parameter. Sample an authentication key by randomly picking elements α and γ in $\mathbb{F}$ and defining the function $\mathsf{MAC}_\alpha^\gamma$:

$$\mathsf{MAC}_\alpha^\gamma(m) = \alpha \cdot m + \gamma \in \mathbb{F}.$$

A verifier can check the authenticity of a message m' given a tag t by computing $t' \leftarrow \mathsf{MAC}_\alpha^\gamma(m')$ and checking if $t = t'$. Forgery (different messages $m \neq m'$ but same tag $t = t'$) is infeasible [69]. Moreover, if $\mathbb{F}$ is sufficiently large, the probability of guessing the authentication key (α, γ) is negligible.

Observe that $\mathsf{MAC}_\alpha^\gamma$ is a *linear function* of the message which makes it possible to verify a secret-shared tag for a secret-shared message. That is, for a message $m \in \mathbb{F}$ that is

	Request Size	Audit Size	Audit Rounds	Server Work	Active Security	Comments
Blinder [1]	$\lvert m\rvert \cdot \lambda\sqrt{U}$	$\lambda \cdot \lvert m\rvert$	$\log U$	$U \cdot (\lambda + \lvert m\rvert)$	✓	Requires at least 5 servers and MPC
Riposte [18]	$\lvert m\rvert + \lambda\sqrt{U}$	$\lambda\sqrt{U}$	1	$U \cdot (\lambda + \lvert m\rvert)$	✗	Requires a separate audit server
Express [28]	$\lvert m\rvert + \lambda^2 + \lambda L$	λ	1	$L \cdot (\lambda + \lvert m\rvert)$	✗	Can only be instantiated with two servers
Two-Server	$\lvert m\rvert + \lambda \log(L)$	λ	1	$L \cdot (\lambda + \lvert m\rvert)$	✓	Based on efficient tree-based DPF [9]
Multi-Server	$\lvert m\rvert + \lambda L$	λ	1	$L \cdot (\lambda + \lvert m\rvert)$	✓	Based on seed-homomorphic DPF [7]

Table 1: Asymptotic efficiency comparison between Spectrum and existing anonymous broadcasting systems on a per request basis when instantiated with L publishers, U total users, $|m|$-sized messages, and global security parameter λ. $O(\cdot)$ notation suppressed for clarity.

additively secret-shared as m_A and m_B such that $m_A - m_B = m \in \mathbb{F}$ and a MAC tag t secret shared as t_A and t_B such that $t_A - t_B = t$, the servers (knowing α and γ) can verify that the tag corresponds to the secret-shared message as follows:

1. ServerA computes $\beta_A \leftarrow (\alpha \cdot m_A + \gamma - t_A)$.
2. ServerB computes $\beta_B \leftarrow (\alpha \cdot m_B - t_B)$.
3. Servers swap β_A and β_B to locally check if $\beta_A \stackrel{?}{=} \beta_B$.

The final condition only holds for a valid tag.

Observe that neither server learns anything about the message m in the process (apart from the tag validity) since both the message and tag remain secret-shared between servers. Note that $\mathbb{F}$ can be any finite field, thus the DC-net example of Section 2, where users send xor-based secret-shares, is just a special case of the additive secret-sharing in $\mathbb{F}$ (any finite field, for example, integers modulo a prime p).

That these properties are *almost* sufficient to prevent disruption in the DC-net described in Section 2. If both the servers and the publisher have access to the key (α, γ), the publisher can compute a tag t which the servers can check for correctness as above.

However, the approach does not allow *subscribers* to send cover messages as they must also generate a tag for their message $m_i = 0$, with no authentication key. If subscribers do not authenticate cover messages, servers could distinguish publishers and subscribers, but giving this key to subscribers would defeat the purpose of authentication.

To overcome this, we make the following observation which is inspired by a technique in the SPDZ [21] multi-party computation protocol. The Carter-Wegman MAC has key (α, γ) where γ acts solely as a "nonce" to prevent forgeries on the value $0 \in \mathbb{F}$ [84]. Because of this, we can eliminate γ while still having the desired unforgeability property of the original MAC for all non-zero messages. Thus, when evaluated over secret shares,

$$\mathsf{MAC}_\alpha(m) = \alpha \cdot m$$

remains as secure as the original $\mathsf{MAC}^\gamma_\alpha$ for all $m \neq 0$. The advantage to us, however, is that this allows subscribers to authenticate the message $m_i = 0$ without knowing α (i.e., subscribers can "forge" the MAC but only for $m_i = 0$).

This makes MAC_α sufficient to prevent disruptors: a publisher can secret-share a non-zero message and valid tag using the broadcast key α while a subscriber can only secret-share the message $m_i = 0$ (and corresponding tag) without knowledge of the authentication key. Any deviation to this would result in an invalid tag which will be caught by the servers when performing the above audit.

Preventing collusion. One final issue with this message authentication technique is that all servers (including the malicious ones) receive the MAC key *in the clear*. This makes the scheme vulnerable in the case that a malicious server colludes with a malicious client to disrupt the protocol by sharing the broadcast key with the client. We resolve this by shifting the entire MAC to the exponent of a group where the Discrete Logarithm (DL) problem is assumed to be hard [77]. This way servers only obtain g^α and can proceed to verify the MAC as before. We explain this further in Section 4.3.

3.3 Spectrum Functionality

We now present the functionality of Spectrum assuming two servers, but note that our definitions generalize to any $n \geq 2$ number of servers (see Section 4.5).

Let N be the total number of clients, of which L are publishers broadcasting messages $m_1, \ldots, m_L$, respectively. Each publisher is assigned to an oblivious "channel" (servers cannot tell when a message is being written to a channel). For our purposes, a channel is simply a slot with an associated broadcast key. Both publishers and subscribers "write" to all channels to make requests indistinguishable to the servers. However, only a publisher, knowing the channel authentication key, can write a non-zero message to the associated channel. That is, channel $j \in [L]$ has an authentication key α_j known to the publisher of the channel (but not to any other client). An immediate problem to overcome is that of key distribution: How do publishers obtain the secret broadcast key for their channel?

Channel Key Distribution. In Spectrum, a publisher must "register" with the servers by giving them an authentication key. However, the servers must not learn the identity of the publisher when receiving this key, which leads us to a

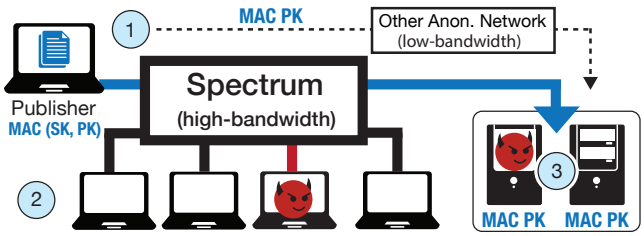

Figure 2: ① A publisher creates a new "channel" by sending a broadcast key to the servers via a (slow, low-bandwidth) anonymous network (e.g., Riposte [18]). ② The publisher can then broadcast using Spectrum (fast and high-bandwidth). ③ Even if a malicious client attempts to disrupt, the servers (blindly) verify that each client request is valid using the broadcast key—only the publisher can write to the channel.

somewhat circular problem: publishers need to anonymously broadcast a key in order to broadcast anonymously.

We solve this one-time setup problem as follows (illustrated in Figure 2). All clients use a slower anonymous bulletin board suitable for low-bandwidth content at system setup time. There are many different ways to realize this functionality using existing anonymous broadcasting systems (e.g., [1, 18, 43]). Keys are small (e.g., 32 bytes) and therefore using existing systems to anonymously share the key is not challenging. Moreover, once the keys for publishers are established, they may be used indefinitely by the publishers to broadcast on their respective channels.

Client Algorithms. Each client must generate a "write" request to the servers. The request must not reveal whether the client is broadcasting or only generating cover traffic. Servers aggregate the requests from all clients to ensure the origins of the broadcasts remain hidden among the set of all honest clients. To this end, both publishers and subscribers generate indistinguishable "request shares" (τ_A, τ_B). The clients send these shares to ServerA and ServerB, respectively. The servers then blindly audit the requests for correctness (check that the message has a valid tag) and aggregate all requests that pass the audit.

Broadcast$(m, j, \alpha_j) \to (\tau_A, \tau_B)$**.** Generates request shares which write message m to the jth channel.

Cover$() \to (\tau_A, \tau_B)$**.** Generates cover request shares, indistinguishable from shares output by Broadcast but resulting in no change to any channel.

Remark 1. *We assume that subscribers can download broadcasts made available through Spectrum directly from the servers since all broadcasts are made public at the end of each round and thus do not require the access key to download. While privacy for subscribers (hiding which broadcast is fetched) is orthogonal to our main goal, we do describe such an extension in Appendix B.*

Server Algorithms. Servers collaboratively audit the request shares received from each client, then combine all requests together to reveal the L messages, ensuring the provenance of each broadcast is hidden. To audit a request sent by a client, each server locally generates an *audit token* and shares it with the other server. Together, the servers audit the request using the provided token in a way that does not reveal information on the request beyond its validity.

GenAudit$(\Gamma, \tau_\circ) \to \beta_\circ$**.** Prepares an audit token for the given client request agains the broadcast authentication keys Γ. Servers then swap the locally computed tokens β_A and β_B with each other.

CheckAudit$(\beta_A, \beta_B) \to$ yes **or** no**.** Validates the result of GenAudit.

Accept$(\tau_\circ) \to \boldsymbol{m}_\circ$**.** Maps a request token to a vector of L secret-shares $\boldsymbol{m}_\circ$, which can be aggregated by the server.

Recover$(\boldsymbol{m}_A, \boldsymbol{m}_B) \to \boldsymbol{m}$**.** Locally, aggregate shares of Accept outputs (one per client). Globally, recovers all messages. We use $\boldsymbol{m}_A \oplus \boldsymbol{m}_B$ as shorthand for Recover, since we implement it with XOR.

We now define the informal properties required for the above algorithms to achieve the security requirements of Section 3.1 with the caveat that both servers must follow protocol correctly when auditing a client request. In Section 3.3 we show how to ensure anonymity for honest clients in the face of an actively malicious server tampering with the auditing process. A more formal description of the protocol properties and requirements can be found in Appendix C.

Correctness. Provided all honest publishers broadcast messages m_1 through m_L and all honest subscribers broadcast cover messages, the protocol recovers all broadcast messages $\boldsymbol{m} = (m_1, \ldots, m_L)$ as output, one message per channel.

Completeness. With valid channel authentication and verification keys, requests generated with Broadcast pass the audit. Cover generates cover messages that pass the audit without needing any authentication key.

Soundness. Without the channel authentication key, it should be computationally infeasible to generate a request that writes a non-zero message to the associated channel.

Privacy. If at least one server is honest, an adversary controlling the other server (but not deviating from protocol) and a subset of malicious clients (possibly deviating from protocol) should learn nothing more from an honest client request other than that the request is well-formed.

These properties are guaranteed by Spectrum provided both servers follow protocol and do not collude with each other. We now describe the functionality of BlameGame used to achieve malicious security.

BlameGame Functionality. We cannot eliminate the requirement for abort in the case of a malicious server, which can always refuse to participate, disrupting availability. But if a client deviates, the servers can proceed. BlameGame decides which of these is the case.

If a client audit in Spectrum fails, then servers double-check each others' work to prevent a malicious server from selectively rejecting requests to reduce the anonymity set over time via a guess-and-check attack (see Appendix A). Each server sends a client-generated "dispute token" $\sigma_\circ$ to the other servers. All servers then use the following functionality to assign blame. If at least one server is honest, then the protocol aborts if another server cheats.

$\mathsf{Resolve}(\mathsf{sk}_\circ, \Gamma, \sigma_\circ) \to \beta'_\circ$**.** A server resolves a dispute $\sigma_\circ$ sent by the other server with Resolve, which outputs a backup client audit token $\beta'_\circ$.

$\mathsf{Absolve}(\mathsf{sk}_\circ, \sigma_\circ) \to \phi_\circ$**.** If the backup audit fails, an honest server can absolve itself by sending an acquittal token $\phi_\circ$ to the other server, which proves adherence to the audit protocol (blaming the client).

$\mathsf{Blame}(\Gamma, \tau_\circ, \sigma_\circ, \phi_\circ) \to \mathsf{server/client}$**.** Assigns blame to either a server or the client based on the request $\tau_\circ$, dispute token $\sigma_\circ$, and acquittal token $\phi_\circ$.

In the case that Blame outputs server, Spectrum is aborted by the honest server. The above functionality, used by the servers as depicted in Figure 4, must satisfy the following properties (formalized in Appendix D).

Completeness. If both servers and clients follow protocol, Resolve outputs $\beta'_\circ$ such that CheckAudit outputs yes. Otherwise, Blame outputs server or client to indicate which party deviated.

Soundness. It must be computationally difficult for a malicious client to generate a malicious request token τ and dispute token σ such that Blame does not output client (i.e., a malicious client should be unable to "frame" an honest server as being malicious) or vice versa.

Privacy. Dispute and Resolve should reveal no information about a client request. Absolve and Blame may reveal the client request (for checking server protocol compliance) which results in a one-shot de-anonymization of an honest broadcasting client with probability at most $\frac{L}{N(1-\epsilon)}$, where N is the total number of users, L is the number of publishers, and ϵ is the fraction of corrupted clients.

4 Spectrum Protocol

In this section we describe the design and architecture of Spectrum. We first show how a *distributed point function* (DPF) [33] can support many broadcast channels with little increase in bandwidth overhead (compared to the setting with only one channel). We prevent disruption by augmenting these DPFs with an extension of the message authentication technique in Section 3.2. Prior works [9, 10, 18, 28] describe techniques for verifying that a DPF is well-formed but do not provide a way to enforce access control as required for our purposes. In Spectrum, we simultaneously enforce message authentication *and* guarantee that the DPF is correctly encoded for each client.

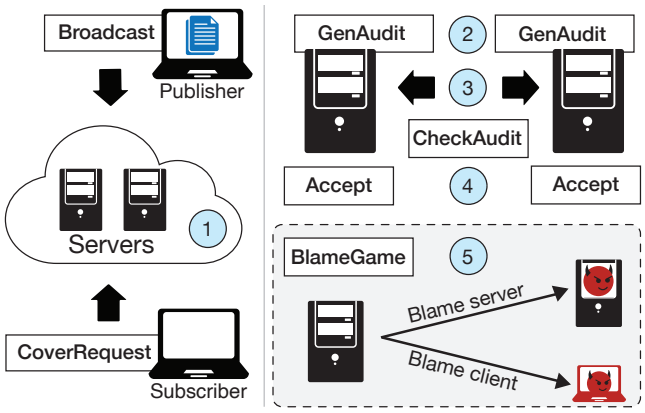

Figure 3: ① Clients send encoded requests to the servers (Broadcast/Cover). ② The servers share "audit tokens" for each client's request (GenAudit). ③ Servers locally validate the tokens (CheckAudit). ④ Servers process the request (Accept). ⑤ If servers or clients misbehave, BlameGame assigns culpability.

We begin by providing a formal definition for a DPF and then explain how we adapt the message authentication technique of Section 3.2 to the multi-channel setting.

4.1 Distributed Point Functions

A *point function* P is a function that evaluates to a message m on a single input j in its domain and evaluates to zero on all other inputs $i \neq j$. In Definition 1, we describe a *distributed* point function: a point function that is encoded and secret-shared in a set of n keys.

Definition 1 (Distributed Point Function (DPF) [18, 33])**.** *Fix positive integers L and $n \geq 2$. An n-DPF consists of (possibly randomized) algorithms:*

- $\mathsf{Gen}(m \in \{0,1\}^\ell, j \in [L]) \to (k_1, \ldots, k_n)$
- $\mathsf{Eval}(k_i) \to \boldsymbol{m}_\circ$

These algorithms must satisfy the following properties:

Correctness. *A DPF is* correct *if the sum of the results of evaluating all keys produced by $\mathsf{Gen}(m, j)$ is a vector $\boldsymbol{m}$ in $(\{0,1\}^\ell)^L$ such that $\boldsymbol{m}_j = m$ and $\boldsymbol{m}_{j'} = 0$ for $j' \neq j$:*

$$\Pr\left[\begin{array}{l}(k_1,\ldots,k_n) \leftarrow \mathsf{Gen}(m,j) \\ s.t.\ \sum_{j=1}^{n} \mathsf{Eval}(k_j) = m \cdot \boldsymbol{e}_j\end{array}\right] = 1$$

where the probability is over the randomness of Gen *and $\boldsymbol{e}_j \in \{0,1\}^L$ is the jth row of the $L \times L$ identity matrix.*

Privacy. *A DPF is* private *if fewer than n evaluation keys created by* $\mathsf{Gen}(\cdot,\cdot)$ *do not reveal any information about the inputs. Let $I \subset [n]$ be any strict subset of indices, and D_I be the distribution over $\{k_i \mid i \in I\}$ where the k_i are sampled as*

$$(k_1,\ldots,k_n) \leftarrow \mathsf{Gen}(m,j)$$

for fixed m and j. Then, there exists a probabilistic polynomial time (PPT) algorithm Sim *such that $D_I \approx_c \mathsf{Sim}(I)$.*

Efficiency. There exist efficient constructions for 2-DPFs where the key size is $O(\lambda \log L + |m|)$ [9, 10]. However, in the general case ($n > 2$) it is not known how to achieve smaller key sizes [9]. While there exist sublinear constructions for n-DPFs [9], the key size is *exponential* in n with a multiplicative factor of $|m|$. Because in our case L is small, for the case where $n \geq 3$, we tolerate a linear key size in L with an additive dependence on $|m|$. We describe such a construction in Appendix F.

DPFs for Broadcasting. To broadcast a message m to channel j, a publisher computes $\mathsf{DPF.Gen}(m,j)$ and sends the output to the servers. A subscriber runs $\mathsf{DPF.Gen}(0,\cdot)$ and does likewise. The servers sum the results of Eval on each key received by a client (each server gets one of the generated DPF keys) to locally recover secret-shares of a message destined for each channel. When the accumulated values are combined, only the message m remains on the jth channel (and 0 is written to other channels).

4.2 Auditing Bandwidth Optimization

When used for private writing, a point function can be viewed as one operation (i.e., write) at a specific index and a sequence of "zero-writes" at all other indices; a DPF efficiently secret-shares these operations. Each "write" might be large (ℓ bits); for the audit, rather than checking the *whole value* of the write, we need only check whether it writes to any unauthorized index. This allows the servers to broadcast $O(\lambda)$ bits to verify the request, rather than $O(\lambda \cdot |m|)$.

Specifically, we can define a function SmallEval that evaluates a DPF key, but into a smaller message space. SmallEval is an efficiently computable mapping from DPF keys to additive secret shares of an L-vector $\boldsymbol{\rho} = (\rho_1,\ldots,\rho_L)$ in some field $\mathbb{F}$ such that $\rho_j = 0$ if and only if the DPF input at index j was zero for all $j \in [L]$). We give a construction of SmallEval alongside our DPF construction in Appendix F. Then, we can check that, for each j, $\rho_j = 0$ *or* the client knows the corresponding broadcast key α_j, applying the Carter-Wegman MAC (Section 3.2).

Let $\boldsymbol{\alpha} = (\alpha_1,\ldots,\alpha_L) \in \mathbb{F}^L$ be the respective broadcast keys of the L channels. Then, the servers can check that $\langle \boldsymbol{\alpha}, \boldsymbol{\rho} \rangle = 0$ ($\langle \cdot,\cdot \rangle$ denotes the inner product). When all $\alpha_j = 0$, this holds trivially. When some $\alpha_j \neq 0$, the user needs α_j to "cancel" the non-zero value. Because the servers hold secret-shares of $\boldsymbol{\rho}$ and plaintext values of $\boldsymbol{\alpha}$, computing the inner product is a linear operation. That is, the servers can compute additive secret-shares of $\langle \boldsymbol{\alpha}, \boldsymbol{\rho} \rangle$ *locally*. They can then reveal these shares and check that they are zero; this value is the same for all honest clients, so the servers learn nothing. In the following section, we show how to perform this check without revealing the keys to the servers.

4.3 Collusion-Secure Blind Authentication

In this section we show how to apply the MAC verification of Section 3.2 "in the exponent" of a group in which the Discrete Logarithm (DL) problem is assumed to be hard [77]. We do so to prevent a malicious server from sharing the broadcast key of a channel with a malicious client (which a malicious server can do if the broadcast key is given to it in the clear). Specifically, let $\mathbb{G}$ be a group of prime order p with generator g such that the DL problem in $\mathbb{G}$ is computationally intractable. That is, for a random element $y \in \mathbb{G}$, finding u such that $y = g^u$ is infeasible in polynomial time.

A publisher, instead of sending α to the servers at setup time, sends g^α – a "public verification key" – which servers can use to audit the MAC tag as before, but this time without having $\boldsymbol{\alpha}$ in the clear. This ensures that a malicious server colluding with a client cannot covertly disrupt a broadcast since doing so would imply that the server was able to recover the discrete logarithm of g^α. We provide a formal analysis of security in Appendix C.

4.4 Protocols

Using a DPF we can instantiate Spectrum with multiple channels while incurring very little bandwidth overhead on clients and servers. The full multi-channel instantiation of Spectrum is provided in Algorithm 1 and uses the DPF construction found in Appendix F. Clients generate a write request by running DPF.Gen, providing the index j to broadcast on the jth channel. To convince the servers that the DPF expands to the correct channel (i.e., the generated keys k_A and k_B are correct), the client generates a tag t by evaluating $\boldsymbol{\rho_A} \leftarrow \mathsf{SmallEval}(k_A)$ and $\boldsymbol{\rho_B} \leftarrow \mathsf{SmallEval}(k_B)$, then computing: $t = \alpha \cdot (\boldsymbol{\rho_A}[j] + \boldsymbol{\rho_B}[j])$. This value is secret-shared to the servers and is used to authenticate the request. To do so, each server evaluates SmallEval on their DPF key and subtracts the share of t. We provide a full security analysis for Algorithm 1 in Appendices C and C.

Theorem 1 (Security)**.** *The construction in Definition 1 satisfies the* correctness*,* completeness*,* soundness*, and* privacy *properties for Algorithm 1 defined in Appendix C.*

Proof. See Appendix C. □

Theorem 2 (Communication)**.** *Each request share generated with* Broadcast *and* Cover *is of size $O(|m| + \lambda \log(L))$ in the*

Algorithm 1: Spectrum

Client Algorithms

def Broadcast(m, j, α):
  $(k_A, k_B) \leftarrow$ DPF.Gen(m, j) // generate DPF keys
  $(\rho_A, \rho_B) \leftarrow$ (SmallEval(k_A),SmallEval(k_B))
  $t \leftarrow \alpha \cdot (\rho_A[j] - \rho_B[j])$ // MAC tag
  $(t_A, t_B) \xleftarrow{R} \mathbb{F} \times \mathbb{F}$ s.t. $t_A - t_B = t$ // secret-share tag
  output $(k_A || t_A, k_B || t_B)$ // requests: keys with tags

def Cover():
  $(k_A, k_B) \leftarrow$ DPF.Gen$(0^\ell, 0)$
  $(t_A, t_B) \xleftarrow{R} \mathbb{F} \times \mathbb{F}$ s.t. $t_A - t_B = 0$ // secret-share of 0
  output $(k_A || t_A, k_B || t_B)$

Server Algorithms

def GenAudit$(\Gamma = (g^{\alpha_1}, \ldots, g^{\alpha_L}), \tau_\circ = k_\circ || t_\circ)$:
  $\rho_\circ \leftarrow$ SmallEval$(k_\circ)$ // where $\rho_\circ \in \mathbb{G}^L$
  // DPF key $k_\circ$
  $\beta_\circ \leftarrow (\prod_{i=1}^{L} (g^{\alpha_i})^{\rho_\circ[i]}) \cdot g^{-t_\circ}$ // Tag share $t_\circ$; keys Γ
  output $\beta_\circ$

def CheckAudit(β_A, β_B):
  output yes if $\beta_A = \beta_B$; no otherwise

def Accept$(\tau_\circ = (k_\circ || t_\circ))$:
  output DPF.Eval$(k_\circ)$

two-server setting and $O(|m| + \lambda L)$ in the multi-server setting. Each audit share is of size $O(\lambda)$.

Proof. The DPF key size ($O(|m| + \lambda \log(L))$ for the 2-DPF [10] and $O(|m| + \lambda \log(L))$ with the DPF in Appendix F [18]) dominates the requests of Broadcast and Cover. The audit token is 3 field elements ($O(\lambda)$). □

Theorem 3 (Computation)**.** Broadcast *and* Cover *require $O(L + |m|)$ work on the client.* Accept *and* GenAudit *require $O(L \cdot |m|)$ work on the server.*

Proof. Both the client and server work is dominated by the DPF algorithms. Generating DPF keys requires $O(L + |m|)$ work in the worst case, depending on the construction (see Appendix F). Evaluating the DPF key requires $O(L \cdot |m|)$ work on the server for all constructions [9]. □

BlameGame Protocol. Many anonymity systems face the problem that if a message is *expected* to be broadcast, then a malicious server can drop or modify a client's request to learn whether they were broadcasting that message. To prevent this attack, prior anonymity systems [18, 28] must abort on a bad client request (allowing disruption). Instead, in Spectrum, following an audit failure servers open a "dispute" which the other servers must resolve. If the resolution fails, an honest server assigns blame to either a server (aborting the protocol) or client (dropping the message).

We require a *verifiable* public-key encryption scheme: the encryptor can prove the correct decryption of a ciphertext to a verifier. This is a property of most public-key encryption schemes (e.g., RSA [64] and ElGamal [27]).

The Protocol. In addition to request tokens τ_A and τ_B, each client encrypts and sends $\sigma_A = [\![\tau_B]\!]_{\mathsf{pk}_B}$ to server ServerA and $\sigma_B = [\![\tau_A]\!]_{\mathsf{pk}_A}$ to ServerB. Note that the encrypted requests are swapped, commiting each server to the backup request and audit. If an audit fails in Spectrum (CheckAudit outputs no), servers exchange σ_A and σ_B and decrypt them to retrieve "backup" requests.

Each server proceeds to run Resolve using the received $\sigma_\circ$ and sends $\beta_\circ$ to the other server. Servers then run CheckAudit(β_A, β_B). However, if the second audit fails, blame can be assigned to either the server or the client as follows. Each server runs Absolve to generate an acquittal token $\phi_\circ$ and prove adherence to protocol by revealing the decryption of the token (a malicious server is unable to do so if it deviated from protocol). Each server runs Blame using the received acquittal token $\phi_\circ$ and blames the party responsible for audit failure. In the case that the server is blamed, Spectrum is aborted (by at least one honest server) and no more client requests are processed. Otherwise, the client is blamed and the request is discarded by all servers. We provide formal requirements for Algorithm 2 in Appendix D.

Theorem 4 (Security)**.** *BlameGame satisfies the* completeness, soundness, *and* privacy *properties defined in Appendix D when a server is actively malicious.*

Proof. See Appendix D. □

Theorem 5 (Communication)**.** *BlameGame incurs an overhead of $O(L\lambda)$ bits per request and at most one additional round of communication between servers.*

Proof. Assume that each dispute token $\sigma_\circ$ is posted to a publicly accessible bulletin board. To resolve disputes, servers decrypt their share $\sigma_\circ$ and run Resolve only if the original audit fails (this requires one round of communication). Servers can then post their shares of Absolve to the bulletin board and assign blame through Blame if the secondary audit fails. The size of $\sigma_\circ$ is at most $O(L\lambda)$ when using the multi-server DPF since the audit is only evaluated over the compressed DPF keys (independently of the message m). □

4.5 Multiple Servers

While we focus our presentation on the two server setting, the underlying building blocks trivially generalize to multi-server settings as well, where we allow an adversary to control all

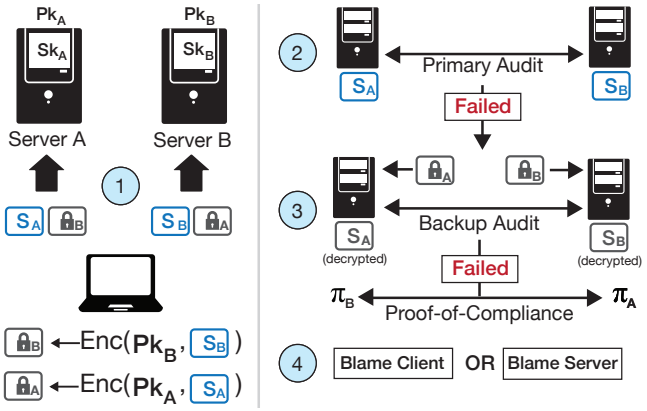

Figure 4: BlameGame protocol. ① Clients sends an encrypted backup of their shares to the *opposite* server. ② Servers audit the secret-shared requests (from Spectrum). ③ If the audit fails, servers swap the backups and perform a second audit over the (decrypted) shares. ④ If *this* audit fails, then blame is assigned to either the client (and the request is discarded) or the server (and the honest server aborts).

but one server. In Appendix F, we provide a construction for a general, multi-server DPF similar to the one used in Riposte. We note that the authorization technique of Section 4.3 applies to this general construction as well. However, instantiating Spectrum with more than two servers requires switching the underlying cryptographic primitives in a DPF [18]. The multi-key DPF presented in Appendix F uses a seed-homomorphic PRG, which is much slower compared to an AES-based PRG used in the two-key DPF construction [9, 10]. We implement and show experimental results demonstrating this effect in Section 5.

5 Evaluation

We implement and evaluate Spectrum, comparing it to the state-of-the-art protocols for anonymous broadcasting: Riposte [18], Blinder [1], and Express [28] (see a full comparison in Section 6).

Riposte is designed for anonymous broadcasting but does not have a concept of *channels*: all users are assumed to be broadcasting all the times. Riposte can be instantiated with a minimum of 3 servers (of which one is an audit server) but generalizes to a many-server setting, where at least one server is assumed to be honest. Riposte was designed for smaller messages and the source code fails to run with messages of size 5 kB or greater.

Blinder builds on Riposte but requires at least 5 servers of which a majority is assumed to be honest. Like Riposte, Blinder also assumes that all users are broadcasting and does not have a concept of channels. However, Blinder is designed to take advantage of a server-side GPU to increase throughput

Algorithm 2: BlameGame

Params : $\mathcal{E} = (\text{Gen}, \text{Enc}, \text{Dec}, \text{DecProof}, \text{VerProof})$.

def Resolve$(\text{sk}_\circ, \Gamma, \sigma_\circ)$:
- parse $\Gamma = (g^{\alpha_1}, \ldots, g^{\alpha_L})$ // verification keys
- parse $\sigma_\circ = [\![\tau_\circ]\!]_{\text{pk}_\circ}$
- $\tau_\circ \leftarrow \text{Dec}(\text{sk}_\circ, [\![\tau_\circ]\!]_{\text{pk}_\circ})$ // decrypt backup request
- $\beta_\circ \leftarrow \text{GenAudit}(\Gamma, \tau_\circ)$
- **output** $\beta_\circ$

def Absolve$(\text{sk}_\circ, \sigma_\circ = ([\![\tau_\circ]\!]_{\text{pk}_\circ})$:
- $\tau_\circ \leftarrow \text{Dec}(\text{sk}_\circ, [\![\tau_\circ]\!]_{\text{pk}_\circ})$ // decrypt backup request
- $\pi_\circ \leftarrow \text{DecProof}(\text{sk}_\circ, [\![\tau_\circ]\!]_{\text{pk}_\circ})$ // proof of decryption
- $\phi_\circ \leftarrow \tau_\circ \| \pi_\circ$ // backup request/decryption proof
- **output** $\phi_\circ$

def Blame$(\Gamma, \tau_\circ = (k_\circ \| t_\circ), \sigma_\circ = ([\![\tau'_\circ]\!]_{\text{pk}_\circ}), \phi_\circ = (\tau'_\circ \| \pi_\circ))$:
- parse $\phi_\circ = \tau'_\circ \| \pi_\circ$
- **if** $\text{VerProof}(\text{pk}_\circ, \pi_\circ, [\![\tau'_\circ]\!]_{\text{pk}_\circ}, \tau'_\circ) = \text{no}$ **then**
  - **output** server // incorrect decryption proof
- $(\beta_\circ, \beta'_\circ) \leftarrow (\text{GenAudit}(\Gamma, \tau_\circ), \text{GenAudit}(\Gamma, \tau'_\circ))$
- **if** $\text{CheckAudit}(\beta_\circ, \beta'_\circ) = \text{yes}$ **then**
  - **output** server // server tampered with audit
- **else**
  - **output** client // client sent an invalid request

and achieves a concrete advantage over Riposte.

Express [28] is an anonymous communication system designed for anonymous "dropbox"-like applications. While not designed for anonymous broadcasting, we find that it can be easily modified to do so (but with much weaker security guarantees; see Section 6). We include Express in our comparison (after modifying it for broadcasting) because, to the best of our knowledge, it is the only existing system that is capable of decoupling publishers and subscribers if used for anonymous broadcasting.

We run Express [28] and Riposte [18] in the same setting as Spectrum[1]. For Blinder we include performance numbers as reported in their paper [1] because their released source had multiple syntax errors and did not compile for us.

5.1 Setup

Implementation. We implement Spectrum[2] in approximately 8,000 lines of Rust code (2020-12-21 nightly build). We instantiate our construction with AES-128 (CTR mode) for the PRG and BLAKE3 [57] for collision-resistant hashing.

[1] Our source repository includes Terraform [35] deployment templates for all three systems for easy reproduction of these experiments.

[2] Source available at http://github.com/znewman01/spectrum-impl

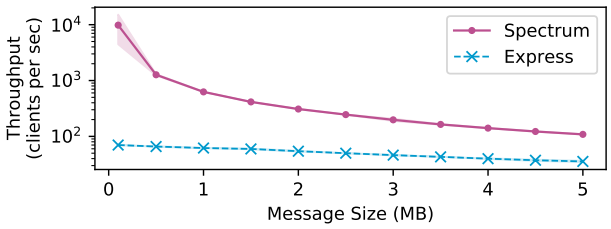

Figure 5: Throughput (client requests per second; higher is better) for a one channel deployment (one publisher and many subscribers). Shaded region represents 95% confidence interval.

For the multi-server extension (Section 4.5), we construct a seed-homomorphic PRG [7] with the Jubjub [38] twisted Edwards curve. All traffic between the client and servers is encrypted using TLS 1.3 [63].

Environment. We use Amazon Elastic Cloud Compute (EC2) for our experiments. Our deployment comprises two or more server machines, along with machines to simulate client traffic. Each is a `c5.4xlarge`[3] virtual machine (VM) instance with an 8-core Intel Xeon Platinum 8000 CPU and 32 GiB RAM, running Ubuntu 20.04 LTS. While comparatively small (only 8 cores), we choose these VMs as are they represent mid-tier commodity servers. Each VM cost $0.68 per hour as of February 2021. Running `openssl speed` reports 3.5 GiB/s throughput for CTR mode AES-128. All servers run in the same region, with 1 ms network RTT.

5.2 Results

Across all settings in which we evaluate these systems, we find Spectrum is 3–140× faster than Express, 0.3–8.5× the speed of Blinder (GPU), 1.3–54× faster than Blinder (CPU), and 328× faster than Riposte.

One channel. We report the throughput (client requests per second) for both Spectrum and Express in the one-channel setting in Figure 5. As expected, throughput scales inversely with the message size for both Spectrum and Express. However, we find that Spectrum, compared to Express, is up to 140× faster on both small (100 kB) messages and up to 3× faster on large (5 MB) messages. Riposte and Blinder have no analog for the single-channel setting.

Many channels. The throughput for both Riposte and Blinder depends only on the total number of users. Therefore, to adequately compare with Spectrum, we first fix the total number of users to 10,000 (increasing to a larger anonymity set size is less favorable to Riposte and Blinder) and vary the number of channels for both Spectrum and Express to 10,000 (which represents the worst case setting for these systems). Even with many simultaneous broadcast channels, Spectrum

[3] See https://aws.amazon.com/ec2/instance-types/c5.4xlarge

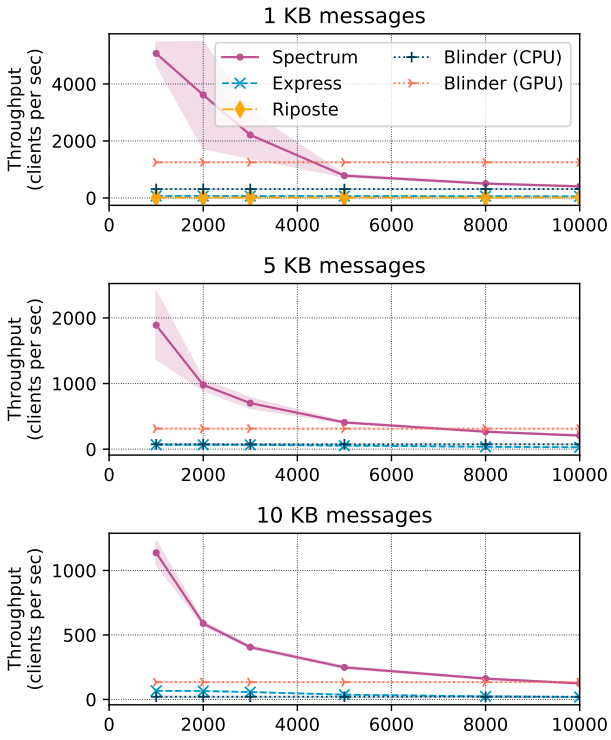

Figure 6: Throughput (requests per second; higher better) for broadcasts with 10,000 users: Express and Spectrum benefit from having relatively few channels. (Blinder numbers as reported by the authors.) Shaded region represents 95% confidence interval.

outperforms Express, Riposte, and Blinder (both CPU and GPU variants) in terms of throughput (see Figure 6). However, as the number of channels increases and approaches the total number of users in the system (10,000), we see a diminishing advantage in performance. As the number of channels reaches the total number of users Spectrum (and Express) perform slightly worse compared to Blinder's GPU deployment (which is an order of magnitude faster than Blinder's CPU variant and Riposte). Therefore, in a setting where every user is broadcasting to their own channel simultaneously, there is no net benefit to using Spectrum. However, we note that most real-world applications have a high ratio of passive subscribers to active publishers [53, 87], which makes Spectrum appealing in most situations.

Overhead. In any anonymous broadcast scheme, every client (even subscribers) must upload data corresponding to the message length $|m|$ to ensure full metadata privacy. For DC-net based schemes, the client sends a size-$|m|$ request to each server. We measure the concrete request sizes of Spectrum and compare to this baseline in Table 2. Client request overhead is small: about 70 B which is roughly 75× smaller compared to request sizes in Express. Moreover, in Spectrum, request audits are under 100 B which is a 120× improvement

Request Size	**Request** per client	**Audit** per client	**Aggregation** once per server
	$\|m\|$+ 70 bytes	70 bytes	$\|m\|$+ 3 bytes
BlameGame (per failed audit)	**Backup Request** per client	**Audit** per client	**Decryption** once per client
	140 bytes	200 bytes	10 µs

Table 2: Upper bound on concrete request size (bytes) for one channel and messages of size $|m|$. BlameGame only incurs an audit and decryption overhead if the first request audit fails (i.e., the BlameGame protocol is invoked).

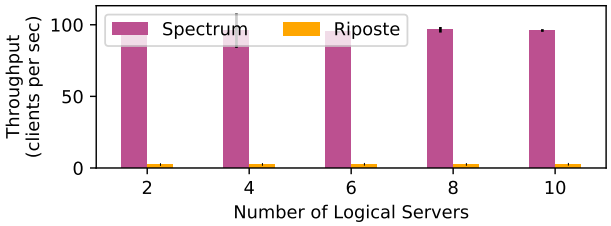

Figure 7: Throughput (higher is better) for n-server deployments with 64 channels of 160 B messages. (Riposte numbers are as reported by the authors for a similar setting.)

over audits in Express [28].

We also find that BlameGame imposes little overhead (both in terms of bandwidth and computation) over Spectrum. Moreover, because BlameGame is only invoked when a request audit fails, these overheads are only incurred for a small fraction of requests (provided servers are honest).

Many servers. We compare the 2-server and n-server versions of Spectrum (where all but one server is assumed to be malicious). Under this threat model, the n-server variant requires the use of a seed-homomorphic PRG [7] to instantiate the DPF (Definition 1), which in turn requires elliptic curve operations (rather than AES). This requires using public-key, rather than symmetric, cryptographic operations to evaluate the PRG resulting in a concrete performance hit (Figure 7). Specifically, we observe a 200× decrease in throughput in a direct comparison to the AES PRG. However, we note that this concrete performance factor remains constant even as the number of servers increases and outperforms the n-server deployment of Riposte.

Scalability. In practical settings, one logical "server" may be deployed as several worker servers. Since each worker server within a logical server is likely to be in the same location and network, running the same software, administered by the same organization, we may trust them identically. In such a deployment, Spectrum can shard client requests across workers composing a logical server. That is, running 10 workers per logical server leads to a 10× increase in overall

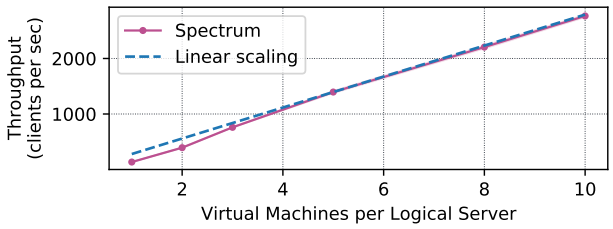

Figure 8: Spectrum is embarassingly parallel: for 500 channels of 100 kB messages, 10 VMs per "server" gives a 10× speedup.

throughput. One benefit of this is that in a cloud deployment, a fixed workload (i.e., number of clients, channels, and message size) can be processed in less time for *negligible additional cost* by parallelizing the servers.

5.3 Discussion

Our evaluation and results showcase the utility of Spectrum on a real-world anonymous broadcasting deployment using only commodity servers. Compared to the state-of-the-art in anonymous broadcasting, Spectrum achieves speedups in settings where the ratio of passive subscribers to publishers is large. We calculate the below times and costs based on throughput observed in our evaluation.

Case study I: Leaking a document. Our deployment of Spectrum can be used to share a small PDF document (1 MB) in 10s within an anonymity set of 10,000 users.

Case study II: Publishing a podcast. Another application of Spectrum is to broadcast a podcast file (50 MB), potentially from a region of the world where internet traffic is monitored and journalistic activities supressed. Our deployment can be used to achieve this in 8m30s with 10,000 users.

Case study III: Publishing a documentary. Finally, we examine the case where Spectrum is used to broadcast a documentary movie (e.g., Navalny's documentary on Putin's Palace [72]). This 2h documentary, available on YouTube, is under 500 MB, requiring roughly 1h24m to upload through Spectrum with an anonymity set size of 10,000.

We note that all of these times can be reduced by parallelizing the logical servers (Figure 8).

Operational costs. To compute cost, we note that for EC2 inbound data is free, and our only outbound data requirement is under 100 bytes per query (about 1 GB per day, at the above rates). Compute costs are $3.85 per GB published through Spectrum (for 10,000 users). We compare with other systems (Case study III, above) in Figure 9. For a cloud-based deployment, parallelizing yields better throughput at almost no additional cost.

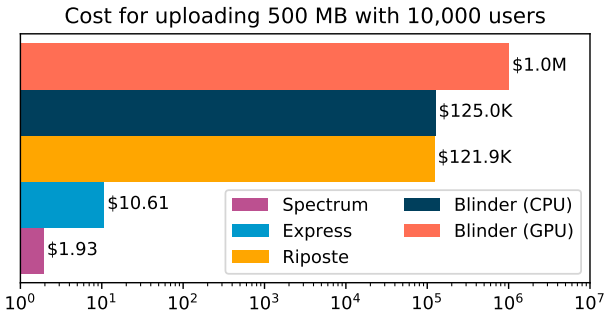

Figure 9: Estimated costs for Spectrum and other systems. For Express and Riposte, we extrapolate from our measurements; for Blinder, we use their reported costs.

Additional extensions. In Appendix B, we discuss two extensions which we believe can be useful to a real-world deployment. First, we show how to incorporate *content moderation* mechanisms to prevent harmful or copyrighted content from being published through Spectrum (a feature that is often at odds with anonymity in other systems but not for Spectrum due to the concept of channels). Second, we show how to provide *subscriber* anonymity when fetching published content by using private information retrieval.

6 Related Work

We provide a comparison to related work in anonymous broadcasting in Table 1. Existing systems for anonymous broadcast are suitable for 140 B to 40 kB [1, 18, 28] broadcasts, which is 2–8 orders of magnitude too small when considering the gigabytes of text documents [68] and multi-terabyte data dumps [59, 67] common today. Anonymity systems such as Tor [25] allow for greater throughput but fail to provide strong anonymity: if only the whistleblower uploads terabytes of data through Tor, metadata (visible to any Tor nodes and network attackers) can be used to uniquely identify them (which is why Tor operators discourage high-bandwidth applications such as BitTorrent [24]).

Mix Networks and Onion Routing. In a mix net [14], users send an encrypted message to a proxy server, which forwards these messages to their destinations in a random order. By chaining several such servers (with encryption between hops), the network protects users from a subset of compromised proxy servers and provides anonymity in the presence of a passive network adversary. Mix nets and their variations [22, 43–46, 48, 49, 52, 54, 61, 62, 73, 74, 82] facilitate anonymous communication but are generally considered slow. To boost performance (by sacrificing strong anonymity), some systems use onion routing instead of a mix net. In onion routing, users encrypt their messages several times (in onion-like layers) and send them to a chain of servers. Tor [25], the most popular onion routing system, anonymizes web traffic with millions of daily users [75]. Tor provides security in many real-world settings, but is vulnerable to traffic analysis. State-of-art attacks [42, 51, 70] de-anonymize users over 90% accuracy. The high-bandwidth broadcast setting is particularly vulnerable: if only one user sends large volumes of data, an adversary can identify them by bandwidth usage alone.

DC-nets. Another group of anonymous communications systems use techniques from the dining cryptographer networks (DC-nets) [15] mentioned in Section 2. We note in Section 2 that DC-Nets are vulnerable to disruption: any malicious participant can clobber a broadcast by sending a "bad" share. Dissent [19] augments the dining cryptographers technique for anonymous group messaging with a system for accountability. Dissent supports only about 40 users; Dissent in Numbers [86] uses an *any-trust* model to handle up to 5,000 participants. While Dissent can support relatively large messages (up to 16 MB), latency exceeds one hour per round.

Riposte [18] enables anonymous broadcast (in the style of Twitter) with many users. Riposte uses a DC-net based on DPFs and an auditing server for preventing disruptors. However, Riposte requires 11 hours to process broadcasts with 1 million users and 160 byte messages and assumes all users are broadcasting.

A more recent work, Blinder [1] extends Riposte with multi-party computation for disruption prevention. Blinder's threat model differs from both Spectrum and Riposte by requiring an honest majority of servers and a minimum of five servers to run. Like Spectrum, Blinder is resilient to active attacks by a malicious server.

Express [28] is a system designed for "mailbox" anonymous communication (writing anonymously to a designated mailbox). Express also uses DPFs for efficient write requests. However, Express can only be instantiated in a two-server deployment and does not generalize to multiple servers using existing cryptographic techniques. Express is *not* a broadcasting system, and while it is possible to adapt it to work in a broadcast setting (as we have done), it is not designed to withstand active attacks by the servers and does not account for client-server collusion, making it relatively insecure for such an application (see Appendix A).

7 Conclusion

We present a new system for achieving anonymous broadcast with strong security guarantees. Spectrum supports high-bandwidth, low-latency transmission from a small set of publishers to a large set of subscribers by introducing a way to obliviously enforce access control to broadcast channels. Our main construction uses only symmetric-key primitives which ensures efficiency in practical deployments. Our experimental results show that Spectrum can be used for uploading gigabyte-sized documents anonymously among 10,000 in a

matter of hours. Moreover, we achieve this on commodity hardware available at scale, making Spectrum a practical tool for anonymous broadcasting with plausible applications in the real world.

8 Acknowledgments

We thank Kyle Hogan, Albert Kwon, and Henry Corrigan-Gibbs for helpful feedback and discussion on early drafts of this paper.

References

[1] I. Abraham, B. Pinkas, and A. Yanai. Blinder: Scalable, robust anonymous committed broadcast. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, pages 1233–1252, New York, NY, USA, 2020. Association for Computing Machinery.

[2] C. F. Alford. Whistleblowers and the narrative of ethics. *Journal of social philosophy*, 32(3):402–418, 2001.

[3] R. W. Apple Jr. 25 years later; lessons from the Pentagon Papers. *The New York Times*, 23 June 1996. Accessed: 2020-05-01.

[4] C. Berret. Guide to SecureDrop, 2016.

[5] S. Bhat, D. Lu, A. Kwon, and S. Devadas. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. *Proceedings on Privacy Enhancing Technologies*, 2019(4):292–310, 2019.

[6] D. Boneh, S. Kim, and H. Montgomery. Private puncturable PRFs from standard lattice assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 415–445. Springer, 2017.

[7] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan. Key homomorphic PRFs and their applications. In *Annual Cryptology Conference*, pages 410–428. Springer, 2013.

[8] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. Denial of service or denial of security? In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 92–102, 2007.

[9] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 337–367, Berlin, Heidelberg, 2015. Springer.

[10] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016.

[11] R. Buettner, S. Craig, and M. McIntire. Long-concealed records show Trump's chronic losses and years of tax avoidance. *The New York Times*, 2020 (Accessed 2020-10-27).

[12] B. Burrough, S. Ellison, and S. Andrews. The Snowden saga: A shadowland of secrets and light. *Vanity Fair*, 2014 (Accessed: 2020-10-27).

[13] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[14] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

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

[16] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[17] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the Signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 451–466. IEEE, 2017.

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

[19] H. Corrigan-Gibbs and B. Ford. Dissent: Accountable anonymous group messaging. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 340–350. ACM, 2010.

[20] H. Corrigan-Gibbs and D. Kogan. Private information retrieval with sublinear online time. Technical report, IACR Cryptology ePrint Archive, 2019: 1075, 2019.

[21] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[22] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III anonymous remailer protocol.

In *2003 Symposium on Security and Privacy, 2003.*, pages 2–15. IEEE, 2003.

[23] C. Delmas. The ethics of government whistleblowing. *Social Theory and Practice*, pages 77–105, 2015.

[24] R. Dingledine. BitTorrent over Tor isn't a good idea, Apr 2010.

[25] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[26] E. Dreyfuss. Chelsea Manning walks back into a world she helped transform, 2017.

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

[28] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In *30th USENIX Security Symposium (USENIX Security 21)*, Vancouver, B.C., Aug. 2021. USENIX Association.

[29] N. S. Evans, R. Dingledine, and C. Grothoff. A practical congestion attack on Tor using long paths. In *USENIX Security Symposium*, pages 33–50, 2009.

[30] C. Feldman. 60 Minutes' most famous whistleblower. *CBS News*, 2016 (Accessed 2020-10-27).

[31] L. Franceschi-Bicchierai. Snowden's favorite chat app is coming to your computer. *Vice*, 2015 (Accessed: 2020-10-27).

[32] A. Gates and K. Q. Seelye. Linda Tripp, key figure in Clinton impeachment, dies. *The New York Times*, 2020 (Accessed 2020-10-27).

[33] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 640–658, Berlin, Heidelberg, 2014. Springer.

[34] T. Gillespie. *Custodians of the Internet: Platforms, content moderation, and the hidden decisions that shape social media*. Yale University Press, 2018.

[35] HashiCorp, Inc. Terraform. https://www.terraform.io/, 2021 (Accessed: 2021-03-01).

[36] R. D. Henderson. Operation Vula against apartheid. *International Journal of Intelligence and Counter Intelligence*, 10(4):418–455, 1997.

[37] N. Hopper, E. Y. Vasserman, and E. Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2):1–28, 2010.

[38] D. Hopwood. Jubjub supporting evidence. https://github.com/daira/jubjub, 2017 (accessed 2020-04-16).

[39] B. Inzaurralde. The Cybersecurity 202: Leak charges against Treasury official show encrypted apps only as secure as you make them. *The Washington Post*, 2018.

[40] L. Kazan-Allen. In memory of Henri Pezerat. http://ibasecretariat.org/mem_henri_pezerat.php, 2009 (Accessed: 2020-10-27).

[41] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373. IEEE, 1997.

[42] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 287–302, 2015.

[43] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 406–422. ACM, 2017.

[44] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134, 2016.

[45] A. Kwon, D. Lu, and S. Devadas. XRD: Scalable messaging system with cryptographic privacy. *arXiv preprint arXiv:1901.04368*, 2019.

[46] D. Lazar, Y. Gilad, and N. Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 711–725, 2018.

[47] D. Lazar, Y. Gilad, and N. Zeldovich. Yodel: strong metadata security for voice calls. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 211–224, 2019.

[48] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. Herd: A scalable, traffic analysis resistant anonymity network for VoIP systems. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 639–652, 2015.

[49] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis. Towards efficient traffic-analysis resistant anonymity networks. *ACM SIGCOMM Computer Communication Review*, 43(4):303–314, 2013.

[50] J. Leopold, A. Cormier, J. Templon, T. Warren, J. Singer-Vine, S. Pham, R. Holmes, A. Ghorayshi, M. Sallah, T. Kozyreva, and E. Loop. The FinCEN Files. *BuzzFeed News*, 2020 (Accessed: 2020-10-27).

[51] S. Li, H. Guo, and N. Hopper. Measuring information leakage in website fingerprinting attacks and defenses. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1977–1992, 2018.

[52] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller. HoneyBadgerMPC and AsynchroMix: Practical asynchronous MPC and its application to anonymous communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 887–903, 2019.

[53] E. May. Streamlabs Q1 2019 live streaming industry report. https://blog.streamlabs.com/youtube-contends-with-twitch-as-streamers-establish-their-audiences-6a53c7b28147, 2019. Accessed: 2020-04-10.

[54] P. Mittal and N. Borisov. ShadowWalker: Peer-to-peer anonymous communication using redundant structured topologies. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 161–172, 2009.

[55] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 215–226, 2011.

[56] J. O'Connor. "I'm the guy they called Deep Throat". *Vanity Fair*, 2006 (Accessed: 2020-10-27).

[57] J. O'Connor, S. Neves, J.-P. Aumasson, and Z. Wilcox-O'Hearn. BLAKE3: One function, fast everywhere, 2020 (accessed: 2020-04-16).

[58] L. Overlier and P. Syverson. Locating hidden servers. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 15–114. IEEE, 2006.

[59] Paradise Papers reporting team. Paradise Papers: Tax haven secrets of ultra-rich exposed. *BBC News*, 2017 (Accessed: 2020-10-27).

[60] D. Phillips. Reality Winner, former NSA translator, gets more than 5 years in leak of Russian hacking report. *The New York Times*, 8, 2019.

[61] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The Loopix anonymity system. In *26th USENIX Security Symposium USENIX Security 17)*, pages 1199–1216, 2017.

[62] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *ACM transactions on information and system security (TISSEC)*, 1(1):66–92, 1998.

[63] E. Rescorla and T. Dierks. The Transport Layer Security (TLS) protocol version 1.3. RFC 1654, RFC Editor, July 1995.

[64] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[65] S. T. Roberts. *Behind the screen: The hidden digital labor of commercial content moderation*. PhD thesis, University of Illinois at Urbana-Champaign, 2014.

[66] C. Savage. Chelsea Manning to be released early as Obama commutes sentence. *The New York Times*, 17, 2017.

[67] M. S. Schmidt and L. Steven. Panama law firm's leaked files detail offshore accounts tied to world leaders. *The New York Times*, 3, 2016.

[68] S. Shane. WikiLeaks leaves names of diplomatic sources in cables. *The New York Times*, 29:2011, 2011.

[69] V. Shoup. On fast and provably secure message authentication based on universal hashing. In *Annual International Cryptology Conference*, pages 313–328. Springer, 1996.

[70] P. Sirinam, M. Imani, M. Juarez, and M. Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1928–1943, 2018.

[71] D. Smith. Trump condemned for tweets pointing to name of Ukraine whistleblower. *The Guardian*, 2019 (Accessed 2020-10-27).

[72] The BBC. Putin critic Navalny jailed in Russia despite protests, 2021 (accessed 2021-02-22).

[73] The Freenet Project. Freenet, 2020.

[74] The Invisible Internet Project. I2P anonymous network, 2020.

[75] The Tor Project. Tor metrics, 2019.

[76] The Wall Street Journal. Got a tip? https://www.wsj.com/tips, 2020 (Accessed: 2020-10-28).

[77] Y. Tsiounis and M. Yung. On the security of ElGamal based encryption. In *International Workshop on Public Key Cryptography*, pages 117–134. Springer, 1998.

[78] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 423–440, 2017.

[79] US Holocaust Memorial Museum. Röhm purge. *Holocaust Encyclopedia*, 2020 (Accessed: 2020-10-27).

[80] US Occupational Safety and Health Administration. The whistleblower protection program. https://www.whistleblowers.gov/, 2020 (Accessed: 2020-10-27).

[81] US Securities and Exchange Commission. Office of the whistleblower. https://www.sec.gov/whistleblower, 2020 (Accessed: 2020-10-27).

[82] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152. ACM, 2015.

[83] Von Spiegel Staff. Inside the NSA's war on internet security. *Der Spiegel*, 2014 (Accessed: 2020-10-27).

[84] L. Wang, K. Ohta, and N. Kunihiro. New key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 237–253. Springer, 2008.

[85] M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.

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

[87] A. Yosilewitz. State of the stream Q2 2019: Tfue rises to the top, non-gaming content grows while esports titles dip, Facebook enters the mix, and we answer what is an influencer? https://blog.streamelements.com/state-of-the-stream-q2-2019-facebook-gaming-growth-gta-v-surges-and-twitch-influencers-get-more-529ee67f1b7e, 2019. Accessed: 2020-04-10.

[88] K. Zetter. Jolt in WikiLeaks case: Feds found Manning-Assange chat logs on laptop. *Wired*, 19 December 2011. Accessed: 2020-05-04.

A The Audit Attack

While many broadcast systems claim privacy with a malicious server, they trade robustness to do so. When a message is *expected*, a server can act as if a user was malicious to prevent aggregation of their request, learning whether that user was responsible for the expected message. If a system aborts in such circumstances, it no longer has the claimed disruption-resistance property. Some systems such as Atom [43] and Blinder [1] solve this by using verifiable secret-sharing in an honest-majority setting; however, this can be costly in practice; others do not prevent this attack.

Express. Express is designed for private readers, but it can be trivially adapted for broadcast (see Sections 5 and 6). However, a malicious server can then exploit the verification procedure [28, Section 4.1] to exclude a user, changing their request to an invalid distributed point function. This excludes the message from the final aggregation, de-anonymizing a broadcaster with probability at least $\frac{1}{(1-\epsilon)N}$ per round (where ϵ is the fraction of corrupted clients). Over even a few rounds, this can lead to a successful de-anonymization of a publisher *without detection* (honest servers cannot tell if a server is cheating and therefore cannot abort protocol).

Riposte. The threat model of Riposte does *not* consider attacks in which servers deny a write request. As a result, a malicious server can eliminate clients undetectably by simply computing a bad input to the audit protocol which causes the request to be discarded by both servers. While this attack can be mitigated by using a multiple servers and assuming an honest majority (as in Blinder [1]), this weakens the threat model and reduces performance.

Application of BlameGame. The BlameGame protocol applies immediately to both Riposte and Express to address this "audit attack" by allowing (honest) servers to assign blame to either a client or server if an audit fails. The only cost (as in Spectrum) is a slight increase in communication overhead which, importantly, is independent of the encoded message in the request.

B Extensions

In this section we describe two practical extensions to Spectrum.

Subscriber Anonymity

The scheme presented in Algorithm 1 provides publisher anonymity; we now consider the problem of *subscriber* anonymity. With multiple channels, clients could download *all* broadcasts to hide which one they want to read. However, with L channels, this requires $L \cdot |m|$ communication. To do better, we use *private information retrieval* (PIR), a well-studied cryptographic primitive designed for this purpose.

Private Information Retrieval. PIR was introduced as a technique to efficiently query a database without leaking information about the retrieved item [16, 41]. Modern PIR schemes trade off communication and server computation. With a state-of-the-art multi-server PIR scheme [6, 20], a client could retrieve one of L $|m|$-bit channels with $O(\log L)$ bandwidth overhead. Despite this low bandwidth, the *computational* overhead is linear in the size of the database (i.e., $O(L \cdot |m|)$). In recent online/offline PIR schemes [20], clients and servers perform a one-time pre-computation, reducing the computational overhead of subsequent retrieval queries. When subscribers download several chunks from the same channel, Spectrum is well suited for such a scheme. Any PIR scheme can be applied "out-of-the-box" to ensure anonymity for subscribers when downloading from the published broadcasts. We also describe a purely practical PIR optimization for a deployment of Spectrum with a few channels (e.g., 10 channels).

Trick: PIR amortization. In simple PIR schemes, a client sends one of 2^L possible queries to each server. When the number of clients N exceeds 2^L, servers can pre-compute responses to *all possible* client queries. Then, the (amortized) server-side computation per client is $O(1)$. For example, with 1,000,000 users and 10 channels, the server-side work is reduced by $\approx 1000\times$.

Content Moderation

In many cases, anonymity is in direct conflict with content moderation and curation. Anonymous platforms are especially susceptible to issues with Child Sexual Abuse Material (CSAM), terrorism, and hate speech [34, 65].

Spectrum allows content moderation without compromising anonymity. Because Spectrum is *permissioned*, the servers can ban a publisher by changing the key of a channel with such content. Users can use the moderation mechanisms deployed on Twitter, Reddit, and similar broadcast platforms (e.g., voting or machine-learning [34]) to remove this material.

C Spectrum Analysis

In this section we formalize the high-level security goals of Section 3.3 with L channels and message space $\{0,1\}^\ell$. Our protocol must be *correct*, *complete*, *sound*, and *private*. Let $\mathbb{G} = (g,p)$ be a group of order p with generator g in which the Discrete Logarithm problem is assumed to be computationally hard [77].

Correctness. Spectrum is *correct* if messages recovered from Cover are all empty (zero):

$$\Pr\left[\begin{array}{l}\textbf{reqs} \leftarrow \mathsf{Cover}();\\ \textbf{msgs} \leftarrow \{\mathsf{Accept}(\tau) \mid \tau \in \textbf{reqs}\}\\ \text{s.t. } \mathsf{Recover}(\textbf{msgs}) = 0^L\end{array}\right] = 1.$$

Additionally, the protocol must recover the input of Broadcast in the appropriate slot (empty elsewhere): for all messages $m \in \{0,1\}^\ell$, channel $j \in [L]$, and key α,

$$\Pr\left[\begin{array}{l}\textbf{reqs} \leftarrow \mathsf{Broadcast}(m,j,\alpha);\\ \textbf{msgs} \leftarrow \{\mathsf{Accept}(\tau) \mid \tau \in \textbf{reqs}\};\\ \textbf{res} \leftarrow \mathsf{Recover}(\textbf{msgs})\\ \text{s.t. } \textbf{res}[j] = m \wedge \textbf{res}[j'] = 0^L \text{ for } j' \neq j\end{array}\right] = 1.$$

Together, these properties are sufficient for protocol correctness: the write tokens from Cover are an identity element for Recover, and the write tokens from Broadcast yield the message m at the correct index with the identity elsewhere.

Completeness of GenAudit and CheckAudit. Spectrum is *complete* if the output of Cover passes the audit: for all lists of channel broadcast keys $\Gamma = (g_1^\alpha, \ldots, g^{\alpha_L})$ with $\alpha_j \in \mathbb{F}_p$,

$$\Pr\left[\begin{array}{l}\textbf{reqs} \leftarrow \mathsf{Cover}();\\ \textbf{audits} \leftarrow \{\mathsf{GenAudit}(\Gamma,\tau) \mid \tau \in \textbf{reqs}\}\\ \text{s.t. } \mathsf{CheckAudit}(\textbf{audits}) = \text{yes}\end{array}\right] = 1.$$

Additionally, the output of Broadcast must pass the audit: for all lists of keys $\Gamma = (g_1^\alpha, \ldots, g^{\alpha_L})$, all channel indices $j \in [L]$, and all messages $m \in \{0,1\}^\ell$,

$$\Pr\left[\begin{array}{l}\textbf{reqs} \leftarrow \mathsf{Broadcast}(m,j,\alpha_j);\\ \textbf{audits} \leftarrow \{\mathsf{GenAudit}(\Gamma,\tau) \mid \tau \in \textbf{reqs}\}\\ \text{s.t. } \mathsf{CheckAudit}(\textbf{audits}) = \text{yes}\end{array}\right] = 1.$$

Soundness of GenAudit and CheckAudit. Spectrum is *sound* if no adversary can generate write requests that pass the audit *and* perform any write operation without the authentication key α_j for channel α_j, except with negligible probability.

Since the adversary "guesses" write requests at each round, we first define an oracle which reveals whether a set of write tokens passes the audit with the given broadcast keys:

$$F(\Gamma, \textbf{reqs}) = \mathsf{CheckAudit}(\{\mathsf{GenAudit}(\Gamma,\tau) \mid \tau \in \textbf{reqs}\}).$$

For all security parameters λ, all subsets of indices $I \subseteq [L]$, and for all probabilistic polynomial time (PPT) adversaries $\mathcal{A}$ with oracle access to $F(\Gamma,\cdot)$:

$$\Pr\left[\begin{array}{l}\boldsymbol{\alpha} = (\alpha_1,\ldots,\alpha_L) \xleftarrow{R} \mathbb{F}^L;\\ \Gamma \leftarrow (g^{\alpha_1},\ldots,g^{\alpha_L});\\ \textbf{reqs} \leftarrow \mathcal{A}^{F(\Gamma,\cdot)}(1^\lambda, \boldsymbol{\alpha}[I]);\\ \textbf{toks} \leftarrow \{\mathsf{GenAudit}(\Gamma,\tau) \mid \tau \in \textbf{reqs}\};\\ \textbf{msgs} \leftarrow \{\mathsf{Accept}(\tau) \mid \tau \in \textbf{reqs}\}\\ \text{s.t. } \mathsf{Recover}(\textbf{msgs})[i] \neq \textbf{0} \text{ for } i \notin I\\ \wedge\ \mathsf{CheckAudit}(\textbf{toks}) = \text{yes}\end{array}\right] \leq \mathsf{negl}(\lambda)$$

where negl is a negligible function. That is, generating a valid write request that passes the audit, without knowledge of the broadcast key associated with the channel index, is only possible with negligible probability.

Privacy. Spectrum is *private* if no adversary controlling all-but-one of the servers can learn which honest clients are publishers: for all fixed $\Gamma = (g^{\alpha_1}, \ldots g^{\alpha_L})$ and for all (strict) subsets of indices $I \subset [L]$, the following distributions are computationally indistinguishable:

$$\left\{ \ \mathbf{reqs}[i], \mathbf{audit} \ \right\}_{i \in I} \approx_c \left\{ \ \mathbf{reqs}'[i], \mathbf{audit}' \ \right\}_{i \in I}$$

where the variables are sampled as

$$\begin{aligned}
&\mathbf{reqs} \leftarrow \mathsf{Broadcast}(m, j, \alpha_j) \\
&\mathbf{audit} \leftarrow \{\mathsf{GenAudit}(\Gamma, \tau) \mid \tau \in \mathbf{reqs}\} \\
&\mathbf{reqs}' \leftarrow \mathsf{Cover}() \\
&\mathbf{audit}' \leftarrow \{\mathsf{GenAudit}(\Gamma, \tau') \mid \tau' \in \mathbf{reqs}'\}.
\end{aligned}$$

That is, no subset of malicious servers can distinguish between a cover request and a broadcast request.

Additionally, because all requests are combined at the end of the round, we also require that revealing the combination of all requests preserves privacy for users. We formalize this property as follows. For all PPT adversaries $\mathcal{A}$, for all messages m and indices j, there exists a negligible function negl such that

$$\Pr\left[\begin{array}{l}
b \xleftarrow{R} \{0,1\}; \\
\mathbf{reqs_0} \leftarrow \mathsf{Broadcast}(m, j, \alpha_j); \\
\mathbf{reqs_1} \leftarrow \mathsf{Cover}(); \\
\mathbf{res} \leftarrow \mathsf{Recover}\Big(\big\{\mathsf{Accept}(\mathbf{reqs_b}), \mathsf{Accept}(\mathbf{reqs_{\bar{b}}})\big\}\Big); \\
b' \leftarrow \mathcal{A}(\mathbf{res}) : b = b'
\end{array}\right]$$
$$\leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

That is, combine does not reveal the ordering of the requests.

Theorem 1 (Security)**.** *The construction in Definition 1 satisfies the* correctness*,* completeness*,* soundness*, and* privacy *properties for Algorithm 1 defined in Appendix C.*

Proof. We prove the theorem for the two-server case. The n-server case follows a similar argument.

Correctness. We have:

$$\begin{aligned}
&\mathsf{Recover}(\{\mathsf{Accept}(\tau) \mid \tau \leftarrow \mathsf{Cover}()\}) \\
&= \sum_{(k \| t) \in \mathsf{Cover}()} \mathsf{DPF.Eval}(k) && (\textit{def.}\ \mathsf{Accept}, \mathsf{Recover}) \\
&= \sum_{k \in \mathsf{DPF.Gen}(0^\ell, 0)} \mathsf{DPF.Eval}(k) && (\textit{def.}\ \mathsf{Cover}) \\
&= (0^\ell)^L. && (\textit{correctness of DPF})
\end{aligned}$$

Further, for any message m, channel index j, and broadcast key α, we have:

$$\begin{aligned}
&\mathsf{Recover}(\{\mathsf{Accept}(\tau) \mid \tau \leftarrow \mathsf{Broadcast}(m, j, \alpha)\}) \\
&= \sum_{(k \| t) \in \mathsf{Broadcast}(m, j, \alpha)} \mathsf{DPF.Eval}(k) && (\textit{def.}\ \mathsf{Accept}, \mathsf{Recover}) \\
&= \sum_{k \in \mathsf{DPF.Gen}(m, j)} \mathsf{DPF.Eval}(k) && (\textit{def.}\ \mathsf{Cover}) \\
&= m \cdot \boldsymbol{e}_j. && (\textit{correctness of DPF})
\end{aligned}$$

The summation operation in the definition of DPF is commutative and associative, with 0^ℓ as the identity element. Consequently, Spectrum recovers the correct broadcast messages.

Completeness. Let $\Gamma = (g^{\alpha_1}, \ldots, g^{\alpha_L})$ be any vector of public authentication keys. First, consider the completeness of Cover. Let $(\tau_A, \tau_B) \leftarrow \mathsf{Cover}()$ where both $\tau_\circ = (k_\circ \| t_\circ)$. We have DPF keys $(k_A, k_B) \leftarrow \mathsf{DPF.Gen}(0^\ell, 0)$ and tags $t_A = t_B \xleftarrow{R} \mathbb{F}$. Let $\beta_\circ \leftarrow \mathsf{GenAudit}(\Gamma, \tau_\circ)$. Then,

$$\begin{aligned}
\beta_A &= g^{\sum_{i=1}^L \alpha_i \cdot \mathsf{SmallEval}(k_A) - t_A} && (\mathsf{GenAudit}(\Gamma, \alpha_A)) \\
\beta_B &= g^{\sum_{i=1}^L \alpha_i \cdot \mathsf{SmallEval}(k_B) - t_B}. && (\mathsf{GenAudit}(\Gamma, \alpha_B)) \\
\implies \mathsf{CheckAudit}(\beta_A, \beta_B) &= \left(\beta_A \stackrel{?}{=} \beta_B\right) = \mathsf{yes}
\end{aligned}$$

since $k_A = k_B$ and $t_A = t_B$ (def. $\mathsf{GenAudit}$), and the protocol is complete for Cover.
Now, consider $\mathsf{Broadcast}$. For all m, j, α_j, we have $(\tau_A, \tau_B) \leftarrow \mathsf{Broadcast}(m, j, \alpha_j)$ where $\tau_A = (k_A \| t_A)$ and $\tau_B = (k_B \| t_B)$. Then:

$$\alpha_j \cdot (\mathsf{SmallEval}(k_A)[j] - \mathsf{SmallEval}(k_B)[j]) = t_A - t_B$$

(def. $\mathsf{Broadcast}$)

$$\alpha_j \cdot \mathsf{SmallEval}(k_A)[j] - t_A = \alpha_j \cdot \mathsf{SmallEval}(k_A)[j] - t_A$$

$$\sum_{i=1}^L \alpha_i \cdot \mathsf{SmallEval}(k_A) - t_A = \sum_{i=1}^L \alpha_i \cdot \mathsf{SmallEval}(k_B) - t_B$$

$$g^{\sum_{i=1}^L \alpha_i \cdot \mathsf{SmallEval}(k_A) - t_A} = g^{\sum_{i=1}^L \alpha_i \cdot \mathsf{SmallEval}(k_B) - t_B}$$

$$\mathsf{GenAudit}(\Gamma, \tau_A) = \mathsf{GenAudit}(\Gamma, \tau_B)$$

(def. $\mathsf{GenAudit}$)

since $\mathsf{SmallEval}(\cdot)[j'] = 0$ for $j' \neq j$. Therefore, $\mathsf{CheckAudit}$ outputs yes as required.

Soundness. We show this by a reduction to the security of the Carter-Wegman MAC scheme based on universal hashing [85] and the collision-resistance of the hash function used in the audit process. Let $\mathcal{A}$ be a PPT adversary $\mathcal{A}$ with oracle access to $F(\Gamma, \cdot)$. For contradiction, assume that $\mathcal{A}$ generates a valid write request τ_A, τ_B with non-negligible probability.

Consider any DPF keys (k_A, k_B) that write a non-zero value to some channel j. For pair of request tokens $\tau_A = (k_A \| t_A), \tau_B = (k_B \| t_B)$ that pass the audit, by inspection of CheckAudit, it must be that:

$$\beta_A = \beta_B$$
$$\implies g^{\sum_{i=1}^{L} \alpha_i \cdot \mathsf{SmallEval}(k_A) - t_A} = g^{\sum_{i=1}^{L} \alpha_i \cdot \mathsf{SmallEval}(k_B) - t_B}.$$

In light of this, there are two cases to consider. In case 1, $\mathsf{SmallEval}(k_A)[j] = \mathsf{SmallEval}(k_B)[j]$. If the DPF keys caused a write, then we must have $m_A \neq m_B$ but $H(m_A) = H(m_B)$. Because we can compute m_A and m_B from the output of $\mathcal{A}$, we can use $\mathcal{A}$ to build an adversary for the security game for the collision-resistance of H with the same probability.
In the other case, we have

$$\mathsf{SmallEval}(k_A)[j] \neq \mathsf{SmallEval}(k_B)[j].$$

Then, we can extract a linear combination of the broadcast keys α^* from the tag shares output by $\mathcal{A}$ as

$$\alpha^* = t \cdot \prod_{i \in I} (\mathsf{SmallEval}(k_A)[i] - \mathsf{SmallEval}(k_B)[i])^{-1},$$

where $j \in I \subseteq [L]$ (i.e., the request may write to more than one channel).

We can think of α^* as being a key for a single channel by merging all channels to which the write occurred into one, which helps simplify the analysis without changing the argument (the sum of random values in a field is random). We can then use $\mathcal{A}$ to build an adversary $\mathcal{B}$ to forge Carter-Wegman MAC tags with the same probability of success: this adversary queries the MAC oracle on message $m = 0$ to obtain the nonce γ and then queries $\mathcal{A}$ to obtain α^*. With (α^*, γ), the adversary forges a valid tag for an arbitrary message $m' \neq m$ and

$$\mathsf{negl}(\lambda) \leq \Pr[\mathcal{A} \text{ succeeds}] = \Pr[\mathcal{B} \text{ succeeds}] = \frac{1}{|\mathbb{F}|}.$$

which is a contradiction.

Privacy. To argue privacy, we must show two things. First, we must argue that the request and audit shares received by a server reveal no information about the message. Second, we must show that the output of the protocol hides the order in which the requests were received and processed.

We first argue that an individual request share in conjunction with the audit shares reveal no information about whether the request is a cover or broadcast message.

Fix some authentication key vector Γ and channel index j. Suppose towards contradiction that some PPT adversary $\mathcal{A}$ could distinguish between the distributions defined for the Spectrum privacy property: namely a view of one request share and both audit tokens.

The variables $\boldsymbol{m}$ and Γ are distributed identically in case of a cover request and a broadcast request. In the two-server setting, the server sees only one element of **reqs**; call this τ (and let $\tau' = (k' \| t')$ indicate the other element). Parsing $\tau = (k \| t)$, we note that k is either one of the DPF keys from $\mathsf{DPF.Gen}(m, j)$ or $\mathsf{DPF.Gen}(0,0)$, and t is distributed uniformly at random without the other half of the proof share. The variable **audit** comprises one element which can be computed from τ and is identically distributed in both cases (if the client computed the request shares correctly). Hence, it must be the case that the adversary was able to distinguish between the distribution of k for $\mathsf{Cover}()$ and $\mathsf{Broadcast}(\boldsymbol{m}[j], j, \Gamma[j])$. However, this would contradict the computational indistinguishability of the DPF simulator. Therefore, we conclude that the request shares on their own reveal no distinguishing information about the underlying message.

Second, we must show that the output of the protocol does not reveal the order in which the requests were accepted by the servers. Specifically, the recovered messages output as a result of applying Recover, are distributed identically regardless of the ordering of inputs to Recover, which follows directly from the associativity of Recover.

With these two requirements covered, we conclude that the protocol is private.

□

D BlameGame Analysis

The BlameGame protocol must be *complete*, *sound*, and *private*.

Completeness. BlameGame is *complete* if for all **reqs** $= (\tau_A, \tau_B)$ generated according to Broadcast or Cover with encryptions $\sigma_A = [\![\tau_A]\!]_{\mathsf{pk}_A}$ and $\sigma_B = [\![\tau_B]\!]_{\mathsf{pk}_B}$,

$$\Pr\left[\begin{array}{l} \beta_A \leftarrow \mathsf{Resolve}(\mathsf{sk}_A, \Gamma, \sigma_A); \\ \beta_B \leftarrow \mathsf{Resolve}(\mathsf{sk}_B, \Gamma, \sigma_B); \\ \mathsf{CheckAudit}(\beta_A, \beta_B) = \mathsf{yes} \end{array}\right] = 1.$$

Soundness. BlameGame is *sound* if no adversary (emulating a malicious client or server) can deflect blame for invalid request tokens. For all **reqs** $= (\tau_A, \tau_B)$ with encryptions $\sigma_A = [\![\tau_A]\!]_{\mathsf{pk}_A}$ and $\sigma_B = [\![\tau_B]\!]_{\mathsf{pk}_B}$ such that $\mathsf{CheckAudit}(\mathsf{GenAudit}(\mathbf{reqs})) = \mathsf{yes}$,

$$\Pr\left[\begin{array}{l} \phi_\circ^* \leftarrow \mathcal{A}(\mathsf{sk}_\circ, \sigma_\circ^*); \\ \mathsf{Blame}(\Gamma, \tau_\circ, \sigma_\circ, \phi_\circ^*) \neq \mathsf{server} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

That is, a malicious server cannot generate acquittal token $\phi_\circ^*$ for any encryption $\sigma_\circ^*$ different from $[\![\tau_\circ]\!]_{\mathsf{pk}_\circ}$ that deflects blame from itself.

Likewise, a malicious client cannot generate a malicious dispute token $\sigma_\circ^*$ that does not pass the audit and deflects blame from itself. For all **reqs** $= (\tau_A, \tau_B)$ with

encryptions $\sigma_A = [\![\tau_A]\!]_{\mathsf{pk}_A}$ and $\sigma_B = [\![\tau_B]\!]_{\mathsf{pk}_B}$ such that $\mathsf{CheckAudit}(\mathsf{GenAudit}(\mathbf{reqs})) = \mathsf{no}$,

$$\Pr\left[\begin{array}{l}\phi_\circ \leftarrow \mathsf{Absolve}(\mathsf{sk}_\circ, \sigma_\circ);\\ \mathsf{Blame}(\Gamma, \tau_\circ, \sigma^*_\circ, \phi_\circ) \neq \mathsf{client}\end{array}\right] \leq \mathsf{negl}(\lambda).$$

That is, for all encrypted request tokens **reqs** which do not pass the audit, the client cannot "frame" the server as being culpable for the failure.

Privacy. The privacy requirement of BlameGame is similar to that of Spectrum. Specifically, the backup request and audit generated using Resolve must not reveal any information about the nature of the request. Formally, for public and private keys

$$\mathbf{pk} \leftarrow \{\mathsf{pk}_i \mid i \in [n]\}$$
$$\mathbf{sk} \leftarrow \{\mathsf{sk}_i \mid i \in [n]\}$$

corresponding to a verifiable encryption scheme, we have that

$$\left\{\ \mathbf{pk}, \mathbf{sk}[i], \mathbf{enc}[i], \mathbf{res}\ \right\}_{i \in I}$$
$$\approx_c \left\{\ \mathbf{pk}, \mathbf{sk}[i], \mathbf{enc}'[i], \mathbf{res}'\ \right\}_{i \in I}$$

where the variables are sampled as

$$\mathbf{enc} \leftarrow \left\{[\![\tau_i]\!]_{\mathsf{pk}_i} \mid \tau_i \in \mathsf{Broadcast}(m, j, \alpha_j)\right\}$$
$$\mathbf{res} \leftarrow \{\mathsf{Resolve}(\mathsf{sk}_i, \Gamma, \mathbf{enc}[i]) \mid \mathbf{enc}[i]\}$$

for the former distribution, and

$$\mathbf{enc}' \leftarrow \left\{[\![\tau_i]\!]_{\mathsf{pk}_i} \mid \tau_i \in \mathsf{Cover}()\right\}$$
$$\mathbf{res}' \leftarrow \{\mathsf{Resolve}(\mathsf{sk}_i, \Gamma, \mathbf{enc}'[i]) \mid \mathbf{enc}'[i]\}.$$

for the latter.

In words, the decrypted request and backup audit shares must not reveal whether the request contains a cover or broadcast message.

However, we note that BlameGame does not require any privacy properties on Absolve, as it may reveal the request for the purpose of assigning blame.

We now describe the security of BlameGame. Specifically, we must show that BlameGame aborts Spectrum when a server deviates from protocol in an attempt to de-anonymize a publisher.

Theorem 4 (Security)**.** *BlameGame satisfies the* completeness*,* soundness*, and* privacy *properties defined in Appendix D when a server is actively malicious.*

Proof. We must first show that the only way in which a malicious server can deviate from protocol is by causing an audit to fail. We then show that deviating in this way results in the protocol aborting.

Ways in which a malicious server can deviate. The goal of a malicious server is to cause a client's broadcast (or cover) request to not appear in the final output. Observing which broadcasts appeared and which did not, reveals information about whether the dropped request was sent by a publisher. A malicious server can deviate by either not adding a client's request to the local message aggregate, modifying the request, or by causing the audit step to fail by changing the audit token generated by GenAudit.

In the first case, the resulting output will be disrupted (servers will recover a random value) regardless of whether or not the publisher's request was dropped, hence revealing no useful information to the malicious server. The same holds true if the server tampers with the request itself.

On the other hand, if the server causes CheckAudit to fail, then the request is dropped by *both* servers which will result in a missing broadcast message in the case that the malicious server successfully guessed which request belonged to a publisher. This latter scenario is what we must prevent with BlameGame.

Since the BlameGame protocol is not invoked when CheckAudit outputs yes (since at least one of the servers will incorporate the request into the aggregate), in what follows we condition on CheckAudit outputting no for client-issued request tokens τ_A and τ_B in the Spectrum protocol. With this in mind, we argue that the required properties are achieved.

Completeness. If the dispute tokens σ_A and σ_B are well formed (are encryptions of request tokens τ_B and τ_A, respectively) then swapping the dispute tokens and decrypting them will result in ServerA decrypting τ_A and ServerB decrypting τ_B. Assuming the client correctly generated and encrypted the request tokens, then we have that the resulting audit tokens β_A and β_B (resulting from running Resolve) will pass the audit, i.e., $\mathsf{CheckAudit}(\beta_A, \beta_B) = \mathsf{yes}$.

Soundness. If the server deviates from executing Resolve on the received dispute token σ, then the probability that Spectrum is not aborted is negligible in the security parameter. To see why, if the malicious server causes $\mathsf{CheckAudit}(\beta_A, \beta_B) = \mathsf{no}$ for β_A and β_B computed with Resolve by each server, then the honest server requests the malicious server to reveal the decryption of σ. A malicious server that changed β such that it is different from what $\mathsf{GenAudit}(\Gamma, \tau)$ outputs on the decrypted request τ, is unable to provide a valid proof-of-decryption to the honest server. Doing so would contradict the soundness of the verifiable encryption scheme. Therefore, the honest server detects that either (a) the malicious server provided an invalid decryption for σ or (b) the malicious server provided a correct proof-of-decryption but the resulting audit token β is different from the one received from the malicious server. In both cases (a) and (b), the honest server aborts since it is clear that the malicious server deviated from protocol. On the other hand, if neither server deviated from protocol and the second audit fails, both servers will be able to prove correct decryption of the received tokens σ_A and σ_B and locally ensure that the audit indeed fails by inspecting

the request (which implies that the encrypted tokens were generated incorrectly). This places blame on the client since the encrypted request does not pass the audit.

Privacy. For all honest publishers, privacy is guaranteed with probability $\frac{L}{N\cdot(1-\epsilon)}$ where ϵ is the fraction of corrupted clients. If the first audit fails but the second audit (generated from the decrypted requests) passes, then privacy follows from the analysis of Spectrum and privacy of the audit therein. If the second audit fails, then the request is revealed to both servers for inspection (in order to adequately assign blame). However, predicated on the revealed request being generated correctly (since we are interested in when an *honest* publisher gets de-anonymized), the protocol aborts if the second audit fails (an honest publisher would have encrypted the request correctly). In this case, both servers see the request which de-anonymizes the client. Thus, for a fraction of corrupted clients ϵ, the probability that the malicious server chooses the correct request to tamper with before being aborted is $\frac{L}{N\cdot(1-\epsilon)}$.

□

E Verifiable Encryption

Definition 2 (Verifiable Encryption)**.** *A verifiable public-key encryption scheme $\mathcal{E}$ consists of (possibly randomized) algorithms* Gen, Enc, Dec, DecProof, VerProof *where* Gen,Enc,Dec *satisfy IND-CPA security and* DecProof, VerProof *satisfy the following properties:*

Completeness. *For all messages $m \in \mathcal{M}$,*

$$\Pr\left[\begin{array}{l}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda);\\ c \leftarrow \mathsf{Enc}(\mathsf{pk},m);\\ (\pi,m) \leftarrow \mathsf{DecProof}(\mathsf{sk},c);\\ \mathsf{VerProof}(\mathsf{pk},\pi,c,m) = \mathsf{yes}\end{array}\right] = 1.$$

where the probability is over the randomness of Enc.

Soundness. *For all PPT adversaries $\mathcal{A}$ and for all messages $m \in \mathcal{M}$,*

$$\Pr\left[\begin{array}{l}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda);\\ c \leftarrow \mathsf{Enc}(\mathsf{pk},m);\\ (\pi,m') \leftarrow \mathcal{A}(1^\lambda,\mathsf{pk},\mathsf{sk},c);\\ \mathsf{VerProof}(\mathsf{pk},\pi,c,m') = \mathsf{yes}\end{array}\right] \leq \mathsf{negl}(\lambda)$$

for negligible function negl(λ)*, where the probability is over the randomness of* Enc *and $\mathcal{A}$.*

We note that most public key encryption schemes (e.g., ElGamal [27] and RSA [64]) satisfy Definition 2 out-of-the-box and can be used to instantiate BlameGame.

F Distributed Point Functions

Algorithm 3 describes a construction for an n-DPF. The construction is a simplified version of the DPF used in Riposte [18] (see details and security analysis therein). We use this construction to instantiate Algorithm 1 with more than two servers. For the two-server case, there exist more efficient constructions [9,10] which can be used, although Algorithm 3 can also be instantiated with two-servers using an AES-based PRG for better concrete efficiency.

The construction uses a *seed-homomorphic* PRG $G : \mathbb{S} \to \mathbb{F}$ where $\mathbb{S}$ is the seed space and $\mathbb{F}$ is the output space. This allows us to choose a set of n seeds that "cancel out" when evaluated and combined. For $n = 2$, *any* PRG suffices, as $G(s) \oplus G(s) = 0$ when $\oplus$ is bitwise exclusive-or. This permits fast instantiations in practice using AES-based PRGs [18].

In Algorithm 3, we also describe the SmallEval algorithm used in Spectrum which outputs the point encoding of the DPF and a hash of the encoded message m.

Algorithm 3: n-key DPF Construction

Params : L and PRG $G : \mathbb{S} \to \mathbb{F}$.

def Gen(m, j):
- **if** $j \in [L]$ **then**
  - $s' \xleftarrow{R} \mathbb{S}$ // masking seed
  - $\boldsymbol{b} \leftarrow e_j \in \mathbb{F}^L$ // $(\ldots, 0, 1, 0, \ldots)$
  - $\boldsymbol{s} \leftarrow s' \cdot e_j \in \mathbb{F}^L$ // $(\ldots, 0, s', 0, \ldots)$
  - $\widehat{m} \leftarrow m - G(s')$ // mask the message
- **else**
  - $\boldsymbol{b} \leftarrow (0,\ldots,0) \in \mathbb{F}^L$
  - $\boldsymbol{s} \leftarrow (0,\ldots,0) \in \mathbb{F}^L$
  - $\widehat{m} \xleftarrow{R} \mathbb{F}$
- split $\boldsymbol{s}$ into additive secret shares $\boldsymbol{s}_i$ for $i \in [n]$
- split $\boldsymbol{b}$ into additive secret shares $[\![\boldsymbol{b}]\!]_i$ for $i \in [n]$
- $k_i \leftarrow (\boldsymbol{s}_i, \boldsymbol{b}_i, \widehat{m})$ for $i \in [n]$
- **output** $(k_1,\ldots,k_n)$

def Eval$(k_\circ)$:
- parse $k_\circ$ as $(\boldsymbol{s}, \boldsymbol{b}, \widehat{m})$
- **for** $j \in [L]$ **do**
  - $m_j \leftarrow G(\boldsymbol{s}_j) + \boldsymbol{b}_j \cdot \widehat{m}$
- **output** $\boldsymbol{m} = (m_1,\ldots,m_L)$

Params : CRHF $H : \{0,1\}^\ell \to \mathbb{G}$.

def SmallEval$(k_\circ)$:
- parse $k_\circ = (\boldsymbol{b}, \boldsymbol{s}, \widehat{m})$
- **for** $j \in [L]$ **do**
  - $\rho_j \leftarrow (\boldsymbol{b}_j, \boldsymbol{s}_j, H(\widehat{m}))$ // compressed output
- **output** $(\rho_1,\ldots,\rho_L)$