Spectrum: High-bandwidth Anonymous Broadcast with Malicious Security

Zachary Newman
MIT CSAIL
zjn@mit.edu

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

Srinivas Devadas
MIT CSAIL
devadas@csail.mit.edu

Abstract

We present Spectrum, a high-bandwidth, metadata-private file broadcasting system with malicious security guarantees. In Spectrum, a small number of broadcasters share a document with many subscribers via two or more non-colluding servers. Subscribers generate cover traffic, hiding which users are broadcasters and which users are simply consumers.

To drastically improve latency and throughput, Spectrum optimizes for few broadcasters and many subscribers, common in real-world broadcast settings. To prevent malicious clients from disrupting broadcasts with malformed requests, we introduce a new blind access control technique that allows servers to reject malicious users. We also ensure security against malicious servers which collude with clients.

We implement and evaluate Spectrum. Compared to the state-of-the-art in cryptographic anonymous communication systems, Spectrum's peak throughput is 4–12,500× faster (and commensurately cheaper) in a broadcast setting. Deployed on two commodity servers, Spectrum allows broadcasters to share 1 GB in 13h 20m with an anonymity set of 10,000 (for a total cost of about $6.84). This corresponds to an anonymous upload of two full-length 720p documentary movies. Operational costs scale roughly linearly in the size of the file and total number of users, and Spectrum parallelizes trivially with more hardware.

1 Introduction

Free and democratic society depends on an informed public, which sometimes depends on whistleblowers shedding light on misdeeds and corruption. Over the last century, whistleblowers have exposed financial crimes and government corruption [51, 59, 65], risks to public health [34, 42], presidential misconduct [11, 36, 58, 69], war and human rights crimes [4, 30, 77], and digital mass surveillance by U.S. government agencies [12]. Political philosophers debate [2, 27] the ethics of whistleblowing, but agree it often has a positive impact.

These whistleblowers take on risks in bringing misdeeds to light. The luckiest enjoy legal protections [78] or financial reward [79]. But many face exile [12] or incarceration [40, 60, 64]. Recently, activist Alexei Navalny was sentenced to prison after releasing of documents accusing Russian president Vladimir Putin of corruption and embezzlement [70].

Many whistleblowers turn to encrypted messaging apps to protect themselves [35, 38, 74]. Secure messaging apps Signal [19] and SecureDrop [6] have proven to be an important resource to whistleblowers and journalists [35, 74]. Encryption works, but cannot provide privacy from powerful adversaries capable of observing network *metadata*.

The source, destination, timing, and size of encrypted data can leak information about its contents. For example, prosecutors used SFTP metadata in the case against Chelsea Manning [86]; a federal judge found Natalie Edwards guilty on evidence of metadata from an encrypted messaging app [40]. Whistleblowing systems must provide users metadata privacy.

Many metadata-hiding systems provide application-specific solutions to this problem. Some recent research systems [1, 22, 33, 44–46, 48, 76, 80] provide precise security guarantees for both anonymous messaging and "Twitter"-like broadcast applications. However, these systems slow dramatically with large numbers of users or large messages. Tor [29] is widely used, but unsuitable for high-bandwidth applications. We compare to related work in Section 8.

Contributions. In this work, we introduce Spectrum, the first anonymous broadcast system supporting high-bandwidth, many-user settings with security against actively malicious clients and servers. It optimizes for the many-subscriber and few-broadcaster setting, which reflects the real-world usage of broadcast platforms. Spectrum's costs scale with the number of *broadcasts* in the system rather than the total number of users, a primary cause of latency in prior work. With Spectrum, users can share very large messages in comparison with prior work.

This paper contributes:

1. design and security analysis of Spectrum, a system for high-bandwidth broadcasting with strong robustness and privacy guarantees,
2. a notion of blind access control for anonymous communication using distributed point functions [37], along with a construction and a black-box transformation to efficiently support arbitrarily large messages,
3. BlameGame, a blame protocol to "upgrade" anonymous broadcast protocols for security against active de-anonymization attacks, and
4. an open-source implementation of Spectrum, evaluated in comparison to other state-of-the-art anonymous communication systems.

Limitations. Spectrum shares some limitations with other metadata-private systems. First, Spectrum provides anonymity among honest online users and requires all users to contribute cover messages to a broadcast (to perfectly hide network metadata), *uploading* as much data as a broadcaster to provide anonymity. However, these cover messages can provide cover for *multiple* broadcast messages at the same time, with little overhead for each additional message (Section 4). Additionally, Spectrum achieves peak performance with exactly two servers (as do recent works in anonymous broadcast [22, 33]) due to the speed of the relevant cryptographic primitives [9].

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 (*broadcasters*) share a *message* (e.g., file) while preventing network observers or other users 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. Users use a *client* to communicate with the system. Because the message sources are anonymous, the servers publish distinct messages in different *channels* or slots. Every broadcaster has exactly one channel, which they anonymously publish to in every iteration of the protocol.

2.1 DC-nets

Chaum [17] presents DC-nets, which enable a rudimentary form of anonymous broadcast assuming all parties are honest. As in prior work [1, 22, 33, 84], we instantiate a DC-net with two or more servers and many clients (in contrast to Chaum's work, which assumes that all parties participate in the DC-net). The motivation behind employing a small number of servers rather than using a fully decentralized DC-net is due to the improved efficiency that can be achieved as well as the ability to guarantee robustness against malicious clients.

DC-nets use secret-sharing to obscure the source of data in the network. One of the clients (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 secret share $r_i \oplus m_i$ to $\mathsf{ServerA}$ and secret share r_i to $\mathsf{ServerB}$. 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 aggregating all shares: $\mathsf{agg}_A = \bigoplus_i (r_i \oplus m_i)$ and $\mathsf{agg}_B = \bigoplus_i r_i$. Because all subscribers send shares of zero, only the broadcaster's message emerges from the aggregation. That is,

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

To see this, observe that:

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

This scheme protects client anonymity, as each server sees a uniformly random share from each client. Revealing the aggregation hides which client submitted $\widehat{m}$.

DC-net challenges. While DC-nets allow fast anonymous broadcast, users can undetectably *disrupt* the broadcast by sending non-zero shares. A major challenge with DC-nets is preventing malicious clients from disrupting broadcasts [1, 22, 33, 45]; a primary cause of high latency in prior work [1, 21, 22, 44, 45, 84] (see related work; Section 8). Also, while DC-nets enable one broadcaster to transmit a message, many clients may wish to broadcast to separate *channels*, or slots. Repeating the protocol in parallel is inefficient, requiring bandwidth linear in the number of broadcasters. Even prior works which overcome this challenge require more channels than broadcasters (sometimes exponentially [33] more channels than broadcasters) to prevent collisions. Additionally, other works [1, 22] require that *each user* broadcasts a message for security, a waste of bandwidth in many realistic settings.

2.2 Main ideas in realizing Spectrum

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

Preventing disruption. In Spectrum, we prevent broadcast disruption by developing a new tool: *anonymous* access control (Section 3.1.1), which we build from the Carter-Wegman MAC [83], and adapt to our setting by using a new observation about its properties. Servers check access to each "channel" to ensure that only a user with a "broadcast key" can write to that channel.

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 naive scheme, they still reserve space for *every* client. As a consequence, the total computation on each server is quadratic in the number of clients, leading to high latency and much "wasted" work. Spectrum derives anonymity from *all* clients, but only the total number of *broadcasters* (rather than the total number of users in the system) influences the per-client work on each server.

Preventing "audit" attacks. Anonymous broadcast servers can *covertly* exclude a client in order to de-anonymize 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 de-anonymize them. Abraham et al. [1] make the same observation and defend against the attack by requiring an honest-majority out of five or more servers; Corrigan-Gibbs and Ford [21] prevent it with an expensive, after-the-fact blame protocol. Other prior works [22, 33] *are* vulnerable. Spectrum is the first system to efficiently defend against this attack using minimal assumptions—we only require that at least one server is honest—while still preventing disruption (rather than blaming users afterwards). We achieve this by introducing BlameGame, a lightweight blame protocol which applies to other anonymous broadcast systems as well (see Section 4.3).

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 exactly one *channel*, or slot, for their message. Subscribers do not have a channel. Spectrum publishes a message to each channel in every execution, but hides whether a user published a message, and if so, which one. 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.

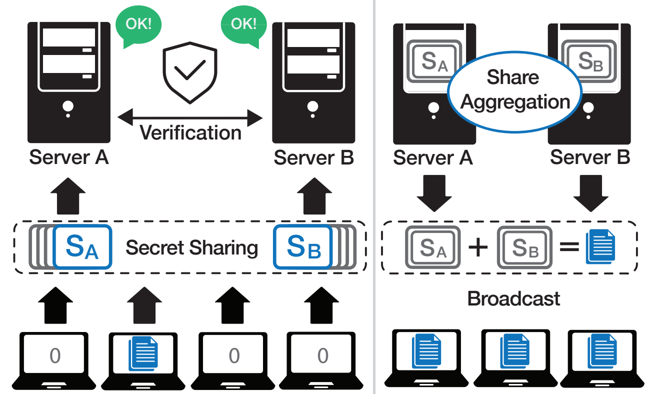


Figure 1: In Spectrum, users upload secret shares to the servers. Servers validate and combine these shares to recover the broadcast message while hiding its provenance.

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, alternately, is writing a zero message. If the message shares pass the audit, the servers aggregate them as in the vanilla DC-net (Section 2.1). Otherwise, the servers perform a blame protocol (see BlameGame, described below). Finally, the servers combine their 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. Clients *cannot* use this protocol to slow down execution of Spectrum (see Section 6.2).

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. Clients may deviate from protocol, collude with other clients, or collude with a subset of malicious servers.
- Only one server must be honest and trusted by clients for privacy. Any subset of servers may deviate from protocol or collude with a subset of malicious clients.

Assumptions. As with prior work [1, 22, 45], we assume the entire network is observed. We only require one non-colluding server. We make black-box use of public key infrastructure (e.g., TLS [63]) to encrypt data between clients and servers. We make use of several standard cryptographic assumptions: (1) hardness of the discrete logarithm problem [32], (2) the decision Diffie-Hellman assumption [7] for more than two servers, (3) the existence of collision-resistant hash functions, and (4) pseudorandom generators. We also assume a setup-free anonymous broadcast system [1, 22, 45, 84] for bootstrapping Spectrum.

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

- *Anonymity.* An adversary controlling a strict subset of servers and clients cannot distinguish between honest clients: broadcasters and subscribers look the same.
- *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 aborts.

Non-goals. As with prior works, we do not protect against denial-of-service attacks by a large number of clients (but we note that standard techniques, such as CAPTCHA [81], anonymous one-time-use tokens [26], or proof-of-work [31, 41] apply). Like all anonymous broadcast systems, intersection attacks on metadata 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

The single-channel version of Spectrum follows the DC-net construction of Section 2.1. 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). 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 access key) can send a non-zero message; all subscribers (who do not have the access 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.

3.1.1 New tool: anonymous access control

We adapt the Carter-Wegman MAC [15, 83] to provide a secret-shared "access proof" accompanying the message shares. Each client sends a secret-shared proof that it is sending either: (1) a share of a broadcast message with knowledge of the access key; or (2) a cover message (i.e., $m_i = 0$) that does not affect the final aggregate computed by the servers. Crucially, this proof does not reveal *which* of these conditions holds when the servers verify it.

Carter-Wegman MAC. Let $\mathbb{F}$ be any finite field of sufficiently large size for security. Sample a random authentication key $(\alpha, \gamma) \in \mathbb{F} \times \mathbb{F}$. We define a Carter-Wegman MAC as:

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

A verifier can check the authenticity of a message m' given a tag t by computing $t' \leftarrow \mathsf{MAC}_{(\alpha,\gamma)}(m')$ and checking if $t = t'$. Forgery (different messages $m \neq m'$ but same tag $t = t'$) is infeasible and the probability of guessing the authentication key (α, γ) is negligible [67]. 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 $\mathsf{ServerA}$ and $\mathsf{ServerB}$. 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:

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

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.

It turns out that $\mathsf{MAC}_{(\alpha,\gamma)}$ is *almost* sufficient to prevent disruption. 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, a malicious server can share (α, γ) with a malicious client who can then covertly disrupt a broadcast.

	Request Size	Audit Size	Audit Rounds	Server Work	Malicious Security	Disruption Handling	Blame Protocol	Comments
Blinder [1]	$\lvert m\rvert \cdot \sqrt{N}$	$\lambda \cdot \lvert m\rvert$	$\log N$	$N \cdot \lvert m\rvert$	✓	Prevent	N/A	Requires 5+ servers and MPC
Dissent [21]	$\lvert m\rvert \cdot L + N$	N/A	N/A	$L \cdot \lvert m\rvert$	✓	Detect	Expensive	Blame quadratic in N
PriFi [5]	$\lvert m\rvert \cdot L + N$	N/A	N/A	$L \cdot \lvert m\rvert$	✓	Detect	Expensive	Similar to Dissent
Riposte [22]	$\lvert m\rvert + \sqrt{N}$	$\sqrt{N}$	1	$N \cdot \lvert m\rvert$	✗	Prevent	✗	Requires a separate audit server
Express [33]	$\lvert m\rvert + L$	λ	1	$L \cdot \lvert m\rvert$	✗	Prevent	✗	Exactly 2 servers
Two-Server	$\lvert m\rvert + \log(L)$	λ	1	$L \cdot \lvert m\rvert$	✓	Prevent	Lightweight	With tree-based DPF [9]
Multi-Server	$\lvert m\rvert + \sqrt{L}$	λ	1	$L \cdot \lvert m\rvert$	✓	Prevent	Lightweight	With seed-homomorphic DPF [8, 22]

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 depending on L, not N (Section 3.3), a request size efficient in L (Section 5.1), a new protocol for fast, small audits (Section 3.1.1), and a new, fast blame protocol for malicious security and disruption resistance (Section 4.3).

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 [24] multi-party computation protocol. The γ value acts solely as a "nonce" to prevent forgeries on the message $0 \in \mathbb{F}$ [82]. 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. That is, subscribers can "forge" a valid tag but *only* for $m = 0$.

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

Preventing client-server collusion. To address the second problem, we need to prevent the servers from learning the access key α while still preserving the ability to verify access. We resolve this by shifting the entire verification "to the exponent" of a group $\mathbb{G}$ (over $\mathbb{F}$) where the discrete logarithm problem is assumed to be computationally intractable [75]. That is, servers obtain a verification key $g^\alpha \in \mathbb{G}$ but not α. Subscribers do not have any key. 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. With these, servers enforce access control as:

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

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. We formally argue security in Section 6.

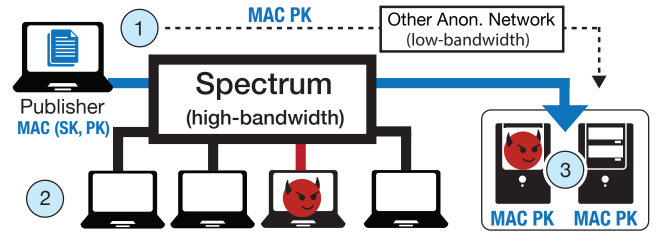


Figure 2: ① A broadcaster creates a new "channel" by sending a broadcast key to the servers via a (slow, low-bandwidth) anonymous network (e.g., Riposte [22]). ② The broadcaster can then broadcast using Spectrum (fast, high-bandwidth). ③ The servers (anonymously) verify that each client request is valid using the broadcast verification key—only the broadcaster can write to the channel.

3.2 Putting things together

In this section we combine the two-server DC-net construction for anonymous broadcast with the anonymous access control technique of Section 3.1.1 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 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 (illustrated in Figure 2). All clients use a slower anonymous broadcast system suitable for low-

bandwidth content at system setup time [1, 22, 45, 84]. The broadcaster shares a authentication key while subscribers share nothing. Keys are small (e.g., 32 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 [3, 21, 33, 48, 80, 84] and we note that Spectrum is agnostic to how this setup takes place.

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 access 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)`

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 with the two servers.

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

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

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

2.3: Servers swap g^{B_A} and g^{B_B} and check if $g^{B_A} \cdot g^{B_B} = g^0$.

The above follows the access control verification (Section 3.1.1). All shares that fail the audit are discarded by both servers. In Section 4, we show how to prevent "audit attacks," where a server tampers with a client request so that this 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.

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 secure with one broadcaster, it does not efficiently extend to multiple broadcasters. Second, a malicious server can perform an *audit attack* by tampering with the audit to make it fail for one or more clients—and learn whether one of them was a broadcaster (see Section 4.3).

Supporting multiple channels. To support multiple channels, we use distributed point functions (DPFs) 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 [1, 22, 33]. However, without access control, these DPFs must expand to a large space to prevent collisions [22, 33]. 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 cannot do so, any honest server aborts Spectrum.

4 Many channels and malicious security

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 *distributed point function* (DPF) [37] to support many broadcast channels with little increase in bandwidth overhead (compared to the one-channel setting), an idea introduced by Corrigan-Gibbs et al. [22]. We prevent disruption by augmenting DPFs with the anonymous access control technique from Section 3.1.1. Prior works [9, 10, 22, 33] describe techniques for verifying that a DPF is well-formed but do not provide a way to enforce 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 $[L]$ 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) [22, 37]). *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:*

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

These algorithms must satisfy the following properties:

Correctness. *A DPF is* correct *if evaluating and summing the output of* Gen *gives the corresponding point function:*

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

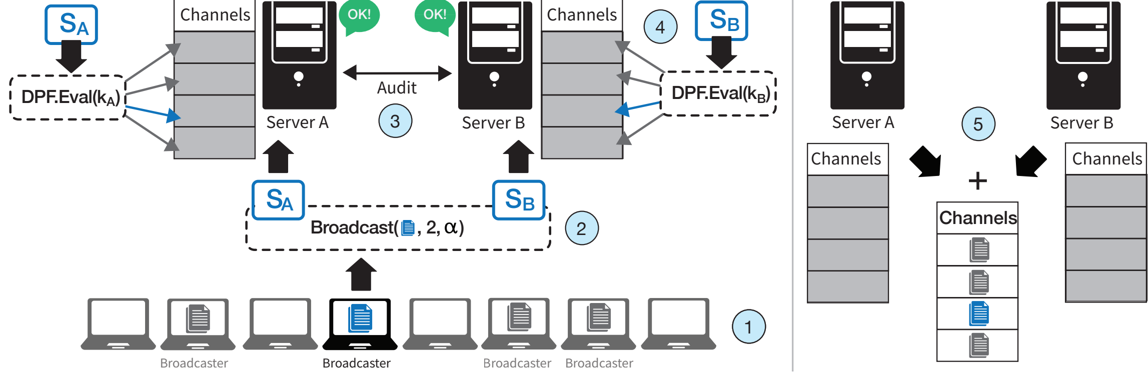


Figure 3: Spectrum with multiple broadcasters and subscribers. ① We trace one broadcaster's message. ② Each client sends secret shares to the servers consisting of a DPF key and access tag. ③ Servers verify the correctness of each client request by exchanging audit messages. ④ Servers locally aggregate all shares that pass the audit. ⑤ After processing requests from all clients, servers combine their aggregated shares to reveal the broadcast messages.

Privacy. *A DPF is* private *if any subset of evaluation keys reveals nothing about the inputs: 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 $[L]$; each index in the domain corresponds to a specific broadcaster/channel. Each broadcaster must write their message m to the channel j, but not elsewhere: we can think of this as a point function P with $P(j) = m$. Then, we can efficiently encode secret-shares of P using a DPF, which is much more efficient than simply secret-sharing its vector representation (as in repeated DC-nets). Evaluated DPF shares can still be combined locally in the manner of a DC-net, and our access control protocol carries over with slight modification (Section 4.2).

DPFs are concretely efficient. The key size for state-of-the-art 2-DPFs [10] is $O(\log L + |m|)$ (assuming PRGs); for the general case [9], when $n > 2$, the key size is $O(\sqrt{L} + |m|)$ under the decisional Diffie-Hellman assumption [7]. Server-side work to expand each DPF uses fast symmetric-key operations in the two-server case [9, 10] and group operations in the multi-server case [22]. For instance, with $L = 2^{20}$, the DPF key size for the two-server construction is 325 B and for the $n > 2$ construction 64 kB (plus 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 [20]. We leave detailed exploration of the fair allocation of broadcast slots to users to future work. Post-setup, both servers hold a vector of L verification keys $(g^{\alpha_1}, \ldots, g^{\alpha_L})$, where each key corresponds to a channel.

Step 1: Sharing a message. Let $y = \alpha_j$ and $j' = j$ if the client is a broadcaster for the jth channel ($y = 0$ 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 $t \leftarrow m \cdot y \in F$.

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 $i \in [n]$.

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

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

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

2.4: Send B_i to all other servers.

All servers check that $\prod_i^n B_i = g^0 = 1_{\mathbb{G}}$. 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 clients request, causing the audit to fail.

Step 3: Recovering the broadcast. Each server keeps an accumulator $\boldsymbol{m_i}$ of L entries, initialized to $\boldsymbol{0} \in \mathbb{F}^L$. 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

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 [45] and Blinder [1] solve this by using verifiable secret-sharing in an honest-majority setting; however, this can be costly in practice. Others do not prevent this attack.

4.3.1 Audit attacks in prior work.

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 [33, Section 4.1] to exclude a user, changing their request to an invalid distributed point function. This excludes the message from the final aggregation, de-anonymizing a broadcaster with probability at least $\frac{1}{(1-\epsilon)N}$ per round (where ϵ is the fraction of corrupted clients). Over even a few rounds, this can lead to a successful de-anonymization of a broadcaster *without detection* (non-colluding 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 [1]), this weakens the threat model and reduces practical performance.

4.3.2 BlameGame.

BlameGame is a network overlay protocol that verifies who received what during a protocol execution. This allows honest servers to discover and blame malicious servers which attempt such an audit attack.

The BlameGame protocol applies immediately to both Riposte and Express to address this audit attack by allowing (non-colluding) 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.

A naive way to instantiate the functionality of BlameGame is using a trusted third party (TTP). The TTP receives all shares from the client and distributes them to each of the servers accordingly. If an audit fails, the TTP reveals all the shares it received from the client, which allows each server to determine whether the shares were bad (i.e., the client was malicious) or whether a server caused the failure by tampering with its share (i.e., the server was malicious). BlameGame achieves this functionality *without* relying on a TTP. Instead, we realize BlameGame using verifiable encryption and a public bulletin board.

Background. We use a *verifiable* encryption scheme [14] where a party with a secret key generates a proof that a ciphertext decrypts to a certain message (generated with $\mathsf{DecProof}$, verified with $\mathsf{VerProof}$; formally defined in Appendix B). Many public-key encryption schemes (e.g., Paillier [23] and ElGamal [32]) are verifiable. Further, BlameGame makes black-box use of a Byzantine broadcast protocol [13] (most of these protocols optimistically terminate after one round if the client is honest).

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 implicates 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 destination 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.[1]

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.

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

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

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 $(\pi_i, \tau_i, \sigma_i')$ and C_i, from servers $i \in [n]$.

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

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`

BlameGame commits clients to specific requests and checks that the servers acted in accordance with those requests. The protocol aborts if and only if a server was malicious—clients cannot impact availability. This detects any attempt at an audit attack. We argue the security of BlameGame in Section 6.2.

[1] An optimization in Section 5.1 makes the size of each C_i constant.

5 Optimizations and extensions

Here, we describe extensions and optimizations to Spectrum.

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 100 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 turn a 2-server DPF over $\mathbb{F}$ into a DPF over ℓ-bit strings, *preserving security* (see Section 6.1). Our main idea to avoid this overhead is based on the following insight: handling large messages as bit strings (rather than elements of $\mathbb{F}$) makes for more efficient server-side processing. However, *auditing* access control requires the shares to be in $\mathbb{F}$. Our black-box transformation uses a pseudorandom generator (PRG) to expand the auditable seed into a larger message. Clients create DPF keys encoding a short PRG seed, rather than a message. The servers perform their checks over this PRG seed, verifying that either the user knows the broadcast key or the seed *expands* to a non-zero value. Then, they expand the seed to a much longer message. This means that the audit is performed efficiently over small elements, but the clients still encode large messages for much less cost.

Consider the following approach to encoding shares of the message in the two-server setting. The client samples two random PRG seeds $(s_A, s_B) \in \mathbb{F}$. The share of each message is then $m_A = (s_A, \bar{m})$ and $m_B = (s_B, \bar{m})$ where $\bar{m} = G(s_A) \oplus G(s_B) \oplus m$. Servers can recover the message by combining shares as:

$$m = G(s_A) \oplus G(s_B) \oplus \bar{m}.$$

Sharing the message in this way allows us to expand the DPF into *PRG seeds*. Further, in the 2-server setting, we have $G(s) \oplus G(s) = 0$ for all $s \in \mathbb{S}$; we can check for $s_A = s_B$ to allow empty writes.

The transformation. Let DPF be a DPF over the field $\mathbb{F}$ and let $\mathsf{DPF}^{\text{bit}}$ 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. $(k_A, k_B) \leftarrow \mathsf{DPF.Gen}(\bar{s}, z)$.
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. $(k_A^{\text{bit}}, k_B^{\text{bit}}) \leftarrow \mathsf{DPF}^{\text{bit}}.\mathsf{Gen}(1, j)$.
6. Send $(\bar{m}, k_A, k_A^{\text{bit}})$ to $\mathsf{ServerA}$, $(\bar{m}, k_B, k_B^{\text{bit}})$ to $\mathsf{ServerB}$.

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);[2] 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 $0 \cdot \bar{m} = 0$, $\mathsf{ServerA}$ computes:

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

$\mathsf{ServerB}$ does the same. Then:

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

To send an empty message, a user can use the DPFs to write the 0 message, and choose a uniformly random "masked message" $\bar{m}$. This results in *identical* DPF keys for each server, so the zero message is written in both slots.
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. Auditing $\boldsymbol{s}$ and $\boldsymbol{b}$ as before 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 for (1) and (2), checking.

Many servers. This construction 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 $n > 2$. Namely, we use a "seed-homomorphic PRG" [8], which preserves additive relationships between seeds s_i and their expansions:

$$\sum_{i \in [n]} s_i = 0 \implies \sum_{i \in [n]} G(s_i) = 0.$$

This lets us check the small seeds for access, rather than the large messages, giving efficiency gains. We give the full transformation in Appendix A. Applying this transformation to a square-root DPF yields the n-server DPF of Corrigan-Gibbs et al. [22], but now with access control.

BlameGame. We note that by far the largest part of the transformed DPF keys is $\bar{m}$, the masked full-length message. In Section 4.3, we describe users sending *each* DPF key to *each* server. However, $\bar{m}$ is the same across all DPF keys. This means that users can just send $\bar{m}$ *once* to each server (servers must compare hashes of $\bar{m}$ to detect client equivocation). This means that the overhead of BlameGame is *independent* of the broadcast message length.

5.2 Private broadcast downloads

Content published using an anonymous broadcast system is likely to be politically sensitive. While Spectrum hides the

[2] To simplify presentation, we use an equivalent definition of DPFs in which the servers *subtract* (rather than add) their DPF evaluations to obtain the point function output.

broadcasters, it provides no such protection to the *subscribers* downloading the content. 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) [18].

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 uses minimal bandwidth for queries [9, 10]. We evaluate specific PIR schemes in Section 7.2.

5.3 Efficiency analysis

Here, we analyze the efficiency of Spectrum and BlameGame (Sections 4.2 and 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 [9, 10], clients send requests of size $O(\log L + |m|)$ (where L is the number of channels). With more than two servers, the communication is $\sqrt{L} + |m|$ when using the seed-homomorphic PRG based DPF construction [22]. The communication between servers when performing an audit is constant.

Computational efficiency in Spectrum. Each server performs $O(L \cdot |m|)$ work for each of the N clients when aggregating the shares and performing the audit. The work on each client is $O(\log L + |m|)$ when using two-server DPFs and $O(\sqrt{L} + |m|)$ otherwise [9].

6 Security analysis

Here, we analyze the security of Spectrum with respect to the guarantees outlined in the threat model of Section 2.4.

6.1 Security of Spectrum

Client anonymity. Spectrum provides client anonymity. We formally argue client anonymity by constructing a simulator for the view of a network adversary $\mathcal{A}$ corrupting any strict subset of servers and clients. Intuitively, such an adversary cannot differentiate between *any* pair of honest clients because their communications look the same: (1) network traffic from non-colluding parties is encrypted, (2) the DPF keys look indistinguishable as long as one server is not controlled by the adversary, and (3) the message tags that the adversary sees are secret-shared.

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 subset of the other servers and an arbitrary number of clients, can distinguish between an honest broadcaster and an honest subscriber within the anonymity set of all honest clients.*

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 $I \subset [n]$.
2. Sample $r_i \xleftarrow{R} \mathbb{F}$ for $j \in [n]$ such that $\sum_i r_i = 0$.
3. $\{k_i \mid i \in I\} \leftarrow \widehat{\mathsf{Sim}}(I)$. `// see Definition 1`
4. Output $\mathsf{View} = \left(\left\{(r_i', k_i) \mid i \in I\right\}, \{g^{r_j} \mid j \in [n] \setminus I\}\right)$.

Analysis. The view includes:

- DPF keys k_i for each corrupted server i.
- Tag shares r_i' from the client at each corrupted server i.
- Audit shares g^{r_j} from every server j.

The DPF keys are computationally indistinguishable from real DPF keys by the security of the DPF simulator [37]. We must argue that the MAC tag and audit shares are correctly distributed. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_L)$. Recall that during an audit, each server j publishes $g^{\langle \boldsymbol{m_i}, \boldsymbol{\alpha} \rangle - t_j}$ where $\boldsymbol{m_i}$ is the output of $\mathsf{DPF.Eval}(k_i)$ and t_j is a secret-share of the MAC tag t. For a subscriber, $\langle \boldsymbol{m_i}, \boldsymbol{\alpha} \rangle$ (the inner product) gives a secret share of 0 and t_j is a secret share of 0, so this value is a (multiplicative) share of g^0. For a broadcaster publishing to channel j, $\langle \boldsymbol{m_i}, \boldsymbol{\alpha} \rangle$ gives a secret share of $m \cdot \alpha_j = t$, so this value is a multiplicative secret share of g^0 as well. This is exactly the same distribution of the audit and tag shares. (Because each client message is encrypted and fixed-size, we simulate network traffic as uniformly 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 by reduction to the discrete logarithm problem [32] in $\mathbb{G}$. Intuitively, any algorithm that could reliably write any message without knowing the corresponding broadcast key could find discrete logarithms with the same probability, leading to a contradiction.

Claim 2. *To write to channel j and pass the audit performed by the servers constitutes a proof-of-knowledge of α_j.*

Proof. Assume towards contradiction that some adversarial client can generate (potentially ill-formed) DPF keys that evaluate to a non-zero vector and and corresponding access tag that passes the audit with non-negligible probability. Then, we can use this client to solve the discrete logarithm problem in $\mathbb{G}$: given g^{α^*}, choose random $\alpha_i \in \mathbb{F}$ for $i \in [L-1]$. 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 access tag t. Given these DPF keys, we can compute $\boldsymbol{m} = (m_1, \ldots, m_L)$. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{L-1}, \alpha^*)$. If

the shares pass the audit, we have $\langle \boldsymbol{m}, \boldsymbol{\alpha} \rangle = t \neq 0$. However, $\boldsymbol{\alpha}$ includes α^*, so we can solve for α^* because all α_i and t are known. Then, the discrete logarithm adversary succeeds exactly when the adversarial client succeeds, which happens non-negligibly. □

Security of 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 for Sim' for DPF' from the simulator Sim for DPF. Sim' simply runs Sim twice and picks an ℓ-bit message uniformly at random. We have that the simulator's random message is computationally indistinguishable from the real message (otherwise, this breaks the security of the PRG used in the transformation). Then, if any efficient algorithm can distinguish between the small messages, it could also distinguish between the outputs of Sim for DPF. □

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

Proof. Assume towards contradiction that there exists an efficient algorithm $\mathcal{A}$ that makes non-zero output for DPF' that passes the audit without α. We can produce a non-zero message and its tag for DPF as follows. Run $\mathcal{A}$ to get a DPF key for each server: two DPF keys k_1, k_2 and a masked message m', along with tag $t = (t_1, t_2)$ for k_1 and k_2, respectively. If this passes the audit, the masked messages are the same (by the collision resistance of the audit hash function). Then, because the key for DPF' writes non-zero, at least one of the two DPF keys must write non-zero, and both must pass the audit. Pick the non-zero key (by evaluating both) and its corresponding tag. This key/tag pair passes the audit for DPF. □

6.2 Security of BlameGame

We must show the following properties for BlameGame: that (1) that an honest client will never be blamed, (2) a misbehaving client will always be blamed, (3) an honest server will never be blamed, and (4) a misbehaving server (during the audit phase) will always be blamed.

To see (1): if a server can blame an honest client, they must have produced a proof of decryption for a message that makes the audit fail (and which the client did not send); this violates the soundness of the verifiable encryption scheme.

To see (2): if a misbehaving client sends a plain text that will fail the audit, but BlameGame blames a server, then either the encryption verification failed (for an honest server, this violates the correctness of the verifiable encryption scheme) or the audit passed (which is a contradiction). Therefore, BlameGame must blame the client.

To see (3): if a server blames an honest server, then either the encryption verification failed (for an honest server, this violates the correctness of the verifiable encryption scheme) or the audit passed. However, the simulated audit passed during BlameGame uses the same shares as the honest server used during Spectrum but the two audits had different outcomes; because verification is deterministic, this is a contradiction.

To see (4): if a server misbehaves during the audit phase, but BlameGame blames an honest client, then the encryption verification must have passed and the BlameGame audit must fail. Because the client is honest, encryption verification must produce a value that makes the audit pass (otherwise, we violate the integrity of the verifiable encryption scheme).

If the server behaves honestly, but doesn't accumulate the clients's evaluated messages, then *all* messages in *all* channels are corrupted. This constitutes a failure in availability, but is no worse than the outcome if a server goes offline. If a malicious server sends a bad share of their messages to the other servers, they achieve the same result.

Overhead of BlameGame. BlameGame has a small amount of bandwidth and computation overhead. This includes extra bandwidth used to broadcast additional shares and time to verify decryption. 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 misbehaving client. However, verifying decryption takes tens of *microseconds*, and running the audit is similarly quick (see Section 7.2). Because the servers delay most of the work until after the audit, a misbehaving client often requires *fewer* cycles than an honest one (but slightly more network communication).

7 Evaluation

We build and evaluate Spectrum, comparing it to state-of-the-art anonymous broadcasting works. This section shows that Spectrum:

- In the best-case setting with only one client sharing a large message (Figures 4 and 7), outperforms existing work (by 4× or greater). This means 100,000 users upload 1 MB in 10 minutes, rather than days.
- In the "worst-case" setting where *all* clients broadcast small messages (Figure 5), achieves comparable performance to the next fastest work, Blinder. For 10 kB messages, Spectrum is 1.5–2× faster than the CPU variant of Blinder and 20× slower than the (much costlier to run) GPU variant. As message sizes get larger or fewer clients broadcast, Spectrum performs better than other systems.
- Increases throughput linearly by sharding (Figure 6), giving potential parallelization speedups of 10× or more (including in the above "worst-case" setting).
- Has very little overhead associated with BlameGame: per request, 140 B of extra network data and 10 µs to compute.

We compare Spectrum to Riposte, Blinder, Express, and Dissent (see related work; Section 8).

Riposte [22] 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 (where at least one is honest). Riposte was designed for smaller messages and the source code fails to run with messages of size 5 kB or greater.

Blinder [1] 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 [33] 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 [21, 84] Dissent 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 [1, Fig. 4] as the released source contained nontrivial compilation errors. 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 [84, Fig. 7].

7.1 Setup

Implementation. We build Spectrum in ~8000 lines of open-source [56] Rust code, using AES-128 (CTR) as a PRG and BLAKE3 [57] as a hash. Because our DPF has relatively few "channels" L, a DPF with $O(L)$-sized keys (adapted from Corrigan-Gibbs et al. [22]) gives the best concrete performance. For the multi-server extension (Section 5.1 and appendix A), we use a seed-homomorphic PRG [8] with the Jubjub [39] curve. We encrypt traffic with TLS 1.3 [63].

Environment. We run VMs on Amazon EC2 to simulate a WAN deployment. Each is a `c5.4xlarge`[3] 8-core instance with 32 GiB RAM, 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 RTTs were 11 ms between Virginia and Ohio, 50 ms between Ohio and California, and 61 ms between Virginia and

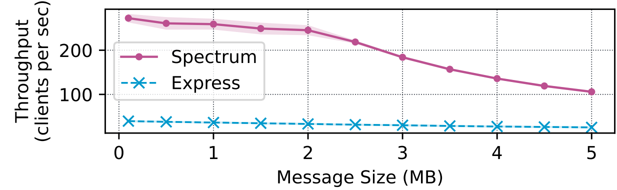


Figure 4: Throughput (client requests per second; higher better) for a one channel deployment (one broadcaster and many subscribers). Shaded region represents 95% confidence interval.

California. Inter-region bandwidth was 524 Mbit/s (shared between many clients simulated on the same machine).

7.2 Results

Across the below settings, we find Spectrum is 4–7× faster than Express, 2× faster than Blinder (CPU) and 13–17× slower than Blinder (GPU) *in settings favorable to Blinder*, 500–7500× faster than Blinder (CPU) and 250–520× faster than Blinder (GPU) *in settings favorable to Spectrum*, and 16–12,500× faster than Riposte. We run 5 trials per setting, shading the 95% confidence interval (occasionally invisible).

One channel. In Figure 4, we report the throughput (client requests per second) for both Spectrum and Express in the one-channel setting. As expected, throughput scales inversely with the message size for both Spectrum and Express. 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 messages of this size.)

Many channels. Riposte and Blinder's throughput depends only on the number of users. To compare, we fix 100,000 users and vary the number of channels for both Spectrum and Express from 1000 (best-case for Spectrum) to 100,000 (worst-case). Even with many simultaneous broadcast channels, Spectrum outperforms Express, Riposte, and Blinder (CPU) (see Figure 5). As the number of channels approaches 100,000, Spectrum's advantage shrinks, performing up to 20× worse than Blinder's GPU deployment (10× faster than the CPU variant). Therefore, in a setting where every user broadcasts to their own channel simultaneously, there is less benefit to using Spectrum (as expected). However, many real-world applications have a high number of passive subscribers per publisher [54, 85].

Timing breakdown. Table 2 shows the breakdown of server-side computation observed on one run. We find that the bulk of time is spent evaluating the PRG to expand the messages: for each client, we must expand to the length of a message,

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

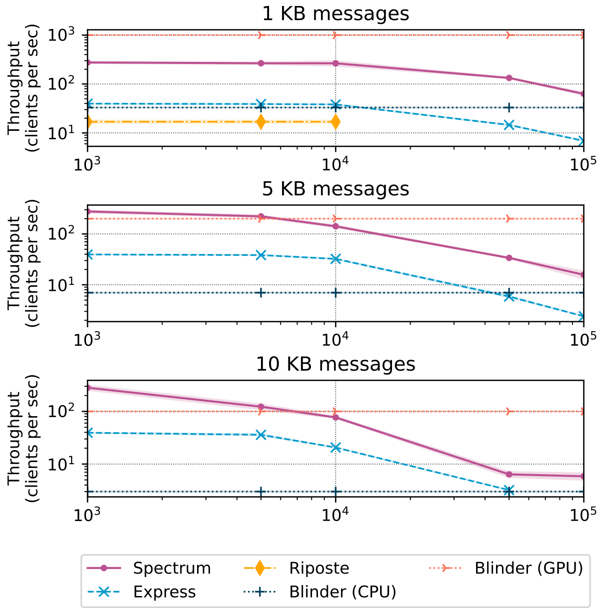


Figure 5: Throughput (requests per second; higher better) for broadcasts with 100,000 users: Express and Spectrum benefit from fewer channels. (Blinder numbers reported by the authors.) Shaded region represents 95% confidence interval.

which might be megabyte. "Idle" includes time waiting for audit messages from the other server; "Other" includes network processing and TLS decryption.

PRG	Hash	Combine	Idle	Other
69%	4%	2%	7%	25%

Table 2: Timing breakdown for server-side operations.

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 3. Client request overhead is small: about 70 B, roughly 75× smaller than in Express. Moreover, in Spectrum, request audits are under 100 B, a 120× improvement over Express [33]. BlameGame imposes little 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. Further, in many cases the BlameGame overhead is *lower* than the cost of the PRG evaluation: it is *cheaper to handle a malicious client* than an honest one.

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

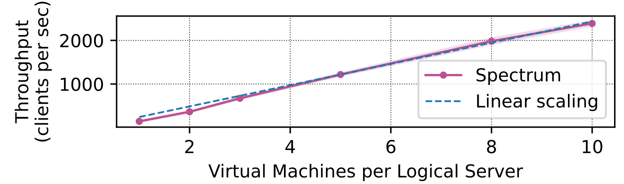


Figure 6: Spectrum is highly parallelizable: for 500 channels of 100 kB messages, 10 VMs per "server" gives a 10× speedup. Shaded region represents 95% confidence interval.

Many servers. In Section 5.1 and Appendix A, we note that our construction of Spectrum generalizes from 2 to n servers (with at least one non-colluding). As in Riposte [22], there is a corresponding performance hit, since we require a seed-homomorphic pseudorandom generator (PRG) [8] (see Section 5.1 and Appendix A). These PRGs are the bottleneck; we measure them on one core of an AMD Ryzen 4650G CPU. For our AES-128 PRG, maximum throughput is 5.7 GB/s; for a seed-homomorphic PRG [8], throughput is 300 kB/s (20,000× slower). For 64 channels of 160 B messages, Spectrum was 3× slower with the seed-homomorphic PRG. With more channels and larger messages, this number approaches 20,000×. We find *no additional* slowdown between 2 to 10 servers. Future work may find faster seed-homomorphic PRGs.

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 number of users. For Spectrum, we use a 1 MB message. For Blinder, we use numbers reported by the authors [1, 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). For Dissent, we benchmark both with no blame round (i.e., no client misbehaved), and with

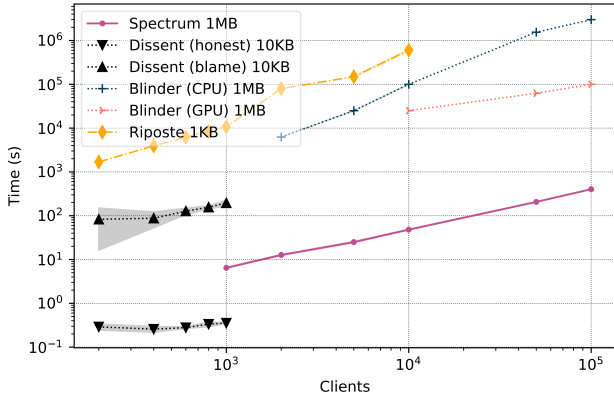
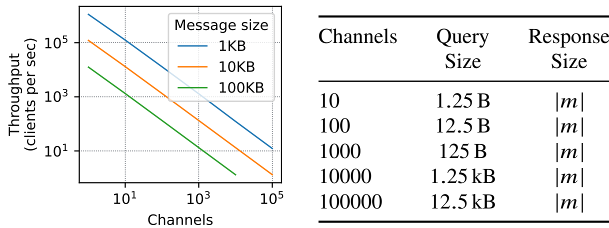


Figure 7: Latency for uploading a single document with varying numbers of users. Blinder numbers as reported by the authors [1, Fig. 4] and linearly scaled to 1 MB messages. Shaded regions represent 95% confidence interval.



Channels	Query Size	Response Size
10	1.25 B	$\lvert m\rvert$
100	12.5 B	$\lvert m\rvert$
1000	125 B	$\lvert m\rvert$
10000	1.25 kB	$\lvert m\rvert$
100000	12.5 kB	$\lvert m\rvert$

Figure 8: Server capacity (one core) to answer PIR queries for private client downloads, along with bandwidth usage. For L channels, the client requests one out of L documents, each of size $|m|$.

one blame round (if any client misbehaves). Express does not 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 for much smaller messages).

Client privacy. In Section 5.2, we outline how private information retrieval (PIR) [18] techniques provide client 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 [18], finding good concrete performance.

7.3 Discussion

Our evaluation 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 [70]) in 6h 40m.

Operational costs. We estimate costs for a cloud-based 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); inbound traffic is free on EC2. We exclude the cost of subscriber downloads, which is identical regardless of anonymous broadcast system: once published, the subscribers can download the content as normal. Most of our data transfer needs are inbound, from the clients to the servers, which is free on EC2. We focus on compute costs: $6.84 per GB published through Spectrum (with 10,000 users). In a cloud deployment, parallelizing yields better throughput at almost no additional cost. We compare costs to publish 1 GB among 10,000 users with other systems in Table 4.

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

Table 4: 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. *Extrapolated from 1000 users.

8 Related work

We provide a comparison to related work in anonymous broadcasting in Table 1. Existing systems for anonymous broadcast are suitable for 140 B to 40 kB [1, 22, 33] broadcasts, which is 2–8 orders of magnitude smaller than large data dumps [59, 65, 66] common today. Anonymity systems such as Tor [29] allow for greater throughput but fail to provide strong anonymity: if one whistleblower uploads large documents through Tor, metadata (visible to any Tor nodes) can be used to uniquely identify them. (Tor operators discourage high-bandwidth applications [28].)

Mix Networks and Onion Routing. In a mix net [16], 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 [25, 46, 47, 49, 50, 53, 55, 61, 62, 71, 72, 80] scale to many servers but are slow and use lots of extra bandwidth. Because the messages are re-sent

many times, mix nets are poorly suited to high-bandwidth applications (requiring long dummy messages from each user). Atom [45] uses mix nets with zero-knowledge proofs to horizontally scale anonymous broadcast to millions of users. (Spectrum achieves about 12,500× the throughput [45, Fig. 9].) Riffle [44] 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). To boost performance (by sacrificing strong anonymity), some systems use onion routing instead of a mix net.

In onion routing, users encrypt their messages several times (in onion-like layers) and send them to a chain of servers. Each server removes a layer of encryption and forwards the message onward. Unlike mix nets, onion routing is asynchronous and privacy relies on the difficulty of network monitoring. Tor [29], the most popular onion routing system, has millions of daily users [73]. Tor provides security in many real-world settings, but is vulnerable to traffic analysis [43, 52, 68]. State-of-art attacks [43, 52, 68] de-anonymize users over 90% accuracy; The high-bandwidth broadcast setting is particularly vulnerable: if only one user sends large volumes of data, an adversary can identify them—Tor discourages high bandwidth applications for this and other reasons [28].

DC-nets. Another group of anonymous communications systems use dining cryptographer networks (DC-nets) [17] (Section 2). In DC-nets, users broadcast anonymously by establishing random secrets between pairs of users in a many-way one-time pad; they can recombine "empty" shares with shares of a single broadcast share to recover the broadcast but hide the identity of the broadcaster. DC-nets are vulnerable to disruption: any malicious participant can clobber a broadcast by sending a "bad" share. Dissent [21, 84] 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 [5] 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 [22] 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, while Spectrum slows linearly.

A more recent work, Blinder [1] uses multi-party computation to prevent disruption. Blinder's threat model requires an honest majority of at least five servers. 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 [33] 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 Section 4.3 for details).

9 Conclusions

We present a new system for anonymous broadcast with strong anonymity and security guarantees. Spectrum supports high-bandwidth, low-latency transmission from a small set of broadcasters to a large set of subscribers by applying new tools to an old technique. Our main construction uses only symmetric-key primitives, which ensures efficiency in practical deployments. Our experimental results show that Spectrum can be used for uploading gigabyte-sized documents anonymously among 10,000 users in 14 hours.

10 Acknowledgments

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

References

[1] 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.

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

[3] 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.

[4] 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: 2020-05-01.

[5] 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.

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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 (Accessed 2020-10-27). URL https://www.nytimes.com/interactive/2020/09/27/us/donald-trump-taxes.html.

[12] Bryan Burrough, Sarah Ellison, and Suzanna Andrews. The Snowden saga: A shadowland of secrets and light. *Vanity Fair*, 2014 (Accessed: 2020-10-27). URL https://www.vanityfair.com/news/politics/2014/05/edward-snowden-politics-interview.

[13] 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.

[14] 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.

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

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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.

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

[25] 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.

[26] 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.

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

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

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

[30] 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/.

[31] 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.

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

[33] 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.

[34] Cassi Feldman. 60 Minutes' most famous whistleblower. *CBS News*, 2016 (Accessed 2020-10-27). URL https://www.theguardian.com/world/2010/nov/28/how-us-embassy-cables-leaked.

[35] Lorenzo Franceschi-Bicchierai. Snowden's favorite chat app is coming to your computer. *Vice*, 2015 (Accessed: 2020-10-27). URL https://www.vice.com/en/article/signal-snowdens-favorite-chat-app-is-coming-to-your-computer.

[36] Anita Gates and Katharine Q. Seelye. Linda Tripp, key figure in Clinton impeachment, dies. *The New York Times*, 2020 (Accessed 2020-10-27). URL https://www.nytimes.com/2020/04/08/us/politics/linda-tripp-dead.html.

[37] 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.

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

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

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

[41] 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.

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

[43] 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.

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

[45] 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.

[46] Albert Kwon, David Lu, and Srinivas Devadas. XRD: Scalable messaging system with cryptographic privacy. *arXiv preprint arXiv:1901.04368*, 2019.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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 (Accessed: 2020-10-27). URL https://www.buzzfeednews.com/article/jasonleopold/fincen-files-financial-scandal-criminal-networks.

[52] 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.

[53] 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.

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

[55] 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.

[56] Zachary Newman and Sacha Servan-Schreiber. Spectrum implementation. https://www.github.com/znewman01/spectrum-impl, 2021.

[57] Jack O'Connor, Samuel Neves, Jean-Philippe Aumasson, and Zooko Wilcox-O'Hearn. BLAKE3: One function, fast everywhere, 2020 (accessed: 2020-04-16). URL https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf.

[58] John O'Connor. "I'm the guy they called Deep Throat". *Vanity Fair*, 2006 (Accessed: 2020-10-27). URL https://www.vanityfair.com/news/politics/2005/07/deepthroat200507.

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

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

[61] 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.

[62] 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.

[63] 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.

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

[65] 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.

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

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

[68] 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.

[69] David Smith. Trump condemned for tweets pointing to name of Ukraine whistleblower. *The Guardian*, 2019 (Accessed 2020-10-27). URL https://www.theguardian.com/us-news/2019/dec/27/trump-ukraine-whistleblower-president.

[70] The BBC. Putin critic Navalny jailed in Russia despite protests, 2021 (accessed 2021-02-22). URL https://www.bbc.com/news/world-europe-55910974.

[71] The Freenet Project. Freenet, 2020. URL https://geti2p.net/en/.

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

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

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

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

[76] 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.

[77] US Holocaust Memorial Museum. Röhm purge. *Holocaust Encyclopedia*, 2020 (Accessed: 2020-10-27). URL https://encyclopedia.ushmm.org/content/en/article/roehm-purge.

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

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

[80] 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.

[81] 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.

[82] 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.

[83] 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.

[84] 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.

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

[86] 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: 2020-05-04.

A 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) [8].

Definition 2 (Seed-Homorphic 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 [7] 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. [8] 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.

B Verifiable Encryption

BlameGame (Section 4.3) uses a verifiable encryption scheme [14], 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 $\mathsf{negl}(\lambda)$, where the probability is over the randomness of Enc *and $\mathcal{A}$.*

We note that many public key encryption schemes (e.g., ElGamal [32] and Paillier [23]) satisfy Definition 3 out-of-the-box and can be used to instantiate BlameGame.