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

May 3, 2023

Abstract

We present Spectrum, a high-bandwidth, metadata-private file broadcasting system. In Spectrum, a small number of broadcasters share a file with many subscribers via two or more non-colluding broadcast servers. Subscribers generate cover traffic by sending dummy files, hiding which users are broadcasters and which users are only consumers.

Spectrum optimizes for a setting with few broadcasters and many subscribers—as is common to many real-world applications—to drastically improve throughput over prior work. Malicious clients are prevented from disrupting broadcasts using a novel blind access control technique that allows servers to reject malformed requests. Spectrum also prevents deanonymization of broadcasters by malicious servers deviating from protocol. 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's peak throughput is 4–120,000× faster (and commensurately cheaper) in a broadcast setting. Deployed on two commodity servers, Spectrum allows broadcasters to share 1 GB (two full-length 720p documentary movies) in 13h 20m with an anonymity set of 10,000 (for a total cost of about $6.84). These costs scale roughly linearly in the size of the file and total number of users, and Spectrum parallelizes trivially with more hardware.

1 Introduction

An informed public often depends on whistleblowers, who expose misdeeds and corruption. Over the last century, whistleblowers have exposed financial crimes, government corruption [61, 69, 75], risks to public health [43, 52], Russian interference in the 2016 U.S. presidential election [61, 70], presidential misconduct [17, 45, 67, 79], war and human rights crimes [5, 38, 88], and digital mass surveillance by U.S. government agencies [18]. Philosophers debate whistleblowing ethics [3, 35], but agree it often has positive effects.

Motivation for this work. Whistleblowers take on great personal risks in bringing misdeeds to light. The luckiest enjoy legal protections [89] or financial reward [90]. But many face exile [18], incarceration [50, 70, 74], or risk their lives [88]. 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 [81].

To mitigate these risks, many whistleblowers turn to technology to protect themselves [47]. Secure messaging apps Signal [26] and SecureDrop [8] have proven to be an important resource to whistleblowers and journalists [44, 85]. Encryption does its job, even against the NSA [80]—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. For instance, prosecutors used SFTP metadata in the case against Chelsea Manning [96]. Newer technology is still vulnerable: a federal judge found Natalie Edwards guilty on the evidence of metadata from an encrypted messaging app [50]. To protect whistleblowers and protect against powerful adversaries (e.g., corrupted internet service providers), 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 [37] 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 deanonymization attacks identify users with a high success rate based on observed traffic [9, 14, 42, 48, 53, 65, 68]. Moreover, the effectiveness of deanonymization attacks increases with the size of the traf-

*The proceedings version of this paper is titled "High-bandwidth Anonymous Broadcast," because reviewers suggested that "malicious security" was ill-defined (and we agree). Unfortunately, we cannot update the title of the ePrint archive version since it has already been cited multiple times. We note, however, that the contents of this version is nearly identical to the proceedings version (aside from a few corrections and some additional details).

fic pattern. Whistleblowers using Tor to upload large files can be more easily deanonymized for this reason.

Some recent academic research systems [2, 30, 41, 54–56, 58, 87, 91] 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 8).

Contributions. Motivated by the lack of anonymous broadcast systems suitable for high-throughput data dumps, we design and build Spectrum: a system for high-bandwidth metadata-private anonymous broadcasting. Spectrum is the first anonymous broadcast system supporting high-bandwidth, many-user settings. It optimizes for the many-subscriber and few-broadcaster setting, which reflects the real-world usage of broadcast platforms. Per-request, Spectrum's server-side processing costs grow with the number of broadcasters rather than the total number of users, significantly improving performance over prior work when only a subset are broadcasting.

This paper contributes:

1. Design and security analysis of Spectrum, a system for high-bandwidth broadcasting with strong robustness and privacy guarantees against malicious adversaries,
2. A notion of blind access control for anonymous communication, along with a construction and a black-box transformation to efficiently support large (1 GB) messages,
3. Identification of an "audit attack" that allows malicious servers to deanonymize users, and BlameGame, a black-box blame protocol to "upgrade" Spectrum and similar systems to defend against this attack.
4. An open-source implementation of Spectrum, evaluated in comparison to other anonymous communication systems. We show that Spectrum supports high-bandwidth, latency-sensitive applications such as real-time anonymous podcasting, video streaming, and large file leaks.

Limitations. Like other metadata-private systems, Spectrum provides anonymity among honest online users and requires all users to contribute cover messages to a broadcast (to perfectly hide network metadata). Additionally, Spectrum achieves peak performance with exactly two servers (similarly to Riposte and Express [30, 41]). Instantiating with more than two servers requires using a less (concretely) efficient cryptographic primitive: a seed-homomorphic pseudorandom generator [12]. Finally, Spectrum requires a one-time "bootstrapping" process at setup time (similar to other systems [4, 29, 41, 58, 91, 95]); see Section 2.3.

2 Anonymous broadcast

In this section, we describe anonymous broadcast and its challenges, along with our system design and techniques.

Setting and terminology. In anonymous broadcast, one or more users/clients (*broadcasters*) share a *message* (e.g., file) while preventing network observers from learning its source. In Spectrum, passive users generate cover traffic (dummy messages) to increase the size of the *anonymity set* (the set of users who could have plausibly sent the broadcast message). These passive users are *subscribers*, consuming broadcasts. Because the message sources are anonymous, the servers publish distinct messages to different *channels* or slots. Every broadcaster has exactly one channel, which they anonymously publish to in every iteration of the protocol. The servers cannot distinguish between a subscriber sending cover traffic and a broadcaster writing to a channel.

The primary challenge in anonymous broadcasting is preventing *disruption* by malicious clients: in simple broadcasting systems, users can clobber other users' messages via undetectable deviations from the protocol [2, 30, 41]. We first begin by explaining the standard building-block for achieving anonymous broadcasting [2, 30]. In subsequent sections, we build off of this basic scheme to achieve disruption resistance.

2.1 DC-nets

Chaum [23] presents DC-nets, which enable a rudimentary form of anonymous broadcast. DC-nets use secret-sharing to obscure the source of data in the network. Like prior work [2, 30, 41, 95], we instantiate a DC-net with two or more servers and many clients. One client (the broadcaster) wishes to share a file; all other clients (subscribers) provide cover traffic. In a two-server DC-net, the ith client samples a random bit string r_i and sends $r_i \oplus m_i$ to ServerA and r_i to ServerB. Servers can recover m_i by combining their respective shares:

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

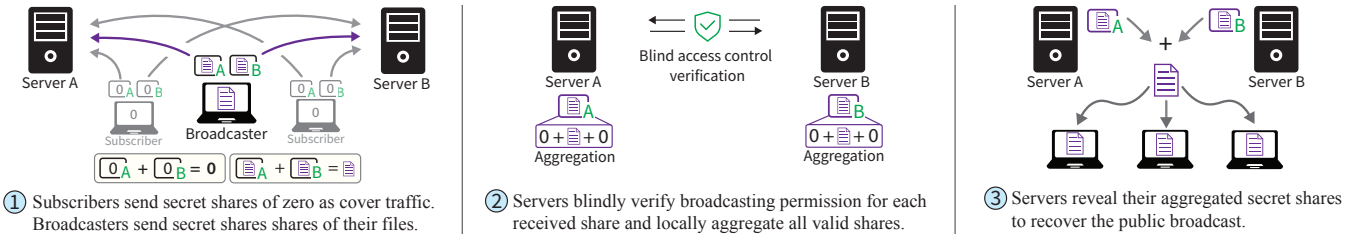
If exactly one of N clients shares a message $m_i = \widehat{m}$ while all other clients share $m_i = 0$, the servers can recover $\widehat{m}$ (without learning which client sent $m_i = \widehat{m}$) by first locally aggregating all received shares as $\mathsf{agg}_A = \bigoplus_i (r_i \oplus m_i)$ and $\mathsf{agg}_B = \bigoplus_i r_i$ and then revealing the aggregation to the other server.

Because all subscribers send shares of zero, combining the local aggregations recovers the broadcaster's message:

$$\widehat{m} = \mathsf{agg}_A \oplus \mathsf{agg}_B.$$

The above scheme protects client anonymity, as each server sees a uniformly random share from each client.

DC-net challenges. While DC-nets allow fast anonymous broadcast, users can undetectably *disrupt* the broadcast by sending non-zero shares. Preventing such disruptions is a major challenge and primary source of latency in prior DC-net-based systems [2, 29, 30, 41, 54, 55, 95] (see related work; Section 8). Also, while DC-nets enable one broadcaster to



(1) Subscribers send secret shares of zero as cover traffic. Broadcasters send secret shares shares of their files.

(2) Servers blindly verify broadcasting permission for each received share and locally aggregate all valid shares.

(3) Servers reveal their aggregated secret shares to recover the public broadcast.

Figure 1: High-level overview of Spectrum when instantiated with two servers (and one broadcaster).

transmit a message, many clients may wish to broadcast. Repeating the protocol in parallel is inefficient, requiring bandwidth linear in the number of broadcasters. Even prior works which overcome the linear (in the number of broadcasters) bandwidth overhead of naïve protocol repetition suffer from linear server-side work per client, regardless of whether or not all clients are broadcasters. In Spectrum, the bandwidth overhead grows *logarithmically* in the number of broadcasters. Additionally, the server-side work only grows linearly in the number of broadcasters, rather than the total number of clients. We compare this work and other DC-net-based anonymous broadcasting systems in Table 1.

2.2 Main ideas in realizing Spectrum

Spectrum builds on top of DC-nets, improving efficiency and preventing disruption by malicious clients.

Practical efficiency. Spectrum capitalizes on the asymmetry of real-world broadcasting: there are typically fewer broadcasters than there are subscribers. While some prior works repeat many executions of the DC-net protocol more efficiently than the naïve scheme, they still reserve space (i.e., channels) for *every* client [2, 30]. As a consequence, the per-request computation on each server is linear in the number of clients, leading to high latency and "wasted" work. Spectrum derives anonymity from *all* clients, but only the total number of *broadcasters* influences the per-request work on each server (rather than the total number of clients in the system).

Preventing disruption. In Spectrum, we prevent broadcast disruption by developing a new idea: *anonymous* access control (Section 3.1), which we realize from the Carter-Wegman MAC [94]. We check access to each "channel" to ensure that only a user with a "broadcast key" can write to that channel.

Preventing "audit" attacks. Anonymous broadcast servers can *covertly* exclude a client in order to deanonymize the corresponding user. While vanilla DC-nets do not have this problem, prior anonymous broadcast systems leave out a client's share if they are found to be ill-formed. This is done to defend against disruption. However, it also makes it possible for a malicious server to exclude a user by framing them as malicious. In the broadcast setting, excluding a user can effectively deanonymize them. Abraham et al. [2] make the same observation and defend against the attack by requiring an honest-majority out of five or more servers. In Dissent [29], deanonymization is prevented with an expensive, after-the-fact blame protocol. Other systems [30, 41] are vulnerable to this attack (see Appendix A for details). Spectrum is the first system to efficiently defend against this attack while still preventing disruption per request (rather than assigning blame after-the-fact). We achieve this by introducing BlameGame (Section 4.3), a lightweight blame protocol which can also be applied to other systems (e.g., Riposte [30] and Express [41]).

2.3 System overview

Spectrum is built using two or more broadcast servers (only one must be honest to guarantee anonymity; see Section 2.4) and many clients consisting of broadcasters and subscribers. One or more broadcaster(s) wish to share a message (as in the DC-net example). The subscribers generate cover traffic to increase the anonymity set. Each broadcaster has a private *channel*—or slot—for their message. Subscribers do not have channels. At the end of each round, Spectrum publishes the contents of each channel, hiding which client wrote to which channel (if any). Spectrum has three phases.

Setup. During setup, all broadcasters register with the servers. All users perform a setup-free anonymous broadcast protocol to establish a channel in Spectrum. Specifically, each broadcaster shares a public authentication key with the servers, which will be used to enforce anonymous access to write to a channel. At the end of the setup phase, the servers publish all parameters, including the number of channels and the maximum size of each broadcast message per round.

Main protocol. The protocol proceeds in one or more rounds (overview in Figure 1; details in Section 4.2). In each round, every client sends request shares to each server. The broadcasters send shares of their messages while the subscribers send empty shares. To enforce access control, the servers perform an efficient audit over the received shares: they obliviously check that each writer to a channel knows the secret channel broadcast key, *or* their message is zero. If the message shares pass the audit, the servers aggregate them as in a DC-net (Section 2.1). Otherwise, the servers perform a blame protocol (see BlameGame, summarized below). Finally, the servers combine aggregated shares to recover the messages.

BlameGame. If any client's request fails the audit, the servers perform BlameGame, a simple blame protocol (detailed in Section 4.3). BlameGame determines whether a client failed the access control check or if a server tampered with the client request in an attempt to frame a client as malicious. If the client is blamed, the servers drop the client's request and proceed with the main Spectrum protocol. Otherwise, if a server is blamed, the honest server(s) abort. This protocol is much faster than fully aggregating a client's request, so a malicious client cannot use this to cause significant delays.

2.4 Threat model and security guarantees

Spectrum is instantiated with two (or more) broadcast servers and many clients (broadcasters and subscribers). Clients send shares of a message to the servers for aggregation.

Threat model

- No client is trusted by any honest server.
- Clients may deviate from the protocol, collude with other clients, or collude with a subset of malicious servers.
- At least one server must be honest to guarantee anonymity for clients (it does not matter which server is honest).
- Any subset of servers may deviate from the protocol and collude with malicious clients or the network adversary.

Assumptions. We make black-box use of public key infrastructure (e.g., TLS [73]) to encrypt data between clients and servers. We make the following cryptographic assumptions: (1) the hardness of the discrete logarithm problem [11], (2) the hardness of the decision Diffie-Hellman problem [10, 40] (when instantiated with more than two servers), and (3) the existence of hash functions and pseudorandom generators. We also assume a setup-free anonymous broadcast system [2, 30, 55, 95] for bootstrapping. As with prior work [2, 29, 30, 41, 55], we assume all communication between parties is observed by the network adversary.

Guarantees. Under the above threat model and assumptions, we obtain the following guarantees.

- *Anonymity.* An adversary controlling the network and a strict subset of servers and clients cannot distinguish between honest clients: broadcasters and subscribers are cryptographically indistinguishable in Spectrum. That is, no adversary observing the network and controlling a subset of servers and clients can distinguish between an honest subscriber and an honest broadcaster.
- *Availability.* If all servers follow the protocol, the system remains available (even if many clients are malicious). If any server halts or deviates from the protocol, then availability is not guaranteed and the protocol may abort.

Non-goals. We do not protect against denial-of-service attacks by a large number of clients (but we note that standard techniques, such as CAPTCHA [92], anonymous one-time-use tokens [33], or proof-of-work [39, 51] apply). Like all anonymous broadcast systems, intersection attacks on participation in the protocol can identify users, so Spectrum requires that users stay online for the duration of the protocol.

3 Spectrum with one channel

In this section, we introduce Spectrum with a single broadcaster (and therefore a single channel), two servers, and many subscribers. Figure 1 depicts an example. This setup mirrors the simplest DC-net protocol of Section 2.1. In Section 4, we extend Spectrum to many broadcasters and many servers.

3.1 Preventing disruption

We denote by $\mathbb{F}$ any finite field of prime order (e.g., integers mod p). We assume that all messages are elements in $\mathbb{F}$. (Section 5.1 shows how to efficiently support large binary messages in $\mathbb{F}_{2^\ell}$.) Each server receives secret-shares of a message m_i, where $m_i = 0 \in \mathbb{F}$ for subscribers and $m_i = \widehat{m} \in \mathbb{F}$ for the broadcaster. To prevent disruption, we enforce the following rule: for each channel, the broadcaster (with knowledge of a pre-established broadcast key) can send a non-zero message; all subscribers (who do not have the broadcast key) can only share a zero message. We give a new technique enabling the servers to verify the rule efficiently *without* learning anything except for the validity of the provided secret-shares.

New tool: anonymous access control. We adapt the Carter-Wegman MAC [21, 94] to provide a secret-shared "access proof" accompanying the message shares. Each client sends a secret-shared proof that it is either: (1) sending a share of a broadcast message with knowledge of the broadcast key; or (2) sending a cover message (i.e., $m_i = 0$) that does not affect the final aggregate computed by the servers.

Carter-Wegman MAC. Let $\mathbb{F}$ be any finite field of sufficiently large size for security. Sample a random authentication key $(\alpha, \gamma) \in \mathbb{F} \times \mathbb{F}$ and define $\mathsf{MAC}_{(\alpha,\gamma)}(m) = \alpha \cdot m + \gamma \in \mathbb{F}$. 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. We demonstrate this with two servers ServerA and ServerB. Let $t = \mathsf{MAC}_{(\alpha,\gamma)}(m)$. If m is additively secret-shared as $m = m_A + m_B \in \mathbb{F}$, and t is secret shared as $t = t_A + t_B \in \mathbb{F}$, the servers (knowing α and γ) can verify that the tag corresponds to the secret-shared message:

- ServerA computes $\beta_A \leftarrow (\alpha \cdot m_A - t_A) \in \mathbb{F}$.
- ServerB computes $\beta_B \leftarrow (\alpha \cdot m_B - t_B) \in \mathbb{F}$.
- Servers swap β_A and β_B and check if $\beta_A + \beta_B = \gamma \in \mathbb{F}$.

The final condition only holds for a valid tag. 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.

	Request Size	Audit Size	Audit Rounds	Server Work	Malicious Security	Disruption Handling	Blame Protocol	Comments
Blinder [2]	$\|m\| \cdot \sqrt{N}$	$\lambda \cdot \|m\|$	$\log N$	$N \cdot \|m\|$	✓	Prevent	N/A	Requires 5+ servers and MPC
Dissent [29]	$\|m\| \cdot L + N$	N/A	N/A	$L \cdot \|m\|$	✓	Detect	Expensive	Blame quadratic in N
PriFi [6]	$\|m\| \cdot L + N$	N/A	N/A	$L \cdot \|m\|$	✓	Detect	Expensive	Similar to Dissent
Riposte [30]	$\|m\| + \sqrt{N}$	$\sqrt{N}$	1	$N \cdot \|m\|$	✗	Prevent	None	Requires a separate audit server
Express [41]	$\|m\| + \log L$	λ	1	$L \cdot \|m\|$	✗	Prevent	None	Exactly 2 servers
Two-Server	$\|m\| + \log L$	λ	1	$L \cdot \|m\|$	✓	Prevent	Lightweight	With tree-based DPF [15]
Multi-Server	$\|m\| + \sqrt{L}$	λ	1	$L \cdot \|m\|$	✓	Prevent	Lightweight	With seed-homomorphic DPF [12, 30]

Table 1: Per-request asymptotic efficiency of Spectrum (highlighted) and prior anonymous broadcasting systems for L broadcasters, N total users, $|m|$-sized messages, and global security parameter λ. $O(\cdot)$ notation suppressed for clarity. Spectrum's advantages include: a request size that is sublinear in L (Section 5.1) and independent of N (Section 3.3), a protocol for lightweight auditing of client requests to prevent disruption (Section 3.1), and a fast blame protocol for security against malicious servers (Section 4.3).

If both the servers and the broadcaster know the key (α, γ), the broadcaster can compute a tag t which the servers can check for correctness as above. However, there are two immediate problems to resolve. First, subscribers cannot generate valid tags on zero messages without knowledge of (α, γ). Second, an honest-but-curious (or compromised) server can share (α, γ) with a malicious client who can then covertly disrupt a broadcast. (A malicious server can always *overtly* disrupt the broadcast by refusing to participate in Spectrum.)

Allowing forgeries on zero messages. To allow subscribers to send the zero message *without* knowing the secret MAC key, we leverage the following insight from the SPDZ [31] multi-party computation protocol. The γ value acts solely as a "nonce" to prevent forgeries on the message $0 \in \mathbb{F}$ [93]. Because of this, we can eliminate γ while still having the desired unforgeability property of the original MAC for all *non-zero* messages. When evaluated over secret shares, $\mathsf{MAC}_\alpha(m) = \alpha \cdot m \in \mathbb{F}$ maintains security for all $m \neq 0$. This satisfies our requirement: Subscribers can send $m = 0$ and a valid tag $t = 0$ *without* knowing α (i.e., subscribers can "forge" a valid tag but only for $m = 0$).

Preventing client-server collusion. To prevent an honest-but-curious server from collaborating with a malicious client to disrupt a broadcast, we must prevent the servers from knowing the broadcast key α while still allowing them to check the MAC tag. To achieve this, we shift the entire verification procedure "to the exponent" of a group $\mathbb{G}$ of prime order p (so that the exponent constitutes a field $\mathbb{F}_p$). For security, we also require that the discrete logarithm problem is computationally intractable in the group $\mathbb{G}$ [86]. Then, instead of α, the servers obtain a public verification key g^α (here g is a generator of $\mathbb{G}$) from each broadcaster. All verification proceeds as before. Each client generates secret-shares (t_A, t_B) of a tag t and shares (m_A, m_B) of the message m, which are distributed to the servers.

- ServerA computes $g^{\beta_A} \leftarrow (g^\alpha)^{m_A} / g^{t_A}$.
- ServerB computes $g^{\beta_B} \leftarrow (g^\alpha)^{m_B} / g^{t_B}$.
- Servers swap g^{β_A} and g^{β_B} and check if $g^{\beta_A} \cdot g^{\beta_B} = g^0 = 1_\mathbb{G}$.

Security. The unforgeability properties are inherited from the Carter-Wegman MAC. Client anonymity (i.e., secrecy of the message m_i) follows from the additive secret-sharing. Client-server collusion is prevented by only the broadcaster knowing the broadcast key α. See Section 6 for full analysis.

3.2 Putting things together

In this section, we combine DC-nets for broadcast with anonymous access control to realize Spectrum with a single channel, generalizing to multiple channels in Section 4.

Setup: broadcast key distribution. The setup in Spectrum involves the broadcaster anonymously "registering" with the servers by giving them the authentication public key g^α. The servers must not learn the identity of the broadcaster when receiving this key, which leads us to a somewhat circular problem: broadcasters need to anonymously broadcast a key in order to broadcast anonymously. We solve this one-time setup problem as follows. All clients use a slower anonymous broadcast system suitable for low-bandwidth content at system setup time [2, 30, 55, 95]. The broadcaster shares an authentication key while subscribers share nothing. Keys are small (e.g., 64 bytes) and therefore practical to share with existing anonymity systems. Moreover, once the keys for the broadcaster are established, they may be used indefinitely. This process is similar to a "bootstrapping" setup found in related work [4, 29, 41, 58, 91, 95]. Spectrum is agnostic to how this setup takes place: one possibility is to use Riposte [27, 30], which shares a similar threat model.

Step 1: Sharing a message. As in the DC-net scheme, the broadcaster generates secret-shares of the broadcast message $\widehat{m}$ in the field $\mathbb{F}$. All other clients (subscribers) generate secret-shares of the message 0. The only difference is that in

Spectrum, the broadcaster knows the broadcast key α while subscribers do not. Let $y = \alpha$, if the client is the broadcaster and $y = 0$ otherwise. Each client proceeds as follows.

1.1: Sample random $m_A, m_B \in \mathbb{F}$ such that $m = m_A + m_B \in \mathbb{F}$.

1.2: Compute $t \leftarrow y \cdot m \in \mathbb{F}$. `// MAC tag (Section 3.1)`

1.3: Sample random $t_A, t_B \in \mathbb{F}$ such that $t = t_A + t_B \in \mathbb{F}$.

1.4: Send (m_A, t_A) to ServerA and (m_B, t_B) to ServerB.

The above amounts to secret-sharing the message and access control MAC tag between both servers.

Step 2: Auditing shares. Servers collectively verify access control using the shares of the message and tag.

2.1: ServerA computes $g^{\beta_A} \leftarrow (g^{\alpha})^{m_A} / g^{t_A}$.

2.2: ServerB computes $g^{\beta_B} \leftarrow (g^{\alpha})^{m_B} / g^{t_B}$.

2.3: The servers swap audit tokens g^{β_A} and g^{β_B} and verify that $g^{\beta_A} \cdot g^{\beta_B} = g^0 = 1_{\mathbb{G}}$.

The above follows the access control verification (Section 3.1). All shares that fail the audit are discarded by both servers. In Section 4, we discuss how we prevent "audit attacks" in which a server tampers with a client request so the check fails.

Step 3: Recovering the broadcast. Servers collectively recover the broadcast message by aggregating all received shares that pass the audit.

3.1: ServerA computes $\mathsf{agg}_A \leftarrow \sum_i (m_A[i]) \in \mathbb{F}$.

3.2: ServerB computes $\mathsf{agg}_B \leftarrow \sum_i (m_B[i]) \in \mathbb{F}$.

3.3: Servers swap agg_A and agg_B.

3.4: Servers compute $\widehat{m} \leftarrow \mathsf{agg}_A + \mathsf{agg}_B \in \mathbb{F}$.

This recovers the broadcast message as in the vanilla DC-net scheme. The recovered message is then made public to all clients (e.g., via a public bulletin board [7, 25]).

3.3 Towards the full protocol

The single-channel scheme presented in Section 3.2 achieves anonymous broadcast while also preventing broadcast disruption by malicious clients. Two problems remain however. First, while the single-channel scheme is fast and robust against malicious clients, it does not efficiently extend to multiple broadcasters. Second, a malicious server can tamper with the audit to make it fail for one or more clients—and learn whether one of them was a broadcaster (see Appendix A).

Supporting multiple channels. To support multiple channels, we use distributed point functions (DPFs) [15, 16, 46] to "compress" secret-shares across multiple instances of the DC-net scheme. DPFs avoid the linear bandwidth overhead of repeating DC-nets for each broadcaster and have been successfully used for anonymous broadcast in other systems [2, 30, 41]. However, without access control, the DPFs must expand to a large space to prevent collisions. We show that our construction for single-channel access control extends to the multi-channel setting, where each broadcaster has a key associated with their allocated channel.

Preventing audit attacks. At a high level, our approach is to commit each server to the shares they receive from a client. In the case of an audit failure, each server efficiently proves that it adhered to protocol to blame the client; if it can't, any honest server aborts Spectrum.

4 Many channels and malicious servers

In this section, we extend the single-channel protocol of Section 3.2 to the multi-channel setting. We first show how to use a DPF to support many broadcast channels with little increase in bandwidth overhead (compared to the one-channel setting), an idea introduced in Riposte [30]. We prevent disruption by augmenting DPFs with the anonymous access control technique from Section 3.1. Prior works [13, 16, 30, 34, 41] describe techniques to verify that a DPF is well-formed, but do not allow for access control. Spectrum does both.

4.1 Tool: distributed point functions

A *point function* P is a function that evaluates to a message m on a single input j in its domain $\{1, \ldots, L\}$ and evaluates to zero on all other inputs $i \neq j$ (equivalently, a vector $(0, 0, \ldots, m, \ldots, 0)$). We define a *distributed* point function: a point function encoded and secret-shared among n keys:

Definition 1 (Distributed Point Function (DPF) [30, 46])**.** *Fix integers $L, n \geq 2$, a security parameter λ, and a message space $\mathcal{M}$. Let $\boldsymbol{e}_j \in \{0,1\}^L$ be the jth row of the $L \times L$ identity matrix. An n-DPF consists of (randomized) algorithms:*

- $\mathsf{Gen}(1^\lambda, m \in \mathcal{M}, j \in \{1, \ldots, L\}) \rightarrow (k_1, \ldots, k_n)$,
- $\mathsf{Eval}(k_i) \rightarrow (m_1, m_2, \ldots, m_L)$.

These algorithms must satisfy the following properties:

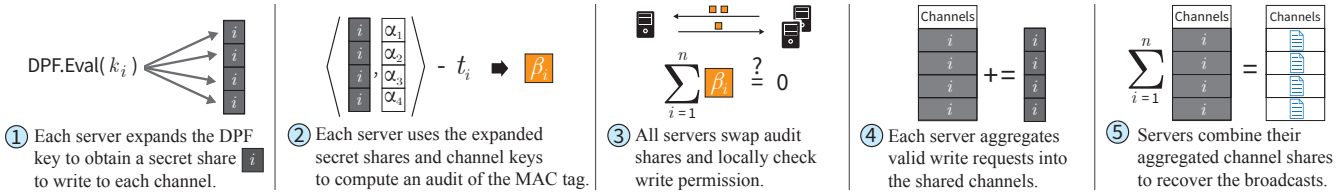
- Correctness. *A DPF is* correct *if expanding the output of* Gen *into the space of L messages $\mathcal{M}^L$ and combining gives the corresponding point function:*

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

  where the probability is over the randomness of Gen

- Privacy. *A DPF is* private *if any subset of evaluation keys reveals nothing about the inputs. That is, there exists an efficient simulator* Sim *which generates output computationally indistinguishable from strict subsets of the keys output by* Gen.

We use a DPF with domain $\{1, \ldots, L\}$, where each broadcaster/channel has an index $j \in \{1, \ldots, L\}$. Each broadcaster must write a message m to channel j, but not elsewhere: we can think of this as a point function P with $P(j) = m$. Then, we can encode secret-shares of P using a DPF more efficiently than secret-sharing its vector representation (as in repeated



① Each server expands the DPF key to obtain a secret share to write to each channel.
② Each server uses the expanded secret shares and channel keys to compute an audit of the MAC tag.
③ All servers swap audit shares and locally check write permission.
④ Each server aggregates valid write requests into the shared channels.
⑤ Servers combine their aggregated channel shares to recover the broadcasts.

Figure 2: Overview of the server-side pipeline when processing a client's request. Steps ①, ④ and ⑤ are computed over the field $\mathbb{F}$. Steps ② and ③ are computed "in the exponent" of the group $\mathbb{G}$ when using the technique described in "Preventing client-server collusion" of Section 3.1.

DC-nets). Evaluated DPF shares can still be aggregated locally, and our access control protocol supports DPFs with a slight modification (Section 4.2).

DPFs are concretely efficient. The key size for state-of-the-art 2-DPFs [16] is $O(\log L + |m|)$ (assuming PRGs); for the general case [15], when $n > 2$, the key size is $O(\sqrt{L} + |m|)$ under the decisional Diffie-Hellman assumption [10]. Server-side work to expand each DPF uses fast symmetric-key operations in the two-server case [15, 16] and group operations in the multi-server case [30]. With $L = 2^{20}$, the DPF key size for the two-server construction is 325 B and for the $n > 2$ construction 64 kB (excluding the message size).

4.2 Spectrum with many channels

In this section, we present the full Spectrum protocol with L channels and $n \geq 2$ servers. Broadcasters reserve a channel in the setup phase. Clients encode their message at their channel (if any) using a DPF; the servers anonymously audit access to all channels before recovering messages.

Setup. The setup in this setting is similar to the setup described in Section 3.2. Each broadcaster anonymously provides a public verification key g^{α_i} to the servers, to be associated with a channel. In addition to their key, any user with content to broadcast might upload a brief description or "teaser" of their content; the servers can choose which to publish, or users could perform a privacy-preserving vote [28]. We leave detailed exploration of the fair allocation of broadcast slots to users to future work. Post-setup, all servers hold a vector of L verification keys $(g^{\alpha_1}, \ldots, g^{\alpha_L})$. Each key corresponds to one channel.

Step 1: Sharing a message. Let $y = \alpha_j$ and $j' = j$ if the client is a broadcaster for the jth channel ($y = 0$ and $j' = 0$ otherwise). Only broadcasters have $m \neq 0$. Each client runs:

1.1: $(k_1, \ldots, k_n) \leftarrow \mathsf{DPF.Gen}(1^\lambda, m, j')$. `// gen DPF keys`

1.2: Compute $t \leftarrow m \cdot y \in \mathbb{F}$.

1.3: Sample $(t_1, \ldots, t_n) \xleftarrow{R} \mathbb{F}$ such that $\sum_{i=1}^n t_i = t \in \mathbb{F}$.

1.4: Send share (k_i, t_i) to the ith server, for $i \in \{1, \ldots, n\}$.

Step 2: Auditing shares. Upon receiving a request share (k_i, t_i) from a client, each server computes:

2.1: $\boldsymbol{m_i} \leftarrow \mathsf{DPF.Eval}(k_i) \in \mathbb{F}^L$.

2.2: $A \leftarrow \prod_{j=1}^L (g^{\alpha_j})^{\boldsymbol{m_i}[j]}$. `//` $A = g^{\langle \boldsymbol{m_i}, (\alpha_1, \ldots, \alpha_L) \rangle}$

2.3: $g^{\beta_i} \leftarrow A / g^{t_i}$.

2.4: Send g^{β_i} to all other servers.

All servers check that $\prod_i^n g^{\beta_i} = g^0 = 1_{\mathbb{G}}$. If this condition does not hold, then the client's request is dropped by all servers. In Section 4.3, we show how to detect a malicious server that tampers with a client's request so that it fails this audit.

Step 3: Recovering the broadcast. Each server keeps an accumulator $\boldsymbol{m_i}$ of L entries (i.e., the channels), initialized to $\mathbf{0} \in \mathbb{F}^L$. Let $S = \{(k_j, t_j) \mid j \leq N\}$ be the set of all valid requests that pass the audit of Step 2. Each server:

3.1: Computes $\boldsymbol{m_i} \leftarrow \sum_{(k,t) \in S} \mathsf{DPF.Eval}(k) \in \mathbb{F}^L$.

3.2: Publicly reveals $\boldsymbol{m_i}$. `// shares of the aggregate.`

Using the publicly revealed shares, anyone can recover the L broadcast messages as $\widehat{\boldsymbol{m}} = \sum_i^n \boldsymbol{m_i} \in \mathbb{F}^L$.

4.3 BlameGame: preventing audit attacks

BlameGame is a network overlay protocol that verifies who received what during protocol execution.

We use a *verifiable* encryption scheme [11, 20] where a party with a secret key can prove that a ciphertext decrypts to a certain message (DecProof makes a proof, and VerProof verifies it; see definition in Appendix C.1). Verifiable encryption reveals the plaintext request shares of a client to all servers if the client or the server is malicious (a malicious server may do this once, but will be immediately eliminated). BlameGame also uses a Byzantine broadcast protocol [19] so that all servers get the (encrypted) shares of all other servers.

BlameGame. BlameGame commits clients and servers to specific requests used in the audit. If the audit fails, honest servers reveal (with a publicly verifiable proof) the share they were given, which allows other servers to verify the results of the audit locally, which indicts the client. Dishonest servers cannot give valid proofs for their shares.

Setup. All servers make a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ and publish pk_i.

Step 1: Generating commitments. Let τ_i be the client's request secret-share for server i. The client runs:

1.1: $C_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \tau_i)$. `// Encryption under` pk_i.

1.2: Byzantine broadcast *all* C_i to *all* servers.

Server i recovers $\tau_i \leftarrow \mathsf{Dec}(\mathsf{sk}_i, C_i)$; clients may go offline at this point. All servers are committed to the *encryption* of their secret-shares. We describe an optimization in Section 5.1 that makes the size of each C_i constant.

Step 2: Proving innocence. Each server publishes their share of the request τ_i and a proof of correct decryption:

2.1: $(\pi_i, \tau_i) \leftarrow \mathsf{DecProof}(\mathsf{sk}_i, C_i)$.

2.2: Send (π_i, τ_i) to all servers.

Step 3: Assigning blame. Using the posted shares and proofs, each server assigns blame:

3.1: Collect (π_i, τ_i) from servers $i \in \{1, \ldots, n\}$ and all C_i.

3.2: Check that $\mathsf{VerProof}(\mathsf{pk}_i, \pi_i, C_i, \tau_i) = \mathsf{yes}$, for $1 \le i \le n$.

3.3: Check the audit using all the shares $(\tau_1, \ldots, \tau_n)$.

3.4: Assign blame:

if 3.2 fails for any i: abort; `// bad server`
else if 3.3 passes: abort; `// bad server`
else if 3.3 fails: drop the client request. `// bad client`

5 Optimizations and extensions

Here, we describe extensions and optimizations to Spectrum. We show how to (1) broadcast *large* messages efficiently and (2) *privately* fetch published broadcasts as a subscriber.

5.1 Handling large messages efficiently

We described Spectrum in Section 4.2 with messages as elements of a field $\mathbb{F}$, which we check to perform access control. While a 16 B field suffices for audit security, large messages require much larger fields (or repeating the protocol many times). These approaches require proportionally greater bandwidth and computation to audit. Instead, we give a black-box transformation from a 2-server DPF over $\mathbb{F}$ to a DPF over ℓ-bit strings, *preserving security* (see Section 6.2). We use a pseudorandom generator (PRG). Clients create DPF keys encoding a short PRG seed, rather than a message. The servers efficiently audit this seed as before to enforce access control. Then, they expand it to a much longer message with the guarantee that the DPF is still non-zero at an index for which the client knows the broadcast key (if the message is non-zero).

The transformation. Let DPF be a DPF over the field $\mathbb{F}$ and let $\mathsf{DPF}^{\mathrm{bit}}$ be a DPF over $\{0,1\}$. Let $G : \mathbb{F} \to \{0,1\}^\ell$ be a PRG. To write to channel j, a user computes:

1. $\bar{s} \xleftarrow{R} \mathbb{F}$. `// random nonzero PRG seed`
2. $(k_A, k_B) \leftarrow \mathsf{DPF.Gen}(\bar{s}, j)$.
3. $s^*_A \leftarrow \mathsf{DPF.Eval}(k_A)[j]$, $s^*_B \leftarrow \mathsf{DPF.Eval}(k_B)[j]$.
4. $\bar{m} \leftarrow G(s^*_A) \oplus G(s^*_B) \oplus m$.
5. $(k^{\mathrm{bit}}_A, k^{\mathrm{bit}}_B) \leftarrow \mathsf{DPF}^{\mathrm{bit}}.\mathsf{Gen}(1, j)$.
6. Send $(\bar{m}, k_A, k^{\mathrm{bit}}_A)$ to ServerA, $(\bar{m}, k_B, k^{\mathrm{bit}}_B)$ to ServerB.

Every server evaluates the DPF keys to a vector $\boldsymbol{s}$, of PRG seeds, and a vector $\boldsymbol{b}$ of bits. Each seed and bit other than the jth is *identical* on both servers (a secret-share of zero); at j, we get $s^*_A \neq s^*_B$. Servers evaluate the DPF by expanding each $\boldsymbol{s}[i]$ to an ℓ-bit string and XORing $\bar{m}$ only when $\boldsymbol{b}[j] = 1$. If we define multiplication of a binary string by a bit as $1 \cdot \bar{m} = \bar{m}$ and $0 \cdot \bar{m} = 0$, ServerA computes:

$$\boldsymbol{m}_A := (G(\boldsymbol{s}_A[1]) \oplus \boldsymbol{b}_A[1] \cdot \bar{m}, \ldots, G(\boldsymbol{s}_A[L]) \oplus \boldsymbol{b}_A[L] \cdot \bar{m}) .$$

ServerB does the same. Then, we get that:

$$\boldsymbol{m}_A[i] \oplus \boldsymbol{m}_B[i] = \begin{cases} G(\boldsymbol{s}[i]) \oplus G(\boldsymbol{s}[i]) = 0^\ell & i \neq j \\ G(s^*_A) \oplus G(s^*_B) \oplus \bar{m} = m & i = j . \end{cases}$$

Servers perform the audit (in $\mathbb{F}$) over the expanded PRG seeds and bits as in Section 3.2. Observe that the final output is non-zero only if: (1) some PRG seed, (2) some bit, or (3) the masked message $\bar{m}$ is different on each server. The $\boldsymbol{s}$ and $\boldsymbol{b}$ audit checks (1) and (2); servers check (3) by comparing hashes of $\bar{m}$. As before, the 0 MAC tag passes the audit for an empty message, and broadcasters can provide a correct tag.

Many servers. The above transformation generalizes to the n-server setting. The intuition is the same: only "non-zero" PRG seeds should expand to write non-zero messages. However, we need a PRG with special properties for this to hold with $n > 2$. We give the full transformation in Appendix B. Applying this transformation to a square-root DPF yields the n-server DPF of Corrigan-Gibbs et al. [30], but now with access control.

BlameGame optimization. The masked message $\bar{m}$, given to all servers, constitutes the bulk of data in *each* DPF key, so clients can omit it in their request commitments (Section 4.3) when using the above transformation because servers do not need it to verify access control. (The verification performed by the servers only depends on the DPF *seeds* and checking equality of the masked message $\bar{m}$.)

5.2 Private broadcast downloads

Content published using an anonymous broadcast system is likely to be sensitive and subscribers might want to have plausible deniability when it comes to which broadcasts they are interested in. In a setting with many channels, we might allow the subscribers to download one channel while hiding *which* channel they download: the exact setting of private information retrieval (PIR) [24]. In (multi-server) PIR, a client submits *queries* to two or more servers, receiving *responses* which they combine to recover one document in a "database." The queries hide which document was requested. In Spectrum, clients can use any PIR protocol to hide which channel they download. Modern PIR schemes based on DPFs have minimal bandwidth overhead for queries [15, 16]. However, the processing time on each server is always linear [24]. We

evaluate the overhead of using PIR for subscriber anonymity in Section 7.1.

6 Security and efficiency analysis

In this section, we analyze the theoretical efficiency and security of Spectrum with respect to the threat model and required guarantees outlined in Section 2.4.

6.1 Efficiency analysis

We briefly analyze the efficiency of Spectrum (Section 4.2) and BlameGame (Section 4.3) with the above optimizations.

Communication efficiency in Spectrum. Spectrum can use any DPF construction with outputs in a finite field using the transformation of Section 5.1 to support ℓ-bit messages with only an additive $O(\ell)$ overhead to the DPF key size. Using optimized two-server DPF constructions [15, 16], clients send requests of size $O(\log L + |m|)$ (for L channels). With more than two servers, the communication is $O\left(\sqrt{L} + |m|\right)$ using the seed-homomorphic PRG based DPF construction [30]. For the audit, inter-server communication is constant.

Computational efficiency in Spectrum. Each server performs $O(L \cdot |m|)$ work per client when aggregating the shares and performing the audit ($O(N \cdot L \cdot |m|)$ total for N clients). The work on each client is $O(\log L + |m|)$ when using two-server DPFs and $O(\sqrt{L} + |m|)$ otherwise [15].

6.2 Security of Spectrum

We first describe the ideal functionality of the anonymous broadcast system which Spectrum instantiates.

Ideal functionality. Ideal Spectrum is defined as follows:

- Receive message $m = 0$ from each subscriber, $m = \widehat{m}$ from the broadcaster, and no input from the servers.
- Output $\widehat{m}$ to both the clients and servers.

Client anonymity. We argue that Spectrum provides client anonymity by constructing a simulator for the view of a network adversary corrupting any strict subset of servers.

Claim 1. *If at least one server is honest, then no probabilistic polynomial time (PPT) adversary $\mathcal{A}$ observing the entire network and corrupting any strict subset of the servers and an arbitrary subset of clients, can distinguish between an honest broadcaster and an honest subscriber.*

Proof. We construct a simulator Sim for the view of $\mathcal{A}$ when interacting with an honest client. Let $\widehat{\mathsf{Sim}}$ be the DPF simulator (see Definition 1). Sim proceeds as follows:

1. Take as input $(\mathbb{G}, g)$, $(g^{\alpha_1}, \ldots, g^{\alpha_L})$, $\mathbb{F}$, and subset of corrupted server indices $I \subset \{1, \ldots, n\}$.
2. Sample $t_i \xleftarrow{R} \mathbb{F}$ for $i \in \{1, \ldots, n\}$ such that $\sum_i t_i = 0$.
3. $\{k_i \mid i \in I\} \leftarrow \widehat{\mathsf{Sim}}(I)$. `// see Definition 1`
4. Output $\mathsf{View} = \left(\left\{(t'_i, k_i) \mid i \in I\right\}, \{g^{t_j} \mid j \in \{1, \ldots, n\} \setminus I\}\right)$.

Analysis. The view includes:

- Each DPF key k_i for corrupted server i.
- Each MAC tag share t'_i for corrupted server i.
- Audit shares g^{t_j} from every honest server j.

The DPF keys are computationally indistinguishable from real DPF keys by the security of the DPF simulator. Therefore, it remains to argue that the tag and audit shares are distributed identically to the real view. Recall that during an audit, server i publishes $g^{\beta_i} = g^{\langle \boldsymbol{m_i}, (\alpha_1, \ldots, \alpha_L)\rangle - t_i}$ where $\boldsymbol{m_i}$ is the output of $\mathsf{DPF.Eval}(k_i)$ and t_i is a secret-share of the MAC tag t. For a subscriber, $\langle \boldsymbol{m_i}, (\alpha_1, \ldots, \alpha_L)\rangle$ (the inner product) gives a random secret share of 0 and t_i is a secret share of 0, so g^{β_i} is a random (multiplicative) secret share of g^0. For a broadcaster publishing to channel j, $\langle \boldsymbol{m_i}, (\alpha_1, \ldots, \alpha_L)\rangle$ is a random secret share of $m \cdot \alpha_j = t$, so g^{β_i} as computed by the ith server is a random multiplicative secret share of g^0 as well. Therefore, the distribution of the audit and tag shares (g^{β_i} and t_i, respectively) is identical to the real view. Finally, because the connection between clients and servers is encrypted (and of fixed-size), we can efficiently simulate network traffic as random encrypted data. □

Disruption resistance in Spectrum. We prove that a client cannot disrupt a broadcast on the jth channel without knowing the channel broadcast key α_j.

Claim 2. *Assuming the hardness of the discrete logarithm problem [11, 40] in $\mathbb{G}$, no probabilistic polynomial time (PPT) client can write to channel j and pass the audit performed by the servers without knowledge of α_j.*

Proof. Assume towards contradiction that some adversarial client can generate (potentially ill-formed) DPF keys that result in a non-zero vector (WLOG, assume that index L is non-zero) and pass the audit for a given access tag with non-negligible probability. We can use the client to extract the discrete logarithm for any element of $\mathbb{G}$ as follows. Given g^{α^*}, choose random $\alpha_i \in \mathbb{F}$ for $i \in \{1, \ldots, L-1\}$. Give the client $\left(g^{\alpha_1}, \ldots, g^{\alpha_{L-1}}, g^{\alpha^*}\right)$ and get in return DPF keys $(k_1, \ldots k_n)$ and MAC tag t. Given these DPF keys, we can compute $\boldsymbol{m} = (m_1, \ldots, m_L)$ by evaluating the DPF. If the shares pass the audit, it must be that $\langle \boldsymbol{m}, \boldsymbol{\alpha}\rangle = t$. However, $\boldsymbol{\alpha}$ includes α^* so we can solve for α^* (t and all α_i except for α^* are known). We conclude that the client has knowledge of α^*. □

Security of the DPF transformation. The construction from Section 5.1 maintains security. This construction transforms a DPF DPF into a DPF DPF' over ℓ-bit messages.

Claim 3. *If Spectrum with DPF preserves client anonymity, Spectrum with DPF' preserves client anonymity.*

Proof. We build a simulator Sim' for DPF' from the simulator Sim for DPF. Sim' simply runs Sim twice (once to generate the seed-DPF keys and once for the bit-DPF keys) and picks an ℓ-bit message uniformly at random for $\bar{m}$. The simulator's $\bar{m}$ is computationally indistinguishable from the real $\bar{m}$ (otherwise, the PRG used to mask the message is not secure). Therefore, if there exists an efficient distinguisher, it can also distinguish between the keys output by Sim and the real DPF keys, a contradiction. □

Claim 4. *If Spectrum with* DPF *has disruption resistance, Spectrum with* DPF' *has disruption resistance.*

Proof. Assume, towards contradiction, that there exists a computationally bounded adversary $\mathcal{A}$ which does *not* obtain the broadcast key α as input, and outputs a set of DPF' keys along with MAC tag shares. If the set of DPF' keys write to at least one channel and the tag shares output by $\mathcal{A}$ pass the server MAC audit, then we can produce a non-zero message and tag for DPF as follows. WLOG, we fix the number of servers to $n = 2$. Run $\mathcal{A}$ to get two DPF' keys k'_1, k'_2 and tag $t = (t_1, t_2)$. By construction, $k'_i = (k_i, k_i^{\text{bit}}, m')$, where k_i and k_i^{bit} are DPF keys with range $\mathbb{F}_p$ and $\mathbb{F}_2$, respectively, and $m' \in \mathbb{F}_{2^\ell}$ is a masked message (identical in each DPF' key). If these keys and tags pass the audit, the masked message in each key is the same (by the collision resistance of the audit hash function). Then, because the key for DPF' writes a non-zero message, at least one of the two DPF keys (either k or k^{bit}) must write a non-zero message (otherwise the keys would be writing zero). If follows that (k_1, k_2) or $(k_1^{\text{bit}}, k_2^{\text{bit}})$ encode a non-zero message, which contradicts Claim 2. □

6.3 Security of BlameGame

We must show that in BlameGame: (1) an honest client will never be blamed, (2) a malicious client will always be blamed, (3) an honest server will never be blamed, and (4) a malicious server will always be blamed. Incorrect blame attribution indicates a failure of the verifiable encryption scheme or audit security; see Appendix C.2 for full proof.

Overhead of BlameGame. BlameGame requires some extra bandwidth and computation time. Clients send a shared message mask *once* to each server; DPF keys add about 100 bytes per client request (details in Section 5.1). The servers must run BlameGame for each malicious client. However, verifying decryption takes tens of *microseconds*, and running the audit is similarly quick (see Section 7.1). Because the servers delay the work of aggregating messages until *after* the audit, a malicious client often requires *fewer* cycles than an honest one (but extra network communication).

7 Evaluation

We build and evaluate Spectrum, comparing it to state-of-the-art anonymous broadcasting works: Riposte, Blinder, Express, and Dissent (see related work; Section 8).

Riposte [30] is designed for anonymous broadcasting where all users broadcast at all times. Riposte uses three servers (one trusted for audits) but generalizes to many servers (one honest). Riposte was designed for smaller messages and the source code fails to run with messages of size 5 kB or greater.

Blinder [2] builds on Riposte but requires an *honest majority* of at least 5 servers. Like Riposte, Blinder also assumes that all users are broadcasting. Blinder supports using a server-side GPU to increase throughput.

Express [41] is an anonymous communication system designed for anonymous "dropbox"-like applications. It does not support broadcast as-is, but can be easily modified to do so. We include Express in our comparison as a recent, high-performance system decoupling broadcasters and subscribers.

Dissent [29, 95] has a setup phase (like Spectrum's), a DC-net phase, and a blame protocol. We give measurements both with and without the blame protocol and exclude the setup phase. Without the blame protocol, the system runs a plain DC-net without any disruption resistance and is quite fast. If *any* user sends an invalid message, Dissent runs the (expensive) blame protocol (up to once per malicious user).

We use data from the Blinder paper [2, Fig. 4] as the source did not compile. The Dissent code (last modified in 2014) ran with up to 1000 users and 10 kB messages, but hung indefinitely after increasing either (though the authors report 128 kB messages with 5000 users). Linearly scaling our measurements, we find them broadly consistent (3× faster) with the authors' reported measurements for 128 kB messages with the same number of users in a similar setting [95, Fig. 7].

Implementation. We build Spectrum in ∼8000 lines of open-source [1] Rust code, using AES-128 (CTR) as a PRG and BLAKE3 [66] as a hash. Because our DPF has relatively few "channels" L, a DPF with $O(L)$-sized keys (adapted from Corrigan-Gibbs et al. [30]) gives the best concrete performance. For the multi-server extension (Section 5.1 and Appendix B), we use a seed-homomorphic PRG [12] with the Jubjub [49] curve. We encrypt traffic with TLS 1.3 [73].

Environment. We run VMs on Amazon EC2 to simulate a WAN deployment. Each `c5.4xlarge` 8-core instance has 32 GiB RAM [76], running Ubuntu 20.04 ($0.68 per hour in September 2021). We run clients in `us-east-2` (Ohio) and servers in `us-east-1` (Virginia) and `us-west-1` (California). Network round trip times (RTTs) were 11 ms between Virginia and Ohio, 50 ms between Ohio and California, and 61 ms between Virginia and California. Inter-region bandwidth was 524 Mbit/s (shared between many clients simulated on the same machine).

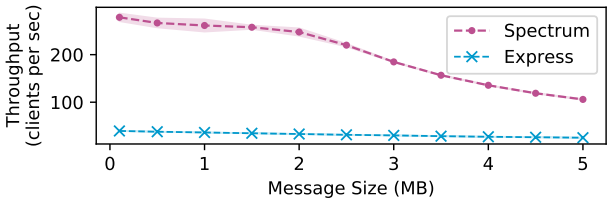


Figure 3: Throughput (client requests per second; higher is better) for a one channel deployment (one broadcaster and many subscribers).

7.1 Results

In our experiments, we find Spectrum is 4–7× faster than Express for 5 MB to 100 kB messages, 2× / 13–17× slower than Blinder (CPU/GPU, resp.) in *unfavorable* settings, 500–7500× / 250–520× faster than Blinder (CPU/GPU) in *favorable* settings, and 16–12,500× faster than Riposte. We run 5 trials per setting, shading the 95% confidence interval (occasionally too small to be seen).

One channel. In Figure 3, we report the throughput (client requests per second) for both Spectrum and Express in the one-channel setting. As expected, performance is worse with larger messages for both systems. However, we find that Spectrum, compared to Express, is 4–7× faster on messages between 100 kB and 5 MB. Riposte and Blinder have no analog for the single-channel setting. (Dissent *does* support a one-channel setting, but did not run with large messages.)

Many channels. Unlike Riposte and Blinder, Spectrum is faster with fewer broadcasters. To compare, we fix 100,000 users and vary the number of channels from 1 (best-case for Spectrum) to 100,000 (worst-case). We evaluate Spectrum with and without the change described in "Preventing client-server collusion" (Section 3.1). Without the change, which we call *"Spectrum (sk)"*, servers obtain the MAC secret key for each channel. This mirrors the threat model of e.g., Express [41]. *With the change*, servers only get MAC *public keys* which prevents covert client-server collusion. However, there is a modest price in terms of performance due to the elliptic curve operations (see Figure 4). We find that Spectrum (both variants) outperform all other systems with 10 kB messages for relatively few channels (up to hundreds), but performs relatively worse with smaller messages or more channels. For "Twitter-like" settings, another system (e.g., Blinder or Riposte) may be appropriate.

Overhead. In any anonymous broadcast scheme, every client (even subscribers) must upload data corresponding to the message length $|m|$ to ensure 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, roughly 75× smaller than in Express. Moreover, in Spectrum, request audits are under 16 B, a 120× improvement over Express [41]. BlameGame imposes little

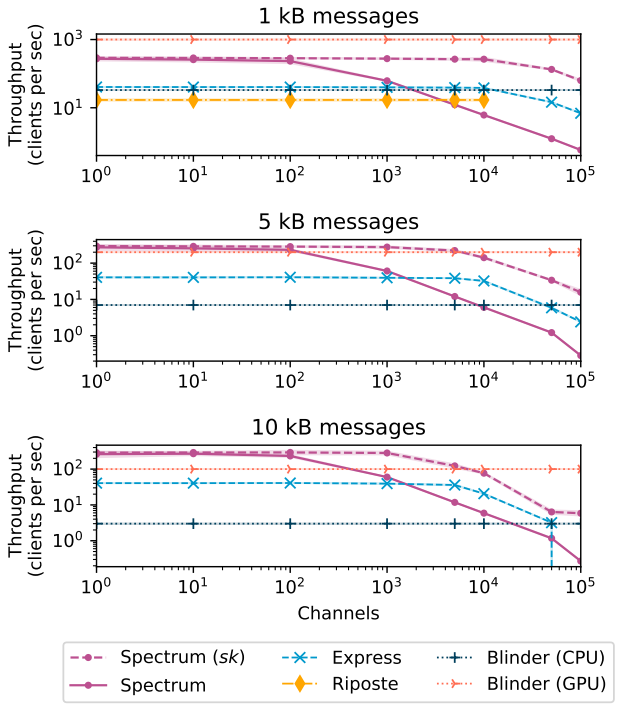


Figure 4: Throughput (requests per second; higher is better) for broadcasts with 100,000 users with varying numbers of broadcasting users ("channels"): Express and Spectrum benefit from fewer channels. (Blinder numbers as reported by the authors [2].)

overhead (both in terms of bandwidth and computation). Because BlameGame runs only when a request audit fails, these overheads occur for few requests in most settings.

Many servers. In Section 5.1 and Appendix B, we note that our construction of Spectrum generalizes from 2 to n servers (with one honest) in a similar manner to Riposte [30]. The n-server construction uses a seed-homomorphic pseudorandom generator (PRG) [12]. We note that when using the DDH-based seed-homomorphic PRG, only "Spectrum (sk)" access control is possible (see Remark 1 in Appendix B for more details). On one core of an AMD Ryzen 4650G CPU, we measured the maximum throughput of our seed-homomorphic PRG at 300 kB/s, 20,000 times slower than

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 request size for one channel and $|m|$-bit messages. BlameGame only runs if the first request audit fails.

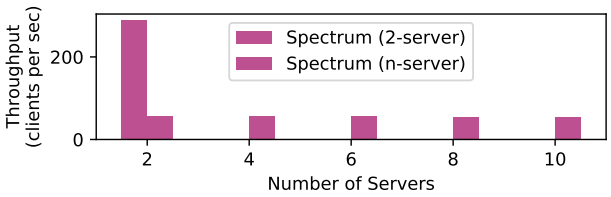


Figure 5: Spectrum can generalize to $n > 2$ servers (shown for 10 kB messages). This uses an expensive *seed-homomorphic* PRG and is therefore slower, but adding more servers causes no additional slowdown.

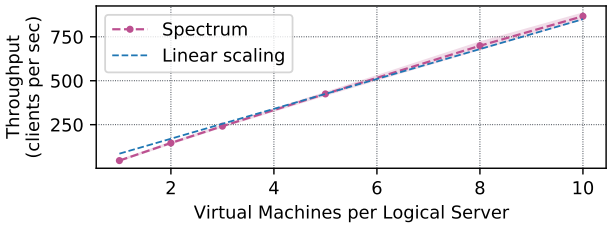


Figure 6: Spectrum is highly parallelizable: for 500 channels of 100 kB messages, 10 VMs per "server" gives a 10× speedup.

an AES-based PRG. For 10,000 kB messages, Spectrum was 5× slower with the seed-homomorphic PRG (Figure 5); with larger messages, the relative difference increases. We find *no additional* slowdown between 2 to 10 servers. An interesting direction for future work would to evaluate Spectrum with LWE-based seed-homomorphic PRG constructions [12], as they are likely to have better concrete performance.

Scalability. We may trust machines administered by the same *organization* equally, viewing several worker servers as one logical server. Client requests trivially parallelize across such workers: running 10 workers per logical server leads to a 10× increase in overall throughput (Figure 6). In a cloud deployment, Spectrum handles the same workload in less time for negligible additional cost by parallelizing the servers.

Latency. In Figure 7, we measure the time to broadcast a single document for these systems with varying numbers of users. For Spectrum, we use a 1 MB message. For Blinder, we use numbers reported by the authors [2, Fig. 4], multiplied to the same message size (the authors explicitly state that repeating the scheme many times is the most efficient way to send large messages). We benchmark Dissent both with and without the blame protocol invoked during a round. The former (blame) is the performance of Dissent if any client misbehaves. The latter (no blame) assumes that no client misbehaves. Express doesn't have a notion of "rounds" so we omit it here. We find that for one channel of large messages, Spectrum is much faster than other systems (except Dissent with no blame protocol; i.e., when all clients are honest).

Client privacy. In Section 5.2, we outlined how private information retrieval (PIR) [24] techniques provide client

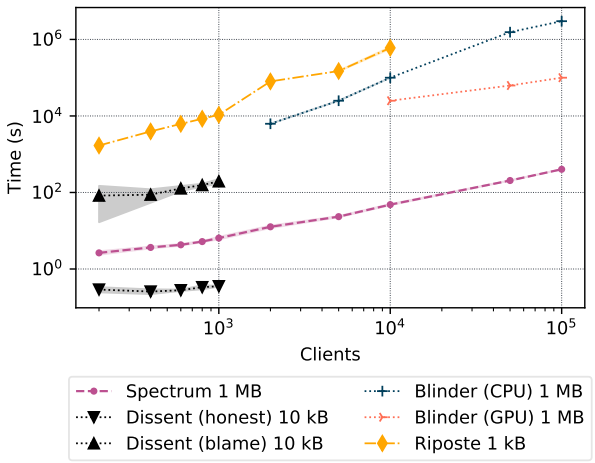


Figure 7: Latency for uploading a single document with varying numbers of users. Blinder numbers as reported by the authors [2, Fig. 4] and linearly scaled to 1 MB messages.

L	up	down
10^1	1.25 B	$\|m\|$
10^2	12.5 B	$\|m\|$
10^3	125 B	$\|m\|$
10^4	1.25 kB	$\|m\|$
10^5	12.5 kB	$\|m\|$

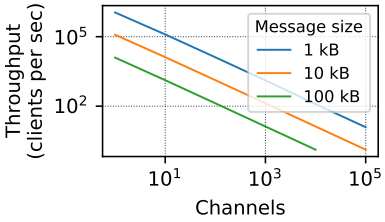


Figure 8: **Left:** Bandwidth usage of a PIR query with varying number of channels L. **Right:** Server capacity (one core) to answer PIR queries for private client downloads. For L channels, the client requests one out of L documents, where channels have size $|m|$.

privacy for multiple channels. Figure 8 shows the server-side CPU capacity to process these requests for 1 kB, 10 kB, and 100 kB messages and 1–100,000 channels. We measure one core of an AMD Ryzen 4650G CPU for a simple 2-server PIR construction [24], finding good concrete performance.

7.2 Discussion

Our evaluations showcase the use of Spectrum for a real-world anonymous broadcasting deployment using commodity servers. Compared to the state-of-the-art in anonymous broadcasting, Spectrum achieves speedups in settings with a large ratio of passive subscribers to broadcasters. Based on our evaluation, with 10,000 users, Spectrum could publish: a *PDF document* (1 MB) in 50s, a *podcast* (50 MB) in 40m, or a *documentary movie* (500 MB, the size of Alexei Navalny's documentary on Putin's Palace at 720p [81]) in 6h40m.

Operational costs. We estimate costs for a cloud deployment of Spectrum using current Amazon EC2 prices, reported in US dollars. Servers upload about 100 bytes per query (in the above settings, at most 1 GB per day) and inbound traffic is free on EC2. We focus on compute costs: $6.84 per GB

published through Spectrum (with 10,000 users). Table 3 compares costs to publish 1 GB among 10,000 users.

System	Cost (USD)
Blinder (GPU)	$2,000,000.00
Blinder (CPU)	$250,000.00
Riposte	$218,000.00
Dissent (with blame protocol; one round)*	$76,000.00
Dissent (honest clients)	$134.00
Express	$30.22
Spectrum	$6.84

Table 3: Cost to upload one 1 GB document anonymously with 10,000 users, based on the *best* observed rate for each system with that many users (that is, the maximum throughput over all settings we measured; for Blinder, we use the best reported rate). We multiply the total time at the maximum throughput by hourly rate to get the cost. *Extrapolated from 1000 users.

8 Related work

Existing systems for anonymous broadcast are suitable for 140 B to 40 kB [2, 30, 41] broadcasts, orders of magnitude smaller than large data dumps [69, 75, 77] common today.

Mix Networks and Onion Routing. In a mix net [22], users send an encrypted message to a proxy server, which collects and forwards these messages to their destinations in a random order. Chaining several such hops protects users from compromised proxy servers and a passive network adversary. Mix nets and their variations [32, 56, 57, 59, 60, 63, 64, 71, 72, 82, 83, 91] scale to many servers. However, because messages are exchanged and shuffled between many servers, mix nets are poorly suited to high-bandwidth applications. Atom [55] uses mix nets with zero-knowledge proofs to horizontally scale anonymous broadcast to millions of users (Spectrum achieves about 12,500× the throughput [55, Fig. 9]). Riffle [54] uses a *hybrid verifiable shuffle*; in the broadcast setting, it shares a 300 MB file with 500 users in 3 hours (Spectrum supports about 10,000 users in that time).

Some systems use onion routing for better performance than 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 [37], the most popular onion routing system, has millions of daily users [84]. Tor provides security in many real-world settings, but is vulnerable to traffic analysis [53, 62, 78]. If only one user sends large volumes of data, an adversary can identify them—Tor discourages high bandwidth applications for this and other reasons [36].

DC-nets. Another group of anonymous communications systems use dining cryptographer networks (DC-nets) [23] (Section 2). DC-nets are vulnerable to disruption: any malicious participant can clobber a broadcast by sending a "bad" share. Dissent [29, 95] augments the DC-nets technique with a system for accountability. Like Spectrum, Dissent performs best if relatively few users are broadcasting. The core data sharing protocol is a standard DC-net, which is very fast and supports larger messages. Further, it supports many servers at little additional cost. However, Dissent is not suitable for many-user applications where disruption is a concern. If *any* user misbehaves, Dissent must undergo an expensive blame protocol (quadratic in the total number of users). This approach detects, rather than prevents, disruption. The user is evicted after this protocol, but an adversary controlling many users can cause many iterations of the blame protocol.

PriFi [6] builds on the techniques in Dissent to create indistinguishability among clients in a LAN. Outside servers help disguise traffic using low-latency, precomputed DC-nets. Like Dissent, PriFi catches disruption after-the-fact using a blame protocol (as often as once per malicious user). The PriFi blame algorithm is much faster, but still scales with *all* users in the system (in Spectrum, each malicious user incurs constant server-side work).

Riposte [30] enables anonymous Twitter-style broadcast with many users using a DC-net based on DPFs and an auditing server to prevent disruptors. We find that Riposte is 16× slower than Spectrum with 10,000 users. Further, Riposte assumes that all users are broadcasting and therefore gets *quadratically* slower in the total number of users.

A more recent work, Blinder [2] uses multi-party computation to prevent disruption. Blinder's threat model requires at least five servers with an honest majority. Like Spectrum, Blinder is resilient to active attacks by a malicious server. It is fast for small messages when most users have messages to share, but much slower for large messages. Blinder allows trading money for speed with a GPU.

Express [41] is a system for "mailbox" anonymous communication (writing anonymously to a designated mailbox). Express also uses DPFs for efficient write requests. However, it only runs in a two-server deployment. Express is *not* a broadcasting system, and while it is possible to adapt it to work in a broadcast setting, it is not designed to withstand active attacks by the servers and is insecure for such an application (see Appendix A for details).

9 Conclusions

Spectrum supports high-bandwidth, low-latency broadcasts from a small set of broadcasters to a large number of subscribers by applying new tools to the classic DC-net architecture. We prevent disruption by malicious clients with an efficient blind access control mechanism that prevents clients from writing to a channel they do not have access to.

Additionally, we introduce optimizations to decouple server-side overhead from the message size, which allows Spectrum to scale to large messages and many broadcasters. To prevent malicious servers from deanonymizing clients, we

develop a lightweight blame protocol to abort Spectrum if a server deviates from the protocol. Our experimental results show that Spectrum can be used for uploading gigabyte-sized documents anonymously among 10,000 users in 14 hours.

10 Acknowledgments

We thank Henry Corrigan-Gibbs, Kyle Hogan, Albert Kwon, and Derek Leung for helpful feedback and discussion on early drafts of this paper. We thank Guy Zyskind for suggesting a clarification to the multi-server access control mechanism (Remark 1 in Appendix B). We would also like to thank our shepherd Alan Liu and the anonymous NSDI reviewers for their insightful feedback and many suggestions that helped to significantly improve this paper.

References

[1] Spectrum implementation. https://www.github.com/znewman01/spectrum-impl, 2021.

[2] Ittai Abraham, Benny Pinkas, and Avishay 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. ISBN 9781450370899. doi: 10.1145/3372297.3417261. URL https://doi.org/10.1145/3372297.3417261.

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

[4] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 551–569, 2016.

[5] Raymond Walter Apple Jr. 25 years later; lessons from the Pentagon Papers. *The New York Times*, 23 June 1996. URL https://www.nytimes.com/1996/06/23/weekinreview/25-years-later-lessons-from-the-pentagon-papers.html. Accessed March 2022.

[6] Ludovic Barman, Italo Dacosta, Mahdi Zamani, Ennan Zhai, Apostolos Pyrgelis, Bryan Ford, Joan Feigenbaum, and Jean-Pierre Hubaux. Prifi: Low-latency anonymity for organizational networks. *Proc. Priv. Enhancing Technol.*, 2020(4):24–47, 2020. doi: 10.2478/popets-2020-0061. URL https://doi.org/10.2478/popets-2020-0061.

[7] Josh Daniel Cohen Benaloh. *Verifiable secret-ballot elections*. PhD thesis, Yale University, 1987.

[8] Charles Berret. Guide to SecureDrop, 2016. URL https://www.cjr.org/tow_center_reports/guide_to_securedrop.php.

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

[10] Dan Boneh. The decision Diffie-Hellman problem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, pages 48–63, 1998. doi: 10.1007/BFb0054851. URL https://doi.org/10.1007/BFb0054851.

[11] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Recuperado de https://crypto. stanford. edu/˜ dabo/cryptobook/BonehShoup_0_4. pdf*, 2017.

[12] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In *Annual Cryptology Conference*, pages 410–428. Springer, 2013.

[13] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776. IEEE, 2021.

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

[15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 337–367, Berlin, Heidelberg, 2015. Springer. ISBN 978-3-662-46803-6.

[16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016.

[17] Russ Buettner, Susanne Craig, and Mike McIntire. Long-concealed records show Trump's chronic losses and years of tax avoidance. *The New York Times*, 2020. URL https://www.nytimes.com/interactive/2020/09/27/us/donald-trump-taxes.html. Accessed March 2022.

[18] Bryan Burrough, Sarah Ellison, and Suzanna Andrews. The Snowden saga: A shadowland of secrets and light. *Vanity Fair*, 2014. URL https://www.vanityfair.com/news/politics/

2014/05/edward-snowden-politics-interview. Accessed March 2022.

[19] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2001. doi: 10.1007/3-540-44647-8_31. URL https://doi.org/10.1007/3-540-44647-8_31.

[20] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer, 2003. doi: 10.1007/978-3-540-45146-4_8. URL https://doi.org/10.1007/978-3-540-45146-4_8.

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

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

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

[24] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[25] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 719–728, 2017.

[26] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas 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.

[27] Henry Corrigan-Gibbs. *Protecting Privacy by Splitting Trust*. PhD thesis, Stanford University, 2019.

[28] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 259–282, 2017.

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

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

[31] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[32] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III anonymous remailer protocol. In *2003 Symposium on Security and Privacy, 2003.*, pages 2–15. IEEE, 2003.

[33] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy Pass: Bypassing internet challenges anonymously. *Proc. Priv. Enhancing Technol.*, 2018(3):164–180, 2018.

[34] Leo de Castro and Antigoni Polychroniadou. Lightweight, maliciously secure verifiable function secret sharing. *Cryptology ePrint Archive*, 2021.

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

[36] Roger Dingledine. BitTorrent over Tor isn't a good idea, Apr 2010. URL https://blog.torproject.org/bittorrent-over-tor-isnt-good-idea.

[37] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[38] Emily Dreyfuss. Chelsea Manning walks back into a world she helped transform, 2017. URL https://www.wired.com/2017/05/chelsea-manning-free-leaks-changed/.

[39] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, pages 139–147, 1992. doi:

10.1007/3-540-48071-4_10. URL https://doi.org/10.1007/3-540-48071-4_10.

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

[41] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In *30th USENIX Security Symposium (USENIX Security 21)*, Vancouver, B.C., August 2021. USENIX Association. URL https://www.usenix.org/conference/usenixsecurity21/presentation/eskandarian.

[42] Nathan S Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on Tor using long paths. In *USENIX Security Symposium*, pages 33–50, 2009.

[43] Cassi Feldman. 60 Minutes' most famous whistleblower. *CBS News*, 2016. URL https://www.theguardian.com/world/2010/nov/28/how-us-embassy-cables-leaked. Accessed March 2022.

[44] Lorenzo Franceschi-Bicchierai. Snowden's favorite chat app is coming to your computer. *Vice*, 2015. URL https://www.vice.com/en/article/signal-snowdens-favorite-chat-app-is-coming-to-your-computer. Accessed March 2022.

[45] Anita Gates and Katharine Q. Seelye. Linda Tripp, key figure in Clinton impeachment, dies. *The New York Times*, 2020. URL https://www.nytimes.com/2020/04/08/us/politics/linda-tripp-dead.html. Accessed March 2022.

[46] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 640–658, Berlin, Heidelberg, 2014. Springer. ISBN 978-3-642-55220-5.

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

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

[49] Daira Hopwood. Jubjub supporting evidence. https://github.com/daira/jubjub, 2017. Accessed March 2022.

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

[51] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks: Communications and Multimedia Security, IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99), September 20-21, 1999, Leuven, Belgium*, pages 258–272, 1999.

[52] Laurie Kazan-Allen. In memory of Henri Pezerat. http://ibasecretariat.org/mem_henri_pezerat.php, 2009. Accessed March 2022.

[53] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of Tor hidden services. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 287–302, 2015.

[54] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134, 2016.

[55] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 406–422. ACM, 2017.

[56] Albert Kwon, David Lu, and Srinivas Devadas. XRD: Scalable messaging system with cryptographic privacy. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 759–776, Santa Clara, CA, February 2020. USENIX Association. ISBN 978-1-939133-13-7. URL https://www.usenix.org/conference/nsdi20/presentation/kwon.

[57] David Lazar, Yossi Gilad, and Nickolai 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.

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

[59] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. Towards efficient traffic-analysis resistant anonymity networks. *ACM SIGCOMM Computer Communication Review*, 43 (4):303–314, 2013.

[60] Stevens Le Blond, David Choffnes, William Caldwell, Peter Druschel, and Nicholas 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.

[61] Jason Leopold, Anthony Cormier, John Templon, Tom Warren, Jeremy Singer-Vine, Scott Pham, Richard Holmes, Azeen Ghorayshi, Michael Sallah, Tanya Kozyreva, and Emma Loop. The FinCEN Files. *BuzzFeed News*, 2020. URL https://www.buzzfeednews.com/article/jasonleopold/fincen-files-financial-scandal-criminal-networks. Accessed March 2022.

[62] Shuai Li, Huajun Guo, and Nicholas 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.

[63] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew 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.

[64] Prateek Mittal and Nikita 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.

[65] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita 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.

[66] Jack O'Connor, Samuel Neves, Jean-Philippe Aumasson, and Zooko Wilcox-O'Hearn. BLAKE3: One function, fast everywhere, 2020. URL https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf. Accessed March 2022.

[67] John O'Connor. "I'm the guy they called Deep Throat". *Vanity Fair*, 2006. URL https://www.vanityfair.com/news/politics/2005/07/deepthroat200507. Accessed March 2022.

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

[69] Paradise Papers reporting team. Paradise Papers: Tax haven secrets of ultra-rich exposed. *BBC News*, 2017. Accessed March 2022.

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

[71] Ania M Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The Loopix anonymity system. In *26th USENIX Security Symposium USENIX Security 17)*, pages 1199–1216, 2017.

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

[73] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) protocol version 1.3. RFC 1654, RFC Editor, July 1995. URL https://www.rfc-editor.org/rfc/rfc1654.txt.

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

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

[76] Amazon Web Services. Amazon EC2 instance types. https://aws.amazon.com/ec2/instance-types/, 2022. Accessed March 2022.

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

[78] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew 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.

[79] David Smith. Trump condemned for tweets pointing to name of Ukraine whistleblower. *The Guardian*, 2019. URL https://www.theguardian.com/us-news/2019/dec/27/trump-ukraine-whistleblower-president. Accessed March 2022.

[80] Von Spiegel Staff. Inside the NSA's war on internet security. *Der Spiegel*, 2014. URL https://www.spiegel.de/international/germany/inside-the-nsa-

s-war-on-internet-security-a-1010361.html. Accessed March 2022.

[81] The BBC. Putin critic Navalny jailed in Russia despite protests. URL https://www.bbc.com/news/world-europe-55910974. Accessed March 2022.

[82] The Freenet Project. Freenet, 2020. URL https://freenetproject.org/.

[83] The Invisible Internet Project. I2P anonymous network, 2020. URL https://geti2p.net/en/.

[84] The Tor Project. Tor metrics, 2019. URL https://metrics.torproject.org/.

[85] The Wall Street Journal. Got a tip? https://www.wsj.com/tips, 2020. Accessed March 2022.

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

[87] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 423–440, 2017.

[88] US Holocaust Memorial Museum. Röhm purge. *Holocaust Encyclopedia*, 2020. URL https://encyclopedia.ushmm.org/content/en/article/roehm-purge. Accessed March 2022.

[89] US Occupational Safety and Health Administration. The whistleblower protection program. https://www.whistleblowers.gov/, 2020. Accessed March 2022.

[90] US Securities and Exchange Commission. Office of the whistleblower. https://www.sec.gov/whistleblower, 2020. Accessed March 2022.

[91] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152. ACM, 2015.

[92] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: using hard AI problems for security. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 294–311, 2003. doi: 10.1007/3-540-39200-9_18. URL https://doi.org/10.1007/3-540-39200-9_18.

[93] Lei Wang, Kazuo Ohta, and Noboru 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.

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

[95] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 179–182, 2012.

[96] Kim Zetter. Jolt in WikiLeaks case: Feds found Manning-Assange chat logs on laptop. *Wired*, 19 December 2011. URL https://www.wired.com/2011/12/manning-assange-laptop/. Accessed March 2022.

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 [55] and Blinder [2] 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 7 and 8). However, a malicious server can then exploit the verification procedure [41, Section 4.1] to exclude a user, changing their request to an invalid distributed point function. This excludes the message from the final aggregation, deanonymizing a broadcaster with probability at least $\frac{1}{(1-\epsilon)N}$ per round (where ϵ is the fraction of corrupted clients). Even with a few rounds, this can lead to a successful deanonymization of a broadcaster *without detection* (honest servers cannot tell if a server is cheating and therefore cannot abort the 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 multiple servers and assuming an

honest majority (as in Blinder [2]), 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 a 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 (see Section 5.1).

B Large message optimization (multi-server)

In Section 5.1, we give a transformation from a 2-server DPF over a field $\mathbb{F}$ to a 2-server DPF over ℓ-bit bitstrings that preserves the auditability of the first DPF without increasing the bandwidth overhead proportionally. Here, we show a more general transformation from n-server DPFs over a field $\mathbb{F}$ to n-server DPFs over a group $\mathbb{G}_y$ of a polynomially larger order. Our transformation uses a seed-homomorphic pseudorandom generator (PRG) [12].

Definition 2 (Seed-homomorphic Pseudorandom Generator)**.** *Fix groups $\mathbb{G}_s, \mathbb{G}_y$ with respective operations $\circ_s$ and $\circ_y$. A seed-homomorphic pseudorandom generator is a polynomial-time algorithm $G : \mathbb{G}_s \to \mathbb{G}_y$ with the following properties:*

- Pseudorandom. *G is a PRG: $|\mathbb{G}_s| < |\mathbb{G}_y|$, with output computationally indistinguishable from random.*
- Seed-homomorphic. *For all $s_1, s_2 \in \mathbb{G}_x$, we have $G(s_1 \circ_s s_2) = G(s_1) \circ_y G(s_2)$.*

Let $\mathbb{G}$ be a group over a field $\mathbb{F}$ and in which the decisional Diffie-Hellman (DDH) problem [10, 11, 40] is assumed to be hard. Fix some DPF with messages in $\mathbb{F}$. We saw in Section 4.2 how to implement anonymous access control for such DPFs. Let $G : \mathbb{F} \to \mathbb{G}_y$ be a seed homomorphic PRG where $\mathbb{G}_y$ is over $\mathbb{F}$. Boneh et al. [12] give a construction of such a PRG for $\mathbb{G}_y = (\mathbb{G})^L$ from the DDH assumption in $\mathbb{G}$.

Then, the larger DPF key for a message m is a DPF key k_1 for a random value $s \in \mathbb{F}$, a DPF key k_2 for $1 \in \mathbb{F}$, and a "correction message" $\bar{m} = m \circ_y G(s)^{-1}$ (each key has the *same* correction message). For a zero message, the larger DPF key is two DPF keys k_1, k_2 for $0 \in \mathbb{F}$ and a random correction message $\bar{m}$.

To evaluate the DPF key, the server computes $s \leftarrow \mathsf{DPF.Eval}(k_1)$, $b \leftarrow \mathsf{DPF.Eval}(k_2)$, and $(\bar{m})^b \circ_y G(s)$. If $s = 0$, then combining the DPF keys gives $(\bar{m})^0 \circ_y G(0) = 1_{\mathbb{G}_y}$. Otherwise, we get $(\bar{m})^1 \circ_y G(s) = m$.

To perform access control for the larger DPF, perform access control for k_1 and k_2 and then also check for the equality of the hashes of $\bar{m}$. We note this construction does not yield a new DPF, but does add authorization to a large class of existing DPFs.

Remark 1. *Depending on the seed-homomorphic PRG construction used (the most efficient one being DDH-based [12]), it may or may not be possible to apply the transformation described in "Preventing client-server collusion" (Section 3.1). Specifically, because the output group of the PRG is already in a group $\mathbb{G}$, and* not *$\mathbb{Z}_p$, we cannot "shift" the audit to the exponent of a group where the discrete logarithm problem is hard. Using a lattice-based seed-homomorphic PRG [12], which does output in $\mathbb{Z}_p$, is an alternative option, however.*

C BlameGame

C.1 Verifiable Encryption

BlameGame (Section 4.3) uses a verifiable encryption scheme [20], which allows a prover to decrypt a ciphertext c and create a proof that c is an encryption of a message m. We formalize these schemes below:

Definition 3 (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 many public key encryption schemes (e.g., ElGamal [40]) satisfy Definition 3 out-of-the-box and can be used to instantiate BlameGame.

C.2 BlameGame security

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

Soundness. BlameGame is *sound* if no honest client or server will ever be blamed:

1. For all honest commitments C_i, no probabilistic polynomial-time (PPT) adversary can create a request share τ_i and proof of decryption π_i such that the BlameGame "Assigning blame" step (Section 4.3) blames the client when run with (π_i, τ_i, C_i).

2. No PPT adversary can create commitments C_i such that an honestly-created request share τ_i and proof of decryption π_i will result in blaming the server after running the BlameGame "Assigning blame" step.

Privacy. The privacy requirement of BlameGame is similar to that of Spectrum. Specifically, the commitments C_i must not reveal any information about the request to any subset of servers. Formally: for randomly sampled pairs of keys pk_i and sk_i (for $i \in \{1,\ldots,n\}$), and all proper subsets $I \subset \{1,\ldots,n\}$, the following distributions are computationally indistinguishable:

$$\{(\mathsf{pk}_i, \mathsf{sk}_i)\ \forall i \in I,\ C_i\ \forall i \in \{1,\ldots,n\}\} \approx_c \{(\mathsf{pk}_i, \mathsf{sk}_i)\ \forall i \in I,\ C'_i\ \forall i \in \{1,\ldots,n\}\}$$

where the C_i are created by honestly encrypting request shares corresponding to a cover request by a subscriber and the C'_i are created by honestly encrypting shares corresponding to any valid write generated by a broadcaster.

We note that BlameGame does **not** require any privacy properties during blame assignment, as it may reveal the request for the purpose of assigning blame.

We now show that BlameGame achieves these properties:

Proof. We prove each property in turn.

Soundness (honest client). Suppose, toward contradiction, that there exists a PPT adversary $\mathcal{A}$ that generates some request shares τ_i and proof of decryption π_i such that BlameGame blames the client. This means that (1) the decryption proof verification succeeds, and (2) running the audit with the request shares failed. By property (1), we can assume that τ_i is a correct decryption of C_i and π_i is a valid proof of decryption; otherwise, $\mathcal{A}$ breaks the soundness property of the verifiable encryption scheme. However, we know that running the audit with the given request shares will pass, because (by assumption) they were created honestly by the client. This is a contradiction.

Soundness (honest server). Let τ_i be a set of request tokens such that the Spectrum audit fails when run with τ_i. Suppose, toward contradiction, that some client creates commitments $C_1,\ldots,C_n$ for $\tau_1,\ldots\tau_n$ such that the BlameGame "Assigning blame" step blames some server (instead of the client, as required). Then, it holds that either (1) the proof of decryption failed, or (2) the audit performed by the servers over the decrypted requests passes. However, if (1) is true (the proof of decryption failed), then the completeness property of the verifiable encryption scheme does not hold (because the request share and proof of decryption are generated honestly by the server). Therefore, we are left with (2); the audit performed by the servers over the decrypted request shares passes. However, this isn't true (by assumption) if the client is malicious. Hence, we have a contradiction.

Privacy. For all honest broadcasters, 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* broadcaster gets deanonymized), the protocol aborts if the second audit fails (an honest broadcaster would have encrypted the request correctly). In this case, all servers see the request which deanonymizes 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 $\frac{L}{N\cdot(1-\epsilon)}$. □

Spectrum (with BlameGame) achieves our desired security properties: a malicious client cannot cause disruption, and a malicious server cannot deanonymize a broadcaster. Because BlameGame is sound, if all servers are honest then Spectrum does not abort (because either the audit passes, or BlameGame blames the client); this prevents disruption due to audit failure. The second property follows from the privacy of BlameGame.