A High-Bandwidth, Low-Latency System for Anonymous Broadcasting

by Zachary James Newman
B.S., Columbia University (2014)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

May 2020

Author: ____

Department of Electrical Engineering and Computer Science
May 15, 2020

Certified by: ____

Srini Devadas
Edwin Sibley Webster Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: ____

Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

A High-Bandwidth, Low-Latency System for Anonymous Broadcasting

by Zachary James Newman

Submitted to the Department of Electrical Engineering and Computer Science on May 15, 2020, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Computer Science

abstract

In this thesis,[1] we design and evaluate Spectrum, a system for bandwidth efficient, anonymous broadcasting. In this system, one (or more) publishers broadcast a message to many users, who provide cover traffic. These users share their message between two or more servers; as long as one of these is honest, even an adversary with a complete view of the network cannot discover a message's source. Prior systems require that each user publish a message; however, streaming media tends to have many consumers for every content producer. We take advantage of this for a large performance improvement. To do so, we provide a cryptographic solution to the *disruption* problem, where dishonest users can corrupt a message by writing noise. This solution provides access control for operations performed obliviously by the servers. Spectrum is 60× faster than prior work for moderate-sized messages (10 kB), Each client uploads under 1 kB of additional data to broadcast a message of *any* size. Using a two-server deployment, Spectrum is fast enough to broadcast 3.4 GB/h to 600 users or streaming video to 5,000 users. Further, we can shard Spectrum across many physical servers for commensurate speedup, scaling a given workload to many more users.

Thesis Supervisor: Srini Devadas
Title: Edwin Sibley Webster Professor of Electrical Engineering and Computer Science

[1] This thesis contains research performed with Sacha Servan-Schreiber, currently unpublished [52]. We developed the protocol collaboratively; I contributed the majority of our implementation, by lines-of-code (84%) and conducted the experiments.

ACKNOWLEDGMENTS

This thesis would not exist if not for the unflagging support of my advisor Prof. Srini Devadas. Srini is always available to dispense wisdom about MIT, academia, computing, or eating more ice cream than I would have assumed possible.

The initial inspiration for this project came from Albert Kwon, without whom my adjustment to graduate school and MIT would have been much more difficult. Albert provided invaluable feedback on the project. My partner-in-crime on this project, Sacha Servan-Schreiber, has constantly pushed me to make it better. We may drive each other a little crazy, but I wouldn't have it any other way. Henry Corrigan-Gibbs was an irreplaceable mentor.

I owe a long list of people for motivating me throughout: My other lab-mates past and present—Alex, Alin, Hanshen, Ilia, Jules, Jun, Kyle, Quan—have been fonts of both knowledge and camaraderie throughout the past two years; I've really missed them while working from home. Profs. Tal Malkin and Allison Bishop inspired me to come to graduate school in the first place. My family—my parents, Kara, Audrey—have been a crucial source of stability since I've been conscious, and I credit them for my survival through high school and onward. My Boston friends, old and new—Maria, Margaret, Lowry, Melissa, Bill—and my friends elsewhere—Christian, Dina, Arta, Thomas, Noah, Appu, Estelle, Drew, Sam, Rob, Alicia, Makulumy, Katelyn, and so many more—have kept me laughing when I really needed it.

Finally, my partner Katie has been the picture of supportive: moving with me to Boston, making sure I don't forget to break for dinner when the workdays get long, and all without a hint of complaint! I know public acknowledgment would mortify her, so I'll keep the magnitude of my appreciation between me and anyone with an internet connection who can read this widely available document.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The voting booth is an apt symbol of democracy—not only because it enables voting, but also because it is a booth. Booths symbolize the need for privacy: the right to hold and express an unpopular viewpoint without fear of retribution, along with protections against blackmail and vote buying. Winston Churchill concurs, deeming "strict secrecy" of voting the "foundation of democracy" [33]. American jurisprudence increasingly accepts an implicit right to privacy in the US Constitution: Justice William O. Douglas writes for the majority in Griswold v. Connecticut that "specific guarantees in the Bill of Rights have penumbras, formed by emanations from those guarantees that help give them life and substance"—privacy the "penumbra" in question [65]. This case laid the basis for many future decisions upholding individual rights [66, 67]. Elizabeth Stoycheff, a scholar of journalism, argues [61] that government surveillance and monitoring programs specifically have a "chilling effect" that stymies minority viewpoints and intellectual freedom. Without private communications, government cannot be representative and just.

These protections do not apply in the converse—representative government must be transparent. The United States has a rich history of government whistleblowing exposing misconduct, often leading to substantive policy or administrative changes. These whistleblowers exposed: military and political deception surrounding the Vietnam War (Daniel Ellsberg, the *Pentagon Papers*) [6]; illegal, politically motivated wiretapping and the resulting cover-up, leading to President Richard Nixon's impeachment and resignation (Mark Felt, also known as "Deep Throat") [8]; sexual misconduct by President Bill Clinton, leading to his impeachment (Linda Tripp) [30]; FBI intelligence failures predating the terrorist attacks of September 11, 2001 (Coleen Rowley) [37]; war crimes in Afghanistan (Chelsea Manning) [13]; and mass surveillance by intelligence agencies (Edward Snowden) [12]. Without transparency to expose inefficiencies and injustices, even democratic government has at best limited accountability to its constituents.

Similarly, private whistleblowers expose misdoings by individuals and non-governmental organizations. The leaked *Panama Papers* detailed fraud, tax evasion, and sanction circumvention, leading to racketeering and bribery charges for FIFA officials and recovered sums in excess of $1 billion [23, 29, 64]; the *Paradise Papers* and many similar leaks describe the abuse of tax loopholes [34].

Whistleblowers are subject to harassment and punishment: many of the above have been prosecuted and in some cases served prison time. In the United States, prosecutors use broad laws against "aiding the enemy," and "treason," or the various provisions of the Espionage Act and Computer Fraud and Abuse Act, to punish leakers. Some legislation protects specific industries or organizations: dubiously-Constitutional "ag-gag" statutes [4] forbid documenting abuses by the agricultural industry. This motivates the frequent use of pseudonyms—"Deep Throat" from Watergate [54], "John Doe" from the *Panama Papers* [18]—in pursuit of anonymity.

In order to blow the whistle, however, one must somehow disseminate information and evidence. Historically, leakers met in-person with a journalist and give them physical papers; later leakers have mailed hard drives with data. Few use the internet directly, as it is a treacherous vehicle. Encryption technology hides the contents of messages, but not *metadata*: who communicates with whom, and when. Large-scale, extrajudicial US government programs vacuum up metadata from online communications [12] and use this metadata as a basis for military and political decision-making [19]. For instance, in the Chelsea Manning trial (where leaks *did* happen over the internet, the prosecution presented metadata evidence (of an SFTP connection) to link Manning to the WikiLeaks organization [73], resulting in conviction and a 35-year sentence.

Many systems for metadata-hiding online communication mitigate these risks for privacy-sensitive users. The most popular, Tor [26], obscures communication patterns with latencies of a few seconds—sufficient for web browsing. While Tor is vulnerable against adversaries with perfect views of the network, millions of daily users trust Tor with their traffic [63]. Other systems trade-off speed for security in this setting. Chapter 2 details many of these systems.

These systems excel where many users *each* want to publish small messages or communicate pairwise. However, the above-mentioned leaks range from a few gigabytes (the *Pentagon Papers* [1], "Cablegate" [13]) to 2.6 TB (the *Panama Papers* [29]). None of these systems support large-scale publishing of live video broadcasts or any large documents to many users. Terabyte-scale data upload is atypical network usage: an adversary watching Tor traffic could identify the rare users uploading this much data as the source of a data leak. Other systems are not vulnerable, but maximum practical transmission rates top out around 1 kbit/s [2, 44].

In this thesis, we present Spectrum, a system for *anonymous broadcast*. We obtain greater performance by focusing on the "few-to-many" setting: prior works assume that *all* users have data to publish, but Spectrum allows most users to be passive subscribers. These subscribers still provide cover traffic, but do not contribute to the system output. Because of the possibility of *disruption*—a malicious user can corrupt data—existing broadcast techniques do not support this setting. Here, we solve the disruption problem using privacy-preserving authentication techniques. We benchmark speeds sufficient for live video broadcast to up to 5,000 users using two servers, or rates of 8 Mbit/s with 600 users, sufficient to publish the *Pentagon Papers* in under two hours. Further, the system is designed for horizontal sharding, and can support more users by adding additional servers.

1.1 SETTING

In an anonymous broadcast, one or more *publishers* share *messages* via a set of intermediate servers while obscuring their source. As in other metadata-private systems, we define privacy among an *anonymity set*: the users who plausibly could have originated a given message. An adversary should not be able to learn which user originated each message with greater accuracy than by guessing at random within this set. In Spectrum, *subscribers* (passive users) provide cover traffic to increase the size of the anonymity set. These users might participate due to interest in the contents of the broadcast. Each broadcast corresponds to a "channel"—we can pool the broadcasts for several channels together, giving each a larger anonymity set.

Such a system should be correct, preserve client privacy, defend against disruption, and be efficient in practice. Users send messages to two or more servers; if one of these servers is honest (the *any-trust* model), no server can identify broadcasters. We elaborate on these goals in sections 3.1.1 and 3.2.1.

The primary challenge in anonymous broadcast is preventing *disruption* by malicious clients: in simple broadcast systems [16], users can overwrite other users' messages via undetectable deviations from the protocol. To work around this, other systems [2, 22] have users write messages to channels chosen from a large space, preventing collisions. In order to support a setting with relatively few publishers, we instead *authenticate* write operations for each channel. Before the broadcast, all publishers generate a key, which they can share with the servers using a lower-bandwidth anonymous broadcast system that requires no setup. Then, publishers must use this key in their broadcast, which the servers check in a private manner. This allows the servers to corrupt a channel, but they already had the ability to disrupt service by aborting the protocol; the addition of digital signatures can address message integrity concerns. Importantly, knowing the key *does not* give servers the ability to deanonymize publishers.

1.2 APPROACH

Spectrum uses *dining cryptographer* networks [16] as the basis for broadcast: users split their messages into "secret shares," which can be recombined to reveal the message. For simplicity, consider the case with one publisher, one subscriber, and two servers. To send a message m, the publisher samples a uniformly random bit string r of the same length, and send r and $r \oplus m$ (bit-wise exclusive-or) respectively to the servers. The subscriber samples uniformly random r' and sends r' to each server. The servers can recover the message as

$$\Big(\overbrace{r' \oplus r}^{\text{first server}}\Big) \oplus \Big(\overbrace{r' \oplus (r \oplus m)}^{\text{second server}}\Big) = (r' \oplus r') \oplus (r \oplus r) \oplus m = m.$$

However, each share on its own gives *no information* about the message, so the servers do not learn which user is the publisher.

The same technique generalizes to support many subscribers and many servers. For multiple publishers, users can repeat the protocol in parallel, acting as a publisher in one channel and a subscriber in others. However, this is inefficient: 100 channels would require each client to upload 100 cover messages, which is prohibitive when message sizes are large. We can do better by relaxing our privacy requirements. The above secret-sharing scheme is information-theoretically private, but for privacy against a computationally-bounded adversary, we can "compress" the randomness using a pseudorandom generator (PRG) and send PRG "seeds" instead. This is a *distributed point function* (DPF) [31]: a (possibly compact) encoding of a vector that is 0 at all points but one; at this point, it encodes a specific value.

Broadcast using DPFs is vulnerable to disruption: to corrupt the broadcast to a channel at a given point, a client can send an encoding of a random value at that point. Prior work [2, 22] using DPFs for broadcast has users write to one of a quadratically large number of channels, mitigating the odds of a collision. Instead, Spectrum uses private keys to authenticate a small number of these channels. We adapt Carter-Wegman [14, 69] style message authentication codes (MACs) to authenticate DPFs. These MACs allow message tagging based on finite field arithmetic. In the multi-party setting, parties can efficiently check these tags by performing arithmetic locally, then combining; the servers learn whether the tag was valid, but nothing more. Further, we can construct them with a "trapdoor," allowing null messages to be tagged without knowing the key. Only the publisher can tag a non-null message in a given channel, so rejecting any messages with invalid tags yields a secure protocol.

Per-client message processing can happen in parallel, so we can shard the protocol horizontally across many servers. Each user must upload data equal in size to the broadcast message, but the overhead beyond that is about twenty bytes per channel and one sixteen-byte message tag. In the two-server setting, we can use symmetric-key operations for the PRG expansion, which constitutes the bulk of the server-side operations. Expanding beyond two servers, these become public-key operations.

1.3 LIMITATIONS

Spectrum shares a few limitations with all protocols for metadata-private broadcast in the presence of passive network adversaries:

- All users (even those with no message to share) must use as much bandwidth as the publisher of the largest message.

  This is inherent to anonymous broadcast unless we relax the threat model. For instance, in Tor, any passive users would have relatively little network traffic—but a passive network adversary can identify the active Tor users.

- System throughput is limited by the upload speed of the slowest client for the same reason.

– The system is vulnerable to intersection attacks: if a particular data stream only appears when a particular user is online, they must be its source.

  If this protocol is repeated, all users must continue to participate, or they must be evicted from the anonymity set. As a result, active network attackers can reduce the anonymity set size by terminating connections.

In addition, our specific approach has further limitations:

– The techniques used in the two-server setting do not extend to the setting with three or more servers, so we use slower methods.

  Two-server constructions of distributed point functions can be built with symmetric-key primitives, which are much faster. Future work may improve the underlying seed-homomorphic pseudorandom generators (defined in section 3.4) or optimize aggregation in this setting.

– The setting requires pre-shared keys, presenting a logistical difficulty.

  However, without some system for authorization, we would lose our performance by over-provisioning the "channels" internal to the system. A different setting might allow authorization without pre-shared keys.

  Though these pre-shared keys are symmetric, the servers cannot use them to deanonymize the source; while this allows the servers to corrupt messages, they can do this already, by sending noise. In fact, the authorization techniques we use extend straightforwardly to the public-key setting; we present the private-key version for simplicity and because the public-key construction provides the same security in our model.

We hope that future work addresses these issues.

2 RELATED WORK

While cryptographers have devoted themselves to protecting communications privacy for millennia, their biggest success has been in protecting communication contents. Together, symmetric and asymmetric encryption are both mature technologies that protect the exchange of financial and medical data along with other sensitive communications. Pervasive state-run monitoring programs (long-suspected and recently confirmed) resort to the collection of *metadata*, suggesting the infeasibility of large-scale decryption. Protecting this metadata has proven trickier. There are essential difficulties: masking communications between two parties requires extraneous communications—a large cost—and attackers can extract patterns from an assortment of timing and presence. Systems for metadata-private communication aim to minimize these costs while providing message indistinguishability for a specific threat model, leaning on cryptographic tools to protect and validate data.

In this section, we survey relevant systems for anonymous communication and describe their suitability for large-scale anonymous communications. For anonymous broadcasting applications, we give a comparison to the performance of our work in chapter 4.

2.1 MIX NETWORKS AND ONION ROUTING

In a *mix network* [15], users send encrypted messages to a proxy server, which waits for a full "batch" of messages. Once this batch is complete, the proxy server shuffles, then forwards, these messages. Based on traffic patterns alone, even an adversary with full view of the network cannot link senders to receivers. By chaining several such servers (with encryption between hops), a mix network also protects users from compromised proxy servers, and provides a larger anonymity set.

Onion routing systems also use chains of proxy servers to hide the source of messages. To prepare a message for onion routing, users encrypt their messages in several layers (analogous to an onion) and send them to a chain of servers. Each of these removes a layer of encryption and forwards the message down the chain. Many systems combine this technique with mix networking, fully batching all messages. Others increase performance by using incomplete communication patterns. We observe the expected trade-off: for general-purpose routing, stricter mixing provides greater security at the expense of communication and computational overhead; the faster techniques are vulnerable against passive network adversaries. Consequently, relatively few practical systems use ideal, complete mix networks: most use some hybrid of mix and onion techniques.

Many systems built on these techniques aim to provide an anonymity layer over the internet. These systems vary deployment parameters, trading off performance and security, and use onion encryption to varying degrees and mix networks of varying size and density. Crowds [57] divides users into small "crowds," and traffic is forwarded peer-to-peer within the crowd before submission. This scales well, as path lengths remain relatively fixed. However, the system is vulnerable to Sybil attacks and traffic analysis. Mixminion [25] is a mix network for email (a *remailer*)—due to the longer latencies afforded by email communications, total delays can be on the order of hours even for small messages. The *Invisible Internet Project* (I2P) [62] combines mix networks and *garlic routing*, in which multiple messages encrypted together. A network adversary with a complete view has some chance of tracing traffic through the network. ShadowWalker [50] uses random walks over the network nodes to provide moderate levels of anonymity, but suffers the same limitation. Loopix [56] takes a principled approach to mixing asynchronous messages, injecting artificial delays modeling a Poisson process. This allows good throughput with better bounds on the possibility of deanonymization. Two systems achieve higher throughputs: Aqua [45] and Herd [46] use constant-rate internal flows supporting bursty traffic from edge nodes. These systems support high throughputs, but at the cost of a relaxed trust model: users select locally-clustered mix servers which they trust not to collude.

The widest-deployed of these systems is Tor [26], which anonymizes web traffic using onion routing. Tor provides security in many real-world settings, but is vulnerable to traffic analysis. State-of-the-art attacks [38, 47, 60] de-anonymize users with over 90% accuracy; further attacks are an area of active research [9, 11, 28, 35, 51, 55]. Despite these limitations, Tor attracts millions of daily users [63] due to its impressive performance: for web browsing, it incurs additional latencies of a few seconds.

Text chat systems Vuvuzela and Karaoke [43, 68] use asynchronous mixing in concert with *differential privacy* techniques to bound information leakage. Unfortunately, long-duration participation—necessary for higher bandwidth communications—can quickly consume the *privacy budget* assigned to each user. Further, these systems communicate point-to-point, and their privacy analyses depend on this: the broadcast setting requires nontrivial adaptations.

While many of these systems are suited for common web-browsing patterns or other activities, they promise probabilistic guarantees of privacy at best. Large-scale broadcast is a difficult test case, as publishing users have unusual traffic patterns. If only one user sends large volumes of data, an adversary can identify them by bandwidth usage alone. If we impose restrictions on communication patterns—users send messages in synchronous, fixed-size rounds—we turn probabilistic guarantees cryptographic. This approach is too expensive to cover general internet traffic, but if we focus on specific use cases, we see a reasonable combination of performance and security.

Pure mix net systems such as AsynchroMix [48] and MCMix [3] use multi-party computation protocols to achieve privacy and correctness. These are both very robust and secure against powerful adversaries: a small fraction of malicious servers cannot disrupt the messages. However, the performance is correspondingly poor: sending tiny messages to just thousands of users takes minutes. By sacrificing robustness, though not privacy, we can achieve better performance. To prevent malicious servers from identifying individuals, Riffle [39] introduces a *hybrid verifiable shuffle*. This is much lighter-weight than the general-purpose multi-party computations used in AsynchroMix and MCMix. However, performance is still limited: Riffle can broadcast a 300 MB file to 500 users in 3 hours. A refinement, Atom [40], combines mix networks with non-interactive zero-knowledge proofs in place of the verifiable shuffle to scale to millions of users and messages of size 32 bytes. Atom is advertised for micro-blogging, a low-throughput broadcast application. Quark [42] replaces many of the public-key cryptographic primitives in Atom with faster, hybrid cryptography. While its throughput is much higher, each message gets routed through many servers, causing high minimum latencies.

In a similar vein, XRD [41] is a parallel mix network-based system for peer-to-peer communication, incurring 4-minute latency when sending 256 B messages. The high-end of performance in this domain is Yodel [44], which allows point-to-point voice calls using hybrid encryption over mix networks (about 10 kbit/s). However, neither of these systems support one-to-many broadcast.

2.2 DINING CRYPTOGRAPHER NETWORKS

Dining cryptographer networks [16] are naturally well-suited to broadcast applications. These networks aggregate secret-shares from many users, obscuring their source. Users can provide shares of empty values, which allow the broadcast messages to be recovered. Systems based on dining cryptographer techniques admit straightforward proofs of security, avoiding the combinatorics and unrealistic assumptions of many systems in the mix network family.

The primary difficulty for these systems is dealing with *disruption*: if we allow anonymously sharing messages to the same slots, a malicious user can write to any slots. We see a variety of techniques for handling this problem. Herbivore [32] partitions the network into cliques to support general-purpose traffic; while the protocol is efficient, users enjoy anonymity only within their clique. At this scale, the users can set "traps" to identify malicious participants. Dissent [21] augments the dining cryptographers technique with a protocol for tracing, but not preventing, disruption. This protocol is still expensive, supporting groups of about 40 users (with latencies over an hour for 16 MB messages). Pung [5] uses private information retrieval techniques to allow private communication over fully-untrusted infrastructure (like Dissent). However, it relaxes the synchrony assumptions of Dissent by assuming batch transmission of messages. Relaxing to the *any-trust* model of security, Dissent in Numbers [70] can handle up to 5,000 participants. To handle still more users, Riposte [22] adds an additional server to audit that messages are well-formed (writing only to one slot), so over-provisioning slots bounds the chance of collision. However, performance is still insufficient: even with small messages (160 B), Riposte incurs 11 h latency. Express [27] uses similar techniques for point-to-point messaging, combined with an efficient auditing protocol to verify that messages are well-formed. The techniques in Express do not generalize to broadcast, as any user reading from a slot can also write to it. Blinder [2] makes two improvements over Riposte. First, Blinder uses Shamir-style secret sharing [59], which can be GPU-accelerated for an order-of-magnitude speed improvement. Second, it uses relatively lightweight multi-party computation techniques to reject disruptors, combined with the same over-provisioning of broadcast slots. This sets a record for larger messages: 10 kB uploads with ten thousand users with only 1.5 min latency. Like the above systems, Blinder processes a message from each user—its greatest computational bottleneck.

3 PROTOCOL

In this chapter, we define the Spectrum protocol. We begin with the protocol setting, interface, and security goals. We present a simple-but-insecure (and inefficient) variation of the protocol. Then, we enhance this variant for security and efficiency, introducing necessary primitives as needed. With these building blocks in place, we construct the final protocol in section 3.6.

3.1 SETTING

In the *anonymous broadcast* problem, one or more *publishers* broadcast fixed-length *messages* to a set of *subscribers*. The subscribers provide *cover traffic*, ensuring the publishers remain anonymous to a (computationally bounded) adversary monitoring the network. These users (subscribers and publishers) form the *anonymity set* for the protocol: the set of individuals which, from the perspective of the adversary, may have plausibly originated any message. When there is more than one publisher, we recover each message on its own *channel*. In Spectrum, users broadcast through two or more non-colluding servers: anonymity must hold provided at least one of these is honest (the *any-trust* model).

Since the number of channels is small, and these channels are world-readable, we must ensure that users cannot write to an arbitrary channel. We do this via per-channel *broadcast keys*, pre-shared between each publisher and the servers. The servers know the keys, but not their provenance; section 1.1 describes how this might be. While a malicious server can use this to forge a valid write message to a channel, this is no worse than before, as we require semi-honest servers to guarantee correct output (see the protocol goals, section 3.1.1). This limitation is common to anonymous broadcast systems: a malicious server can always perform a denial-of-service attack by aborting the protocol.

We assume a public-key infrastructure and that all network communications are encrypted and authenticated (for instance, using TLS [58]).

3.1.1 *Protocol Goals*

An anonymous broadcast protocol must achieve the following properties (we formalize these in section 3.2.1):

Correctness and availability If the servers execute the protocol correctly, they reveal *all* messages from all publishers at the conclusion. This property must hold even in the presence of malicious users.

Anonymity An adversary with a passive view of all network communications and controlling any proper subset of servers must not be able to identify the source of any message with probability much better than chance, given the set of publishers and subscribers.

Efficiency The protocol must be efficient, both asymptotically and concretely. Asymptotically, the total computation at each server must be linear in the total number of users. Each user must use bandwidth equal to the length of the *one* message, along with a term linear (or sublinear) in the number of publishers. Concretely, the protocol should support live broadcast of video (throughput of 1 Mbit/s with latency under 5 s) with an anonymity set of thousands of clients. Further, client request processing should be parallelizable: this allows scaling the protocol.

3.2 protocol interface

Let n be the number of servers and L be the number of channels. Let $\mathcal{K}$ be the space of per-channel broadcast keys, $\mathcal{M}$ be the space of possible messages, $\mathcal{R}$ be a "request space," and $\mathcal{B}$ be an "audit space." Let λ be a security parameter. For any set S, we use $S^+ = S^1 \cup S^2 \cup \dots$ (the Kleene plus) for the set of all sequences of one or more elements of S. Spectrum comprises the following (possibly randomized) algorithms:

- $\mathsf{Broadcast}(k \in \mathcal{K}, j \in [L], m \in \mathcal{M}) \to \mathcal{R}^n$. Generates a request share for each server encoding message m at channel j authorized by key k.
- $\mathsf{Cover}(\,) \to \mathcal{R}^n$. Generates a cover message share for each server, not encoding any message at any channel.
- $\mathsf{Audit}(\mathbf{k} \in \mathcal{K}^L, r \in \mathcal{R}) \to \mathcal{B}$. Generates an audit share for the request given the broadcast key vector.
- $\mathsf{Verify}(\boldsymbol{\beta} \in \mathcal{B}^n) \to \{0,1\}$. Verifies audit shares from each server. Outputs 1 if the audit shares are valid (attest to a well-formed message), or 0 otherwise.
- $\mathsf{Accept}(r \in \mathcal{R}) \to \mathcal{M}^L$. Maps a request to a vector of messages. The vector messages can be combined (see Combine) to recover messages for each channel.
- $\mathsf{Combine}(\mathbf{m} \in (\mathcal{M}^L)^+) \to \mathcal{M}^L$. Combines one or more vectors of messages into a single vector of messages.

The protocol proceeds as follows (see fig. 3-1):

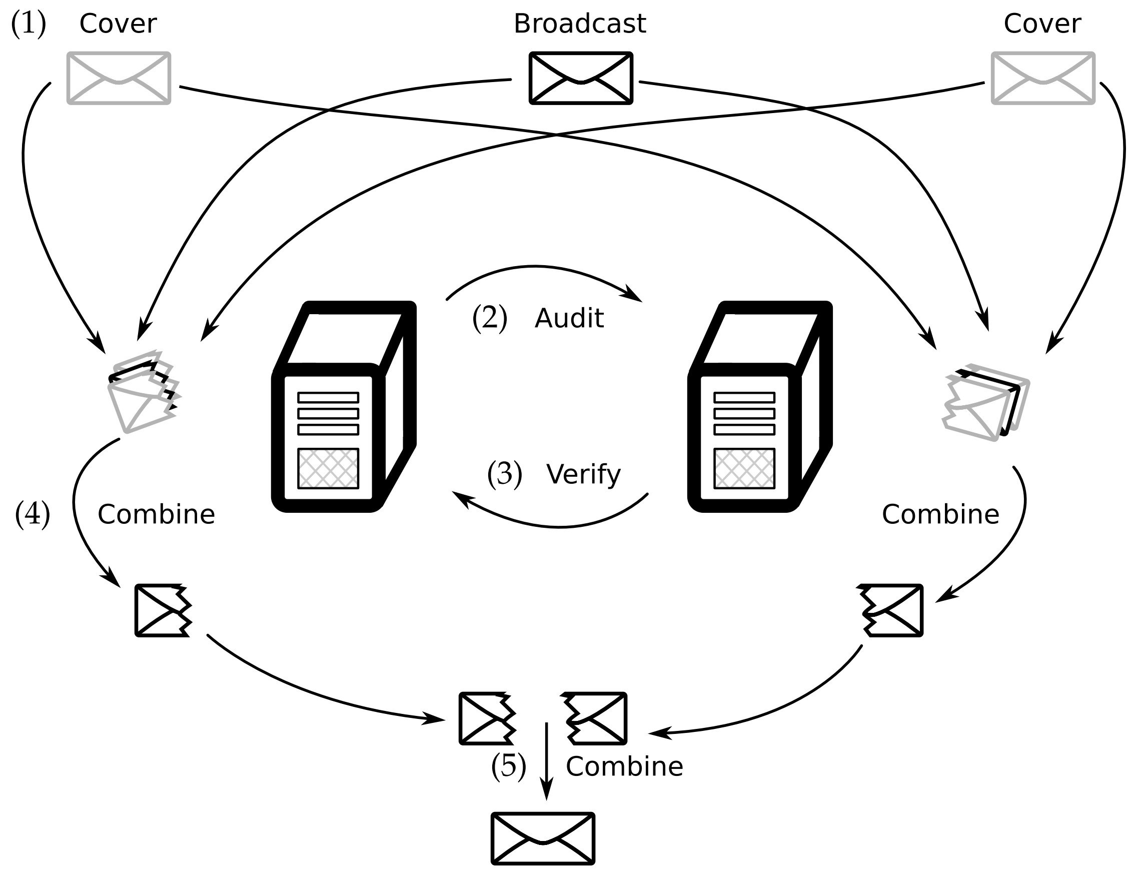

Figure 3-1: Overview of the Spectrum protocol: (1) Publishers send message shares; subscribers send cover traffic to each server. (2, 3) Together, servers check that messages are valid: authorized for broadcast or empty. (4) Each server aggregates data locally. (5) Servers aggregate data together, recovering the message.

1 Each client generates request shares for each server. Publishers use Broadcast (with the appropriate broadcast key), while subscribers use Cover. They send the n shares respectively to each of the n servers.

2 Each server runs Audit on every share received, distributing the n audit shares to each server.

3 On receipt of an audit share from each server for a given client's request, the server runs Verify. Verification fails only if the client (or some other server) cheats. If verification succeeds, the server maps the client's request to an accumulator using Accept.

4 After processing each client, the server combines the accumulators using Combine and publishes the resulting accumulator.

5 Once all such "server shares" are known, any party can combine these shares using Combine to recover the L messages.

3.2.1 *Security Goals*

Fix the protocol parameters n, L, $\mathcal{K}$, $\mathcal{M}$, $\mathcal{R}$, $\mathcal{B}$, and λ as above. Spectrum must satisfy the following properties:

Correctness of Broadcast and Cover Executed correctly, the protocol recovers all broadcast messages: for any sequence of L messages $\mathbf{m}$ in $\mathcal{M}^L$, sequence of L broadcast keys $\mathbf{k}$ in $\mathcal{K}^L$, non-negative number of subscribers C, and permutation π, we require that

$$\Pr\left[\begin{array}{l} \mathbf{r} \leftarrow \{\mathsf{Broadcast}(\mathbf{k}[j], j, \mathbf{m}[j]) \mid j \in [L]\}; \\ \mathbf{r}' \leftarrow \{\mathsf{Cover}(\,) \mid j' \in [C]\}; \\ \mathbf{m}' \leftarrow \{\mathsf{Accept}(r) \mid r \in \mathbf{r} \cup \mathbf{r}'\} \\ \text{s.t. } \mathsf{Combine}(\pi(\mathbf{m}')) = \mathbf{m} \end{array}\right] = 1.$$

Completeness of Broadcast and Cover The outputs of Broadcast and Cover must pass the audit: for any sequence of L broadcast keys $\mathbf{k}$ in $\mathcal{K}^L$, message m in $\mathcal{M}$, and index j in $[L]$, we require:

$$\Pr\left[\begin{array}{l} \mathbf{r} \leftarrow \mathsf{Broadcast}(\mathbf{k}[j], j, m) \\ \text{s.t. } \mathsf{Verify}\big(\{\mathsf{Audit}(\mathbf{k}, r) \mid r \in \mathbf{r}\}\big) = 1 \end{array}\right] = 1$$

and

$$\Pr\left[\begin{array}{l} \mathbf{r} \leftarrow \mathsf{Cover}() \\ \text{s.t. } \mathsf{Verify}\big(\{\mathsf{Audit}(\mathbf{k}, r) \mid r \in \mathbf{r}\}\big) = 1 \end{array}\right] = 1.$$

Soundness of Audit and Verify Without knowing the broadcast key for a channel, it should be computationally hard to create a set of requests that writes to that channel and passes the audit, even after many attempts: Let O be a verification oracle:

$$\mathsf{O}(\mathbf{k}, \mathbf{r}) = \mathsf{Verify}\big(\{\mathsf{Audit}(\mathbf{k}, r) \mid r \in \mathbf{r}\}\big).$$

Then, for all vectors of broadcast keys $\mathbf{k}$ in $\mathcal{K}^L$, channel indexes j in $[L]$, and probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$ with oracle access to O, we require:

$$\Pr\left[\begin{array}{l} \widehat{\mathbf{k}} \leftarrow \mathbf{k} \text{ with } \widehat{\mathbf{k}}[j] \xleftarrow{\$} \mathcal{K} \\ \mathbf{r} \leftarrow \mathcal{A}^{\mathsf{O}(\widehat{\mathbf{k}},\cdot)}(1^\lambda, \mathbf{k}j) \\ \text{s.t. } \mathsf{Verify}(\{\mathsf{Audit}(\mathbf{k}, r) \mid r \in \mathbf{r}\}) = 1 \\ \text{and } \mathsf{Combine}(\mathbf{r})[j] = m \end{array}\right] \leqslant \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\lambda)$.

Privacy It should be computationally hard to distinguish between the messages of a publisher and subscriber, even after seeing the final message and all audit shares: for any sequence of L broadcast keys $\mathbf{k} = (k_1, \ldots, k_L)$ in $\mathcal{K}^L$, set of messages $\mathbf{m}$ in $\mathcal{M}^L$, channel index j in $[L]$, and server index i in $[n]$, we require that the following distributions are computationally indistinguishable:

$$\big\{\{\mathbf{r}[i'] \mid i' \neq i\}, \boldsymbol{\beta}, \mathbf{k}\big\}_{\mathbf{r},\boldsymbol{\beta},\mathbf{k}} \approx_c \big\{\{\mathbf{r}'[i'] \mid i' \neq i\}, \boldsymbol{\beta}', \mathbf{k}\big\}_{\mathbf{r}',\boldsymbol{\beta}',\mathbf{k}}$$

where we sample the variables as

$$\begin{array}{ll} \mathbf{r} \leftarrow \mathsf{Broadcast}(\mathbf{k}[j], j, \mathbf{m}[j]), & \mathbf{r}' \leftarrow \mathsf{Cover}(), \\ \boldsymbol{\beta} \leftarrow \{\mathsf{Audit}(\mathbf{k}, r) \mid r \in \mathbf{r}\}, & \text{and } \boldsymbol{\beta}' \leftarrow \{\mathsf{Audit}(\mathbf{k}, r) \mid r \in \mathbf{r}'\}. \end{array}$$

3.2.2 *Trivial Protocol*

Given the above definitions, the following is a trivial, correct, and complete (but unsound and non-private) protocol:

$$\begin{array}{rlrl} \mathsf{Broadcast}(k, j, m) &= (0, \ldots, \underbrace{m}_{\text{index } j}, \ldots, 0) & \mathsf{Cover}() &= (0, \ldots, 0) \\ \mathsf{Audit}(\mathbf{k}, r) &= \perp & \mathsf{Verify}(\boldsymbol{\beta}) &= 1 \\ \mathsf{Accept}(r) &= r & \mathsf{Combine}(\mathbf{m}) &= \bigoplus_{m \in \mathbf{m}} m. \end{array}$$

This protocol shortcuts the auditing process, always returning 1, and encodes a message to be written at an index as a tuple of the message and index.

3.3 Anonymity via Dining Cryptographers

To achieve anonymity in our broadcast protocol, we use the "dining cryptographers" technique [16]. This technique broadcasts using *secret sharing*.

SECRET-SHARING [59] A secret-sharing scheme divides a value into "shares" such that any proper subset of shares reveals no information about the original value, but having *all* shares allows efficient recovery of the original value.

DEFINITION 1 (Secret-Sharing).
Consider set $\mathcal{S}$. An n-way secret-sharing *scheme consists of algorithms:*

- $\mathsf{Share}(m \in \mathcal{S}) \to \mathcal{S}^n$.
- $\mathsf{Recover}(\mathbf{m} \in \mathcal{S}^n) \to \mathcal{S}$.

satisfying the following properties:

CORRECTNESS *For all messages $m \in \mathcal{S}$,*

$$\Pr[\mathsf{Recover}(\mathsf{Share}(m)) = m] = 1.$$

SECRECY *For all proper subsets of indexes $I \subset [n]$ and all (possibly unbounded) adversaries $\mathcal{A}$, we have*

$$\Pr\begin{bmatrix} m \xleftarrow{\$} \mathcal{S}; \\ \mathbf{s} \leftarrow \mathsf{Share}(m) \\ \text{s.t. } \mathcal{A}\big(\{\mathbf{s}[i] \mid i \in I\}\big) = m \end{bmatrix} = \frac{1}{|\mathcal{S}|}.$$

We often use the notation $[\![x]\!]$ for a single secret share of some value x.

EXAMPLE: XOR SECRET SHARING Consider the following construction of an n-way secret-sharing scheme based on bit-wise exclusive-or of ℓ-bit binary strings. Then, let $\mathcal{S} = \{0,1\}^\ell$. Define:

$$\mathsf{Share}(m) = \left([\![m]\!]_1, \ldots, [\![m]\!]_{n-1}, m \oplus \bigoplus_{i \in [n-1]} [\![m]\!]_i\right),$$

sampling $[\![m]\!]_1, \ldots, [\![m]\!]_{n-1} \xleftarrow{\$} \{0,1\}^\ell$, and

$$\mathsf{Recover}([\![m]\!]_1, \ldots, [\![m]\!]_n) = \bigoplus_{i \in [n]} [\![m]\!]_i.$$

EXAMPLE: ADDITIVE SECRET SHARING Consider the following construction of an n-way secret-sharing scheme based on addition in some finite field $\mathbb{F}$: Let $\mathcal{S} = \mathbb{F}$. Then, define:

$$\mathsf{Share}(m) = \left([\![m]\!]_1, \ldots, [\![m]\!]_{n-1}, m - \sum_{i \in [n-1]} [\![m]\!]_i\right),$$

sampling $[\![m]\!]_1, \ldots, [\![m]\!]_{n-1} \xleftarrow{\$} \mathbb{F}$, and

$$\mathsf{Recover}([\![m]\!]_1, \ldots, [\![m]\!]_n) = \sum_{i \in [n]} [\![m]\!]_i.$$

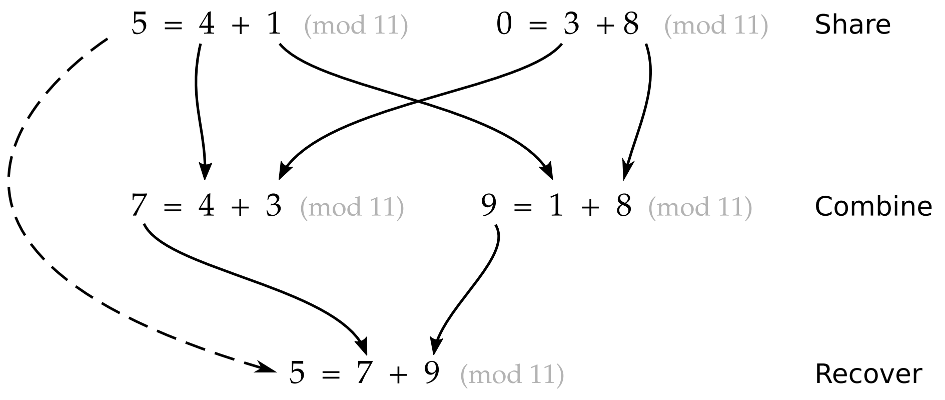

Figure 3-2: Dining cryptographer networks build anonymous broadcast from homomorphic secret sharing. In this case (working in $\mathbb{Z}_{11}$), we see how two distinct secret shares (one, a share of an empty value) can be combined before recovering the original message, 5.

Homomorphic secret sharing [7] Dining cryptographer networks require an additional property: the secret-sharing scheme must be *homomorphic*. This means that for some binary operator $\odot$ over $\mathcal{S}$ such that for any messages $m, m' \in \mathcal{S}$ we have:

$$\Pr\begin{bmatrix} \mathbf{s} \ \leftarrow \mathsf{Share}(m); \\ \mathbf{s}' \leftarrow \mathsf{Share}(m'); \\ \text{s.t. } \mathsf{Recover}\big(\{\mathbf{s}[i] \odot \mathbf{s}'[i] \mid i \in [n]\}\big) = m \odot m' \end{bmatrix} = 1.$$

When $\odot$ is $\oplus$ (bit-wise exclusive-or), we say the scheme is *xor-homomorphic*; when $\odot$ is $+$, we say the scheme is *additively-homomorphic*. The above secret-sharing schemes are respectively xor- and additively-homomorphic.

Dining cryptographer networks Fix some n-way secret-sharing scheme which is additively-homomorphic. We construct an anonymous broadcast scheme for one publisher mediated by n servers: the publisher secret-shares their message m; other parties secret-share a zero message. Then, the servers can combine their request shares using addition before adding these combined shares to recover m (fig. 3-2).

This generalizes to L publishers: every user does the protocol L times, where they share a non-zero value only for the channel on which they broadcast. In the language of our anonymous broadcast protocol (with $\mathcal{M} = \mathcal{R} = \mathcal{S}$, $\mathcal{S}$ being the domain of the secret-sharing scheme):

$$\begin{aligned} \mathsf{Broadcast}(k, j, m) &= (\mathsf{Share}(0), \ldots, \overbrace{\mathsf{Share}(m)}^{j\text{th position}}, \ldots, \mathsf{Share}(0)) \\ \mathsf{Cover}() &= (\mathsf{Share}(0), \ldots, \mathsf{Share}(0)) \\ \mathsf{Accept}(r) &= r \\ \mathsf{Combine}(\mathbf{m}) &= \sum_{m \in \mathbf{m}} m. \end{aligned}$$

This construction is correct, complete (we omit no-op Audit and Verify methods) and private, by reduction to secret-sharing privacy, but not efficient or sound.

3.4 EFFICIENCY VIA DISTRIBUTED POINT FUNCTIONS

The repeated dining cryptographer scheme is inefficient: it requires that each user send L elements of $\mathcal{S}$ to each server. Observe that these bits are mostly random: provided our adversary is computationally limited, we can "compress" these using a pseudorandom generator (PRG). The Riposte [22] scheme uses *distributed point functions* (DPFS) in this manner to reduce the bandwidth used in dining cryptographer-style broadcast; we use a similar technique. Since each user encodes at most one message, they can mask this message with pseudorandom seeds, rather than full-length random bits.

First, we define point functions and distributed point functions:

DEFINITION 2 (Point Function).
Fix some message space $\mathcal{M}$ with a zero element $0 \in \mathcal{M}$ and positive integer L. Then, for any message $m \in \mathcal{M}$ and index $j \in [L]$, define the point function $P_{j,m} : [L] \to \mathcal{M}$ *as:*

$$P_{j,m}(x) = \begin{cases} m & \text{if } x = j \\ 0 & \text{otherwise.} \end{cases}$$

Note that functions from $[L]$ to $\mathcal{M}$ are isomorphic to vectors in $\mathcal{M}^L$; we use these interchangeably. Further, note that where $\mathcal{M}$ is a field with some multiplicative identity element 1, we can write $\mathbf{e}_j \in \mathcal{M}^L$ for the jth row of the $L \times L$ identity matrix. Then, we have $P_{j,m} = \mathbf{e}_j \cdot m$. When clear from context, we may omit L and $\mathcal{M}$.

DEFINITION 3 (Distributed Point Function [31]).
Fix positive integers L and n and message space $\mathcal{M}$. An (L, n)-distributed point function is a family of functions $\{f_d : [L] \to \mathcal{M}\}$ indexed by keys $d \in \mathcal{D}$ along with algorithms:

– $\mathsf{DPF.Gen}(m \in \mathcal{M}, j \in [L]) \to \mathcal{D}^n$.

– $\mathsf{DPF.Eval}(d \in \mathcal{D}) \to \mathcal{M}$.

satisfying the following properties:

CORRECTNESS *For all point indexes $j \in [L]$ and all messages $m \in \mathcal{M}$, we have*

$$\Pr\left[\begin{array}{l} \mathbf{d} \leftarrow \mathsf{DPF.Gen}(m, j) \\ \text{s.t. } \sum_{d \in \mathbf{d}} (\mathsf{DPF.Eval}(d)) = \mathbf{e}_j \cdot m \end{array}\right] = 1.$$

SECRECY *For any proper subset of server indexes $I \subset [n]$, all messages $m \in \mathcal{M}$, and all channel indexes $j \in [L]$, there exists some simulator Sim_I such that the following distributions are computationally indistinguishable:*

$$\mathsf{Sim}_I() \approx_c \{\mathbf{d}[i] \mid i \in I\}_{\mathbf{d} \leftarrow \mathsf{DPF.Gen}(j,m)}.$$

In effect, a DPF is a secret-sharing of a point function. The dining cryptographer network repeated L times as above is a DPF with key size $L \cdot |\mathcal{S}|$ bits.

Using pseudorandom generators, we can reduce the key size substantially. For now, consider the two-server setting: $n = 2$. Let $G : \mathcal{S} \to \mathcal{M}$ be a pseudorandom generator from a small "seed" in $\mathcal{S}$ to a large "message" in $\mathcal{M}$. Then, we have:

$$\begin{aligned}
\mathsf{DPF.Gen}(m, j) = {} & \begin{matrix}\big((\widehat{m}, (x_1, \ldots, x_j, \ldots, x_L), (b_1, \ldots, \ \ b_j, \ldots, b_L)), \\ (\widehat{m}, (x_1, \ldots, x_j{}', \ldots, x_L), (b_1, \ldots, \neg b_j, \ldots, b_L))\big)\end{matrix} \\
& \text{where } x_1, \ldots, x_L \xleftarrow{\$} \mathcal{S}, \\
& \qquad x_j' \xleftarrow{\$} \mathcal{S}, \\
& \qquad b_1, \ldots, b_L \xleftarrow{\$} \{0, 1\}, \\
& \qquad \text{and } \widehat{m} \leftarrow m \oplus G(x_j) \oplus G(x_j')
\end{aligned}$$

$$\mathsf{DPF.Eval}(\widehat{m}, \mathbf{x}, \mathbf{b}) = \big((\mathbf{b}[1] \cdot \widehat{m}) \oplus G(\mathbf{x}[1]), \ \ldots, \ (\mathbf{b}[L] \cdot \widehat{m}) \oplus G(\mathbf{x}[L])\big).$$

Then, with bit-wise XOR for summation, the above is a DPF. In this construction, each DPF key consists of an encoded message, a bit vector, and a seed vector. The vectors are identical for $j' \neq j$, so the computed values cancel out:

$$\big((\widehat{m} \cdot b_{j'}) \oplus G(x_{j'})\big) \oplus \big((\widehat{m} \cdot b_{j'}) \oplus G(x_{j'})\big) = 0.$$

At $j' = j$, we recover the message:

$$\begin{aligned}
&\big((\widehat{m} \cdot b_j) \oplus G(s_j)\big) \oplus \big((\widehat{m} \cdot \neg b_j) \oplus G(x_j')\big) \\
&\quad = \widehat{m} \oplus G(s_j) \oplus G(s_j') \\
&\quad = \big(m \oplus G(x_j) \oplus G(x_j')\big) \oplus G(x_j) \oplus G(x_j') \\
&\quad = m.
\end{aligned}$$

$n > 2$ WITH SEED-HOMOMORPHIC PSEUDORANDOM GENERATORS The above technique requires the ability to compute two PRG seeds x_1 and x_2 that appear random when viewed independently but $G(x_1) \oplus G(x_2) = 0$: any $x_1 = x_2$ sampled uniformly at random suffices. To generalize to $n > 2$, we must find $\mathbf{x} = (x_1, \ldots, x_n)$ such that

$$\sum_{x \in \mathbf{x}} G(x) = 0.$$

To do this, we use seed-homomorphic PRGS:

DEFINITION 4 (Seed-Homomorphic Pseudorandom Generator [10]).
A seed-homomorphic pseudorandom generator *is a pseudorandom generator (PRG)* $G : \mathcal{S} \to \mathcal{M}$ *along with binary operators* $\otimes$ *on* $\mathcal{S}$ *and* $+$ *on* $\mathcal{M}$ *such that for all* $x_1, x_2 \in \mathcal{S}$ *we have*

$$G(x_1 \otimes x_2) = G(x_1) + G(x_2).$$

Let $\mathcal{M}$ be a field of prime order p with additive operator $+$ and generators $(g_1, \ldots, g_L)$, each in $\mathcal{M}$. Then, $G : \mathbb{Z}_p \to \mathcal{M}^L$ is a seed-homomorphic PRG (with $\otimes = +$):

$$G(x) = (g_1 \cdot x, \dots, g_L \cdot x)$$

where (with addition applied element-wise)

$$\begin{aligned} G(x_1 + x_2) &= (g_1 \cdot (x_1 + x_2), \dots, g_L \cdot (x_1 + x_2)) \\ &= (g_1 \cdot x_1, \dots, g_L \cdot x_1) \cdot (g_1 \cdot x_2, \dots, g_L \cdot x_2) \\ &= G(x_1) + G(x_2). \end{aligned}$$

Further, $G(0)$ is the additive identity in $\mathcal{M}^L$.

Using an additive secret-sharing scheme, we can now generalize our construction to (L, n)-DPFs:

$$\begin{aligned} \mathsf{DPF.Gen}(m, j) = &\left\{ (\widehat{m}, [\![\mathbf{x}]\!]_i, [\![\mathbf{b}]\!]_i) \mid i \in [n] \right\} \\ &\text{where } x \xleftarrow{\$} \mathbb{Z}_p, \qquad \widehat{m} \leftarrow m - G(x), \\ &\qquad \mathbf{x} \leftarrow \mathbf{e}_j \cdot x, \quad [\![\mathbf{x}]\!] \leftarrow \mathsf{Share}(\mathbf{x}), \\ &\qquad \mathbf{b} \leftarrow \mathbf{e}_j, \text{ and } \quad [\![\mathbf{b}]\!] \leftarrow \mathsf{Share}(\mathbf{b}) \\ \mathsf{DPF.Eval}\left((\widehat{m}, \mathbf{x}, \mathbf{b})\right) = &\{\widehat{m} \cdot \mathbf{b}[i] + G(\mathbf{x}[i]) \mid i \in [n]\}. \end{aligned}$$

Note that the above 2-server DPF is exactly this scheme, using XOR in place of both addition and subtraction. We have (using $G(\mathbf{x})$ for $(G(\mathbf{x}[1]), \dots, G(\mathbf{x}[L]))$ and with x, $\mathbf{x}$, $[\![\mathbf{x}]\!]$, $\mathbf{b}$, $[\![\mathbf{b}]\!]$, and $\widehat{m}$ as defined above):

$$\begin{aligned} \sum_{d \in \mathsf{DPF.Gen}(m,j)} \mathsf{DPF.Eval}(d) &= \sum_{i \in [n]} \left(\widehat{m} \cdot [\![\mathbf{b}]\!]_i + G\left([\![\mathbf{x}]\!]_i\right) \right) \\ &= \widehat{m} \cdot \sum_{i \in [n]} \left([\![\mathbf{b}]\!]_i\right) + \sum_{i \in [n]} G\left([\![\mathbf{x}]\!]_i\right) \\ &= \widehat{m} \cdot \mathbf{b} + G(\mathbf{x}) \\ &= \widehat{m} \cdot \mathbf{e}_j + G(\mathbf{e}_j \cdot x) \\ &= \mathbf{e}_j \cdot ((m - G(x)) + G(x)) \\ &= \mathbf{e}_j \cdot m. \end{aligned}$$

The above construction has key sizes *linear* in the number of points L, which in our setting is small and fixed (corresponding to the number of channels). Each key now has size $|m| + 2\lambda L$, rather than $|m| \cdot L$ as above.

DISTRIBUTED POINT FUNCTIONS FOR ANONYMOUS BROADCAST Ignoring verification, we have *most of* an efficient broadcast scheme for L channels and n servers with an (L, n)-DPF. To send a message $\mathbf{m}[j]$ on channel j, each publisher runs $\mathsf{DPF.Gen}$:

$$\mathbf{d}_j \leftarrow \mathsf{DPF.Gen}(\mathbf{m}[j], j) \quad \forall j \in [L].$$

Each broadcaster then distributes the DPF keys among the servers, which evaluate and combine them:

$$\overbrace{\sum_{i\in[n]} \underbrace{\sum_{j\in[L]} \mathsf{DPF.Eval}(\mathbf{d}_j[i])}_{\text{locally, at each server}}}^{\text{between servers}} = \sum_{j\in[L]} \sum_{i\in[n]} \mathsf{DPF.Eval}(\mathbf{d}_j[i]) = \sum_{j\in[L]} \mathbf{e}_j \cdot \mathbf{m}[j] = \mathbf{m}.$$

To generate cover traffic, we augment our DPF with another method, DPF.Empty, such that

$$\sum_{d\in\mathsf{DPF.Empty}()} \mathsf{DPF.Eval}(d) = \mathbf{0}.$$

The distribution of DPF.Empty must be indistinguishable from the distribution of the simulator defined for DPF privacy. For instance, in the above construction, we have:

$$\begin{aligned} \mathsf{DPF.Empty}() = \big\{(\widehat{m}, [\![\mathbf{0}]\!]_i, [\![\mathbf{0}]\!]'_i)\big\}_{i\in[n]} \\ \text{where } \widehat{m} &\xleftarrow{\$} \mathcal{M}, \\ [\![\mathbf{0}]\!] &\leftarrow \mathsf{Share}(\mathbf{0}), \\ \text{and } [\![\mathbf{0}]\!]' &\leftarrow \mathsf{Share}(\mathbf{0}). \end{aligned}$$

Then, we can define Cover = DPF.Empty.

3.5 SOUNDNESS VIA MESSAGE AUTHENTICATION CODES

Instantiating anonymous broadcast with the above schemes neglects soundness; consequently, a malicious user can send a malformed message corrupting the publishers' messages. To prevent this, we have the users attach to their request a certificate that the request is well-formed. Namely, this certificate attests for each channel that *either* the request writes 0 to that channel, or the user knows a secret key for the channel (see section 3.1). The servers must verify this certificate *without* learning which condition the request satisfies.

To construct these certificates, we extend the Carter-Wegman message authentication code (MAC) [14, 69] and give a technique for verifying these codes over secret-shared data.

CARTER-WEGMAN MESSAGE AUTHENTICATION CODE Working in a finite field $\mathbb{F}$, we present a simple MAC for messages in $\mathbb{F}$ in the Carter-Wegman style [14, 69]; the SPDZ multiparty computation protocol [24] employs a similar technique. Assume that two parties share a secret key $k \xleftarrow{\$} \mathbb{F}$. We compute the tag for a message m as

$$\mathsf{MAC}_k(m) = k \cdot m.$$

We can then authenticate a message-tag pair (m, t) by independently computing the same key. Note that for any key k, $\mathsf{MAC}_k(0) = 0$. Without knowledge of k, forging this tag is information-theoretically hard for any non-zero message.

Authentication of secret-shared data Note that the above scheme is *linear*: for any message m_1, m_2 and any key k, we have $\mathsf{MAC}_k(m_1+m_2) = \mathsf{MAC}_k(m_1)+\mathsf{MAC}_k(m_2)$. Several parties can validate the putative tag t for a message m given only additive secret shares $[\![t]\!]_i$ and $[\![m]\!]_i$, computing:

$$\overbrace{\sum_{i\in[n]} (\mathsf{MAC}_k([\![m]\!]_i) - [\![t]\!]_i)}^{\text{sum across servers}} = \overbrace{\sum_{i\in[n]} \mathsf{MAC}_k([\![m]\!]_i)}^{\text{expected tag}} - \overbrace{\sum_{i\in[n]} [\![t]\!]_i}^{\text{provided tag}} = \mathsf{MAC}_k(m) - t.$$

If the provided tag is valid, this value should be 0.

3.5.1 *Authenticating Distributed Point Functions*

Using the techniques from the above MAC, we can augment our (L, n)-DPF with per-point keys authenticating DPF keys at a given index. Let $\mathcal{T}$ (the "tag space"), and $\mathcal{B}$ (the "audit space") be sets, with key space $\mathcal{K}$ as above. Then, consider the algorithms:

– $\mathsf{DPF.Tag}(\mathbf{d} \in \mathcal{D}^n, j \in [L], k \in \mathcal{K}) \to \mathcal{T}^n$. Tags a message using a key.

– $\mathsf{DPF.TagEmpty}(\mathbf{d} \in \mathcal{D}^n) \to \mathcal{T}^n$. Tags an empty message, *with no key*.

– $\mathsf{DPF.Audit}(\mathbf{k} \in \mathcal{K}^L, d \in \mathcal{D}, t \in \mathcal{T}) \to \mathcal{B}$. Create an *audit token*, which can be used to validate a message/tag pair.

– $\mathsf{DPF.Verify}(\boldsymbol{\beta} \in \mathcal{B}^n) \to \{0, 1\}$. Verify that the audit tokens from each server correspond to a valid message/tag pair.

with the following properties:

Completeness For all messages $m \in \mathcal{M}$, point indexes $j \in [L]$, and point key vectors $\mathbf{k} \in \mathcal{K}^L$, we have:

$$\Pr\left[\begin{array}{l}\mathbf{d} \leftarrow \mathsf{DPF.Gen}(j, m)\\ \mathbf{t} \leftarrow \mathsf{DPF.Tag}(\mathbf{d}, j, \mathbf{k}[j])\\ \text{s.t. } \mathsf{DPF.Verify}\big(\{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i], \mathbf{t}[i]) \mid i \in [n]\}\big) = 1\end{array}\right] = 1.$$

Additionally, for all point key vectors $\mathbf{k} \in \mathcal{K}^L$, we have:

$$\Pr\left[\begin{array}{l}\mathbf{d} \leftarrow \mathsf{DPF.Empty}()\\ \mathbf{t} \leftarrow \mathsf{DPF.TagEmpty}(\mathbf{d})\\ \text{s.t. } \mathsf{DPF.Verify}\big(\{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i], \mathbf{t}[i]) \mid i \in [n]\}\big) = 1\end{array}\right] = 1.$$

Soundness Let O be the oracle function

$$\mathsf{O}(\mathbf{k}, \mathbf{d}, \mathbf{t}) = \mathsf{DPF.Verify}\big(\{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i], \mathbf{t}[i]) \mid i \in [n]\}\big).$$

For all PPT adversaries $\mathcal{A}$ with oracle access to O, point key vectors $\mathbf{k}$ in $\mathcal{K}^L$, and channel indexes $j \in [L]$ we have:

$$\Pr\left[\begin{array}{l} \quad\widehat{\mathbf{k}} \quad \leftarrow \mathbf{k} \text{ with } \widehat{\mathbf{k}}[j] \xleftarrow{\$} \mathcal{K}; \\ (\mathbf{d}, \mathbf{t}) \leftarrow \mathcal{A}^{\mathsf{O}(\widehat{\mathbf{k}},\cdot,\cdot)}(1^\lambda, \mathbf{k}, j); \\ \quad\boldsymbol{\beta} \quad \leftarrow \{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i], \mathbf{t}[i]) \mid i \in [n]\} \\ \text{s.t.} \quad \mathsf{DPF.Verify}(\boldsymbol{\beta}) = 1 \\ \text{and} \ \displaystyle\sum_{d \in \mathbf{d}} (\mathsf{DPF.Eval}(d))[j] \neq 0 \end{array}\right] \leqslant \mathsf{negl}(|\mathcal{K}|)$$

for some negligible function $\mathsf{negl}(\cdot)$.

Privacy For any proper subset of indexes $I \subset [n]$, all messages $m \in \mathcal{M}$, all point key vectors $\mathbf{k} \in \mathcal{K}^L$, and all indexes $j \in [L]$, there exists some simulator Sim_I such that the following distributions are pairwise computationally indistinguishable:

$$\begin{aligned} \mathsf{Sim}_I() &\approx_c \{(\mathbf{d}[i], \mathbf{t}[i], \boldsymbol{\beta}) \mid i \in I\}_{\substack{\mathbf{d} \leftarrow \mathsf{DPF.Gen}(j,m) \\ \mathbf{t} \leftarrow \mathsf{DPF.Tag}(\mathbf{d},j,\mathbf{k}[j]) \\ \boldsymbol{\beta} \leftarrow \{\mathsf{DPF.Audit}(\mathbf{k},\mathbf{d}[i'],\mathbf{t}[i']) \mid i' \in [n]\}}} \\ &\approx_c \{(\mathbf{d}[i], \mathbf{t}[i], \boldsymbol{\beta}) \mid i \in I\}_{\substack{\mathbf{d} \leftarrow \mathsf{DPF.Empty}() \\ \mathbf{t} \leftarrow \mathsf{DPF.TagEmpty}(\mathbf{d}) \\ \boldsymbol{\beta} \leftarrow \{\mathsf{DPF.Audit}(\mathbf{k},\mathbf{d}[i'],\mathbf{t}[i']) \mid i' \in [n]\}.}} \end{aligned}$$

That is, $\mathsf{DPF.Tag}$ with a correct key produces tags which pass an honestly-evaluated audit at each server, but without knowing this key producing such tags is hard. DPF privacy should hold even after seeing the tags and audit shares.

Construction We now construct a DPF authentication scheme for the above seed-homomorphic PRG-based DPF. Let H be a collision-resistant hash function (CRHF) sampled from a CRHF family $\mathcal{H}$. Fix a PRG $G : \mathcal{K} \to \mathcal{M}$ (with $\mathcal{K}$ a finite field) and let the key space for DPF authentication be $\mathcal{K}$. Let $(\mathsf{Share}, \mathsf{Recover})$ be the additive secret-sharing scheme from section 3.3 over $\mathcal{K}^2$. Let $\langle\cdot,\cdot\rangle$ denote the dot product of two vectors, and define multiplication in $\mathcal{K} \times \mathcal{K}^2$ as element-wise multiplication of the latter two elements by the first element. Then, we have:

$$\begin{aligned}
\mathsf{DPF.Tag}(\mathbf{d}, j, k) &= \mathsf{Share}\big((k \cdot x, k)\big) \\
&\quad \text{where } \big((\widehat{m}, [\![\mathbf{x}]\!]_1, [\![\mathbf{b}]\!]_1), \dots\big) = \mathbf{d}, \text{ and} \\
&\qquad\qquad x = \sum_{i \in [n]} [\![\mathbf{x}]\!]_i[j] \\
\mathsf{DPF.TagEmpty}(\mathbf{d}) &= \mathsf{Share}\big((0, 0)\big) \\
\mathsf{DPF.Audit}(\mathbf{k}, d, t) &= (H(\widehat{m}), (\langle \mathbf{k}, \mathbf{x} \rangle, \langle \mathbf{k}, \mathbf{b} \rangle) - t) \\
&\quad \text{where } (\widehat{m}, \mathbf{x}, \mathbf{b}) = d \\
\mathsf{DPF.Verify}(\boldsymbol{\beta}) &= \begin{cases} 1 & \text{if all } \{h_i\}_{i \in [n]} \text{ equal and } \sum_{i \in [n]} c_i = (0, 0) \\ 0 & \text{otherwise} \end{cases} \\
&\quad \text{where } \big((h_1, c_1), \dots\big) = \boldsymbol{\beta}.
\end{aligned}$$

Here, we use the MAC technique described above to verify that for each DPF index, the (secret-shared) tag is valid, where zero-messages have the zero-tag. There are two changes: first, we compare hashes of the encoded message to prevent equivocation. Second, we take the dot product of the key vector with the seed and bit vectors and check the *sum* of the expected tags. To compute this requires the correct tag for *each* non-zero index.

We can show that this construction is *complete*, *sound*, and *private* when instantiated with $\mathbb{Z}_p$ with λ-bit prime p.

Claim 1. *The construction is complete.*

Proof. First, note that both DPF.Tag and DPF.TagEmpty insert the same encoded message in each share, ensuring that the message hashes will always be equal. Fix any message $m \in \mathcal{M}$, point index $j \in [L]$, point key vector $\mathbf{k} \in \mathcal{K}^L$. Then, we have (for the completeness of DPF.Tag):

$$\begin{aligned}
&\Pr\left[\begin{array}{l} \mathbf{d} \leftarrow \mathsf{DPF.Gen}(j, m) \\ \mathbf{t} \leftarrow \mathsf{DPF.Tag}(\mathbf{d}, j, \mathbf{k}[j]) \\ \text{s.t. } \mathsf{DPF.Verify}\Big(\{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i], \mathbf{t}[i])\}_{i \in [n]}\Big) = 1 \end{array}\right] && \text{(def. completeness)} \\
= &\Pr\left[\begin{array}{l} x \xleftarrow{\$} \mathbb{Z}_p \\ \widehat{m} \leftarrow m - G(x) \\ \mathbf{t} \leftarrow \mathsf{Share}\big((\mathbf{k}[j] \cdot x, \mathbf{k}[j])\big) \\ [\![\mathbf{x}]\!] \leftarrow \mathsf{Share}(\mathbf{e}_j \cdot x) \\ [\![\mathbf{b}]\!] \leftarrow \mathsf{Share}(\mathbf{e}_j) \\ \text{s.t. } \sum_{i \in [n]} \big((\langle \mathbf{k}, [\![\mathbf{x}]\!]_i \rangle, \langle \mathbf{k}, [\![\mathbf{b}]\!]_i \rangle) - \mathbf{t}[i]\big) = (0, 0) \end{array}\right] && \text{(def. DPF algorithms)} \\
= &\ 1.
\end{aligned}$$

The final equality holds because

$$\begin{aligned}
&\sum_{i\in[n]}\big((\langle \mathbf{k}, [\![\mathbf{x}]\!]_i\rangle, \langle \mathbf{k}, [\![\mathbf{b}]\!]_i\rangle) - \mathbf{t}[i]\big)\\
&= (\langle \mathbf{k}, \mathbf{e}_j \cdot x\rangle, \langle \mathbf{k}, \mathbf{e}_j\rangle) - (\mathbf{k}[j]\cdot x, \mathbf{k}[j])\\
&= (\mathbf{k}[j]\cdot x, \mathbf{k}[j]) - (\mathbf{k}[j]\cdot x, \mathbf{k}[j])\\
&= (0,0).
\end{aligned}$$

We now consider the completeness of DPF.TagEmpty. Fix any point key vector $\mathbf{k} \in \mathcal{K}^L$. Then,

$$\begin{aligned}
&\Pr\left[\begin{array}{l}\mathbf{d} \leftarrow \mathsf{DPF.Empty}()\\ \mathbf{t} \leftarrow \mathsf{DPF.TagEmpty}(\mathbf{d})\\ \text{s.t. } \mathsf{DPF.Verify}\Big(\{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i], \mathbf{t}[i])\}_{i\in[n]}\Big) = 1\end{array}\right] && \text{(def. completeness)}\\
=\ &\Pr\left[\begin{array}{l}[\![\mathbf{x}]\!] \leftarrow \mathsf{Share}(\mathbf{0})\\ [\![\mathbf{b}]\!] \leftarrow \mathsf{Share}(\mathbf{0})\\ \mathbf{t} \quad \leftarrow \mathsf{Share}\big((0,0)\big)\\ \text{s.t. } \sum_{i\in[n]}\big((\langle \mathbf{k}, [\![\mathbf{x}]\!]_i\rangle, \langle \mathbf{k}, [\![\mathbf{b}]\!]_i\rangle) - \mathbf{t}[i]\big) = (0,0)\end{array}\right] && \text{(def. DPF algorithms)}\\
=\ &\Pr[(0,0) - (0,0) = (0,0)]\\
=\ &1
\end{aligned}$$

as required. □

Claim 2. *The construction is sound.*

Proof. To show the soundness of DPF.Audit and DPF.Verify, we first consider the following *guessing game*: let O' be the *equality oracle*

$$\mathsf{O}'(k, k') = \begin{cases} 1 & k = k' \\ 0 & \text{otherwise} \end{cases}$$

and let $\mathcal{B}$ be any PPT adversary with access to such an oracle. Then, the guessing game is:

$$\begin{array}{l} k \xleftarrow{\$} \mathcal{K}\\ k' \leftarrow \mathcal{B}^{\mathsf{O}'(k,\cdot)}(1^\lambda)\\ \mathcal{B} \text{ succeeds if } k = k'. \end{array}$$

Define the truncated equality oracle

$$\mathsf{O}'^{(z)}(k, k') = \begin{cases} \mathsf{O}'(k, k') & \text{for the first } z \text{ queries} \\ 0 & \text{subsequently.} \end{cases}$$

Let q be the maximum number of queries issued by $\mathcal{B}$. Consider the following sequence of games:

GAME 0 The guessing game, as above.

GAMES 1 TO q Replace O' with $O'^{(q')}$, where q' is the index of the current game.

GAME $(q+1)$ Replace O' with an oracle which always returns 0.

The probability that $\mathcal{B}$ succeeds in GAME $(q+1)$ is at most $(1/|\mathcal{K}|)$, as k' is sampled independently of k. For every adjacent pair of games q' and $q'+1$, the outcome differs only if $\mathcal{B}$ guesses k for the $(q'+1)$th query, which happens with probability at most $(1/|\mathcal{K}|)$. Since $\mathcal{B}$ runs in PPT, q is polynomial in λ and we have:

$$\Pr\left[\mathcal{B} \text{ succeeds in GAME } 0\right] \leqslant \frac{\mathsf{poly}(\lambda)+1}{|\mathcal{K}|}.$$

When $|\mathcal{K}|$ is exponential in λ, this expression is negligible in λ.

Now, suppose towards contradiction that there exists some key vector $\mathbf{k} \in \mathcal{K}^L$, channel index $j \in [L]$, and PPT adversary $\mathcal{A}$ such that $\mathcal{A}$ succeeds in the soundness game for DPF.Audit and DPF.Verify with non-negligible probability $\epsilon(\lambda)$. We use $\mathcal{A}$ to build an adversary $\mathcal{B}$ for the guessing game. First, define a subroutine ComputeKey to compute a key from a vector of DPF keys and corresponding MAC tags:

```
ComputeKey($\mathbf{d}, \mathbf{t}$)
parse each $\mathbf{d}[j]$ as $(\hat{m}_j, [\![\mathbf{x}]\!]_j, [\![\mathbf{b}]\!]_j)$
$\mathbf{b} \leftarrow \mathsf{Recover}([\![\mathbf{b}]\!]_j \mid j \in [L])$
$\mathbf{x} \leftarrow \mathsf{Recover}([\![\mathbf{x}]\!]_j \mid j \in [L])$
$(t_1, t_2) \leftarrow \mathsf{Recover}(t \mid t \in \mathbf{t})$
$g \leftarrow (t_1 - \langle \mathbf{k}, \mathbf{x} \rangle + \mathbf{k}[j] \cdot \mathbf{x}[j])$
$g' \leftarrow (t_2 - \langle \mathbf{k}, \mathbf{b} \rangle + \mathbf{k}[j] \cdot \mathbf{b}[j])$
if $\mathbf{x}[j] = 0$ and $\mathbf{b}[j] = 0$
    return 0
else if $\mathbf{x}[j] \neq 0$ and $\mathbf{b}[j] \neq 0$
    $k \leftarrow g \cdot \mathbf{x}[j]^{-1}$
    $k' \leftarrow g' \cdot \mathbf{b}[j]^{-1}$
    if $k \neq k'$
        return $\perp$
    return $k$
else if $\mathbf{x}[j] \neq 0$
    return $g \cdot \mathbf{x}[j]^{-1}$
else if $\mathbf{b}[j] \neq 0$
    return $g' \cdot \mathbf{b}[j]^{-1}$
```

Then, we can define $\mathcal{B}$:

$\mathcal{B}^{O'(k,\cdot)}(1^\lambda)$

```
1  $(\mathbf{d}, \mathbf{t}) \leftarrow \mathcal{A}(1^\lambda, \mathbf{k}, j)$
2     answer query $O(\mathbf{k}, \mathbf{d}, \mathbf{t})$ with:
3       parse each $\mathbf{d}[j]$ as $(\widehat{m}_j, [\![\mathbf{x}]\!]_j, [\![\mathbf{b}]\!]_j)$
4       **if** any two message hashes $H(\widehat{m}_j)$ unequal
5         **return** $\perp$
6       $k' \leftarrow \mathsf{ComputeKey}(\mathbf{d}, \mathbf{t})$
7       **if** $k' = 0$
8         **return** $O(\mathbf{k}, \mathbf{d}, \mathbf{t})$
9       **return** $k' \neq \perp$ and $O'(k, k')$
10 **return** $\mathsf{ComputeKey}(\mathbf{d}, \mathbf{t})$
```

In line 2, we answer oracle queries with exactly the same distribution as in the soundness game: let $\widehat{\mathbf{k}}$ be $\mathbf{k}$ with the jth element replaced by the unknown k from the guessing game. Recall that the soundness oracle O checks that all message hashes are equal and

$$\left(\left\langle \widehat{\mathbf{k}}, \mathbf{x} \right\rangle, \left\langle \widehat{\mathbf{k}}, \mathbf{b} \right\rangle\right) - t = (0, 0).$$

If the hashes are unequal, we return 0 in response to $\mathcal{A}$'s queries. Otherwise, we consider two cases: if $\mathbf{x}[j] = 0$ and $\mathbf{b}[j] = 0$, then $O(\widehat{\mathbf{k}}, \mathbf{d}, \mathbf{t}) = O(\mathbf{k}, \mathbf{d}, \mathbf{t})$ (which we can compute, as $\mathbf{k}$ is fixed). Otherwise, $\mathsf{ComputeKey}$ gives us the k from the guessing game if and only if $O(\widehat{\mathbf{k}}, \mathbf{d}, \mathbf{t})$ succeeds.

Then, $\mathcal{B}$ succeeds with probability at least negligibly close to the probability $\mathcal{A}$ succeeds in the soundness game. Recall that $\mathcal{A}$ succeeds if it outputs $(\mathbf{d}, \mathbf{t})$ that pass verification and write some non-zero value to channel j. In this case, $\mathsf{ComputeKey}$ gives the k required by the guessing game unless $\mathcal{A}$ outputs $\mathbf{d}$ with distinct encoded messages but the same hash values. This happens with at most negligible probability, by definition of a collision-resistant hash function.

We have an upper-bound on the probability that $\mathcal{B}$ succeeds, so

$$\begin{aligned} &\Pr\left[\mathcal{A} \text{ succeeds in the soundness game}\right] \\ \leqslant\ &\Pr\left[\mathcal{B} \text{ succeeds in the guessing game}\right] + \mathsf{negl}(\lambda) \\ =\ &\frac{\mathsf{poly}(\lambda) + 1}{|\mathcal{K}|} + \mathsf{negl}(\lambda) \end{aligned}$$

which is negligible in λ, since $|\mathcal{K}|$ is exponential in λ. □

Claim 3. *The construction is private.*

Proof. Fix any proper subset of server indexes $I \subset [L]$, message $m \in \mathcal{M}$, point key vector $\mathbf{k} \in \mathcal{K}^L$, and channel index j. Then, define

$$\begin{aligned}
\mathsf{Sim}_I() = &\left\{ \left(\widehat{m}, [\![\mathbf{x}]\!]_i, [\![\mathbf{b}]\!]_i \right), [\![t]\!]_i, \boldsymbol{\beta} \mid i \in [n] \right\} \\
&\qquad \text{where } \widehat{m} \xleftarrow{\$} \mathcal{M}, \\
&\qquad\qquad [\![\mathbf{x}]\!] \leftarrow \mathsf{Share}(\mathbf{0}), \\
&\qquad\qquad [\![\mathbf{b}]\!] \leftarrow \mathsf{Share}(\mathbf{0}), \\
&\qquad\qquad [\![t]\!] \leftarrow \mathsf{Share}((0,0)), \\
&\qquad \forall i \in [n], \; [\![c]\!]_i \leftarrow \left(\langle \mathbf{k}, [\![\mathbf{x}]\!]_i \rangle, \langle \mathbf{k}, [\![\mathbf{b}]\!]_i \rangle \right) - [\![t]\!]_i \\
&\text{and } \forall i \in [n], \quad \boldsymbol{\beta}_i \leftarrow (H(\widehat{m}), [\![c]\!]_i)
\end{aligned}$$

We first note that this simulator produces *exactly* the same distribution as a view over DPF.Empty:

$$\mathsf{Sim}_I() = \{(\mathbf{d}[i], \mathbf{t}[i], \boldsymbol{\beta}) \mid i \in I\}_{\substack{\mathbf{d} \leftarrow \mathsf{DPF.Empty}() \\ \mathbf{t} \leftarrow \mathsf{DPF.TagEmpty}(\mathbf{d}) \\ \boldsymbol{\beta} \leftarrow \{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i'], \mathbf{t}[i']) \mid i' \in [n]\}.}} .$$

We now argue that this simulator is computationally indistinguishable from a view over DPF.Tag:

$$\mathsf{Sim}_I() \approx_c \{(\mathbf{d}[i], \mathbf{t}[i], \boldsymbol{\beta}) \mid i \in I\}_{\substack{\mathbf{d} \leftarrow \mathsf{DPF.Gen}(j, m) \\ \mathbf{t} \leftarrow \mathsf{DPF.Tag}(\mathbf{d}, j, \mathbf{k}[j]) \\ \boldsymbol{\beta} \leftarrow \{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i'], \mathbf{t}[i']) \mid i' \in [n]\}}} .$$

Suppose towards contradiction that some PPT adversary $\mathcal{A}$ could distinguish the two distributions. Then, we use $\mathcal{A}$ to build adversary $\mathcal{B}$ to distinguish $G(x)$ from random elements of $\mathcal{M}$ (where $x \xleftarrow{\$} \mathbb{Z}_p$):

```
$\mathcal{B}(1^\lambda, y)$
1  $V \leftarrow \mathsf{Sim}_I$
2  replace $\widehat{m}$ and $H(\widehat{m})$ in V with $y \oplus m$ and $H(y \oplus m)$ respectively
3  return $\mathcal{A}(1^\lambda, V)$
```

If $y \xleftarrow{\$} \mathcal{M}$, then $y \oplus m$ is also distributed uniformly over the message space, and V is distributed exactly as in Sim_I. Otherwise, $y \sim G(x)$ where $x \xleftarrow{\$} \mathbb{Z}_p$ so $\widehat{m}$ in V is distributed as $G(x) \oplus m$. Because any two identically-sized incomplete subsets of secret shares are distributed exactly the same (even for shares of different values), this is exactly the distribution of the view for DPF.Gen. Therefore, $\mathcal{B}$ breaks the security of the PRG—a contradiction. □

3.6 THE SPECTRUM PROTOCOL

Fix any (L, n)-DPF equipped with authorization methods, as above. Let $\mathcal{M}$ be the message space of the DPF, $\mathcal{T}$ be the tag space of the DPF, and $\mathcal{B}$ be the audit space of the DPF. Spectrum for L channels and n servers consists of this DPF and minor syntactic glue. The message space and audit space of the protocol are $\mathcal{M}$ and $\mathcal{B}$

respectively; the request space $\mathcal{R}$ is $\mathcal{M} \times \mathcal{T}$.

$$\begin{aligned}
\mathsf{Broadcast}(m, j, k) &= \{\mathbf{d}[i], \mathbf{t}[i] \mid i \in [n]\} \\
&\quad \text{where } \mathbf{d} \leftarrow \mathsf{DPF.Gen}(m, j) \text{ and} \\
&\quad \phantom{\text{where }} \mathbf{t} \leftarrow \mathsf{DPF.Tag}(\mathbf{d}, j, k) \\
\mathsf{Cover}() &= \{\mathbf{d}[i], \mathbf{t}[i] \mid i \in [n]\} \\
&\quad \text{where } \mathbf{d} \leftarrow \mathsf{DPF.Empty}() \text{ and} \\
&\quad \phantom{\text{where }} \mathbf{t} \leftarrow \mathsf{DPF.TagEmpty}(\mathbf{d}) \\
\mathsf{Audit}\big(\mathbf{k}, r = (d, t)\big) &= \mathsf{DPF.Audit}(\mathbf{k}, d, t) \\
\mathsf{Verify}(\boldsymbol{\beta}) &= \mathsf{DPF.Verify}(\boldsymbol{\beta}) \\
\mathsf{Accept}\big(r = (d, t)\big) &= \mathsf{DPF.Eval}(d) \\
\mathsf{Combine}(\mathbf{m}) &= \sum_{m \in \mathbf{m}} m.
\end{aligned}$$

This protocol is correct and secure:

Theorem 1. *Let* $(\mathsf{DPF.Gen}, \mathsf{DPF.Eval}, \mathsf{DPF.Empty})$ *be a* DPF *satisfying correctness and privacy and* $(\mathsf{DPF.Tag}, \mathsf{DPF.TagEmpty}, \mathsf{DPF.Audit}, \mathsf{DPF.Verify})$ *be associated* DPF *authentication algorithms satisfying completeness, soundness, and privacy. Spectrum instantiated with these algorithms satisfies* correctness, completeness, soundness, *and* privacy *for anonymous broadcast (as defined in section 3.2.1).*

Proof. We show each property in turn.

Correctness We have:

$$\begin{aligned}
\mathsf{Combine}\big(\{\mathsf{Accept}(r) \mid r \in \mathsf{Cover}()\}\big) &= \sum_{d \in \mathsf{DPF.Empty}()} \mathsf{DPF.Eval}(d) \\
&= \mathbf{0} \qquad \text{(correctness of } \mathsf{DPF.Empty}\text{)}
\end{aligned}$$

and, for all messages $m \in \mathcal{M}$ and channels $j \in [L]$,

$$\begin{aligned}
\mathsf{Combine}\big(\{\mathsf{Accept}(r) \mid r \in \mathsf{Broadcast}(k, j, m)\}\big) &= \sum_{d \in \mathsf{DPF.Gen}()} \mathsf{DPF.Eval}(d) \\
&= m \cdot \mathbf{e}_j. \qquad \text{(correctness of } \mathsf{DPF.Gen}\text{)}
\end{aligned}$$

For any message vector $\mathbf{m} \in \mathcal{M}^L$, key vector $\mathbf{k} \in \mathcal{K}^L$, and non-negative number of describers C, define (as in the definition of correctness):

$$\begin{aligned}
\mathbf{r} &\leftarrow \{\mathsf{Broadcast}(\mathbf{k}[j], j, \mathbf{m}[j]) \mid j \in [L]\} \\
\mathbf{r}' &\leftarrow \{\mathsf{Cover}() \mid j' \in [C]\} \\
\mathbf{m}' &\leftarrow \{\mathsf{Accept}(r) \mid r \in \mathbf{r} \cup \mathbf{r}'\}.
\end{aligned}$$

By the associativity and commutativity of addition in $\mathcal{M}$, for any permutation π we have:

$$\begin{aligned}\mathsf{Combine}(\pi(\mathbf{m}')) &= \sum_{m \in \pi(\mathbf{m}')} m \\ &= \sum_{j \in [L]} \mathbf{m}[j] \cdot \mathbf{e}_j + \sum_{c \in C} \mathbf{0} \\ &= \mathbf{m}\end{aligned}$$

as required.

Completeness Fix any sequence of channel broadcast keys $\mathbf{k}$ in $\mathcal{K}^L$, channel index j in $[L]$, and message $m \in \mathcal{M}$. Then,

$$\begin{aligned}&\Pr\begin{bmatrix}\mathbf{r} \leftarrow \mathsf{Broadcast}(\mathbf{k}[j], j, m) \\ \text{s.t. } \mathsf{Verify}(\{\mathsf{Audit}(\mathbf{k}, r) \mid r \in \mathbf{r}\}) = 1\end{bmatrix} \\ &= \Pr\begin{bmatrix}\mathbf{d} \leftarrow \mathsf{DPF.Gen}(m, j); \\ \mathbf{t} \leftarrow \mathsf{DPF.Tag}(\mathbf{d}, j, \mathbf{k}[j]) \\ \text{s.t. } \mathsf{DPF.Verify}\left(\{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i], \mathbf{t}[i])\}_{i \in [n]}\right) = 1\end{bmatrix} && \text{(def. Spectrum)} \\ &= 1. && \text{(completeness of DPF)}\end{aligned}$$

and

$$\begin{aligned}&\Pr\begin{bmatrix}\mathbf{r} \leftarrow \mathsf{Cover}() \\ \text{s.t. } \mathsf{Verify}(\{\mathsf{Audit}(\mathbf{k}, r) \mid r \in \mathbf{r}\}) = 1\end{bmatrix} \\ &= \Pr\begin{bmatrix}\mathbf{d} \leftarrow \mathsf{DPF.Empty}(); \\ \mathbf{t} \leftarrow \mathsf{DPF.TagEmpty}(\mathbf{d}) \\ \text{s.t. } \mathsf{DPF.Verify}\left(\{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i], \mathbf{t}[i])\}_{i \in [n]}\right) = 1\end{bmatrix} && \text{(def. Spectrum)} \\ &= 1 && \text{(completeness of DPF)}\end{aligned}$$

as required.

Soundness Suppose towards contradiction that there exists some PPT adversary $\mathcal{A}$, vector of channel keys $\mathbf{k} \in \mathcal{K}^L$, and channel index $j \in [L]$ (with O defined as in section 3.2.1) such that

$$\Pr\begin{bmatrix}\widehat{\mathbf{k}} \leftarrow \mathbf{k} \text{ with } \widehat{\mathbf{k}}[j] \xleftarrow{\$} \mathcal{K} \\ \mathbf{r} \leftarrow \mathcal{A}^{O(\widehat{\mathbf{k}}, \cdot)}(1^\lambda, \mathbf{k}, j) \\ \text{s.t. } \mathsf{Verify}\big(\{\mathsf{Audit}(\widehat{\mathbf{k}}, r) \mid r \in \mathbf{r}\}\big) = 1 \\ \text{and } \mathsf{Combine}(\mathbf{r})[j] = m\end{bmatrix} = \varepsilon(\lambda)$$

for some non-negligible $\varepsilon(\cdot)$. We then use $\mathcal{A}$ to build adversary $\mathcal{A}'$ to break the soundness of the DPF authorization with oracle access to O', where O' is the oracle defined in section 3.5.1:

```
$\mathcal{A}'^{\,\mathsf{O}'(\mathbf{k},\cdot,\cdot)}(1^\lambda, \widehat{\mathbf{k}}, j)$
$\mathbf{r} \leftarrow \mathcal{A}(1^\lambda, \widehat{\mathbf{k}}, j)$
    answer query $\mathsf{O}(\mathbf{k}, (r, t))$ with:
        **return** $\mathsf{O}'(\mathbf{k}, r, t)$
$(\mathbf{d}, \mathbf{t}) \leftarrow \mathbf{r}$
**return** $(\mathbf{d}, \mathbf{t})$
```

Then, we have:

$$\begin{aligned}
&\Pr\left[\begin{array}{l} \widehat{\mathbf{k}} \;\leftarrow \mathbf{k} \text{ with } \widehat{\mathbf{k}}[j] \xleftarrow{\$} \mathcal{K}; \\ (\mathbf{d}, \mathbf{t}) \leftarrow \mathcal{A}'^{\,\mathsf{O}'(\widehat{\mathbf{k}},\cdot,\cdot)}(1^\lambda, \mathbf{k}, j); \\ \;\boldsymbol{\beta} \;\leftarrow \{\mathsf{DPF.Audit}(\mathbf{k}, \mathbf{d}[i], \mathbf{t}[i]) \mid i \in [n]\} \\ \text{s.t. } \;\mathsf{DPF.Verify}\,(\boldsymbol{\beta}) = 1 \\ \text{and } \displaystyle\sum_{d \in \mathbf{d}} (\mathsf{DPF.Eval}(d))[j] \neq 0 \end{array}\right] \\
&= \Pr\left[\begin{array}{l} \widehat{\mathbf{k}} \;\leftarrow \mathbf{k} \text{ with } \widehat{\mathbf{k}}[j] \xleftarrow{\$} \mathcal{K} \\ \mathbf{r} \;\leftarrow \mathcal{A}^{\mathsf{O}(\widehat{\mathbf{k}},\cdot)}(1^\lambda, \mathbf{k}, j) \\ \text{s.t. } \;\mathsf{Verify}\big(\{\mathsf{Audit}(\widehat{\mathbf{k}}, r) \mid r \in \mathbf{r}\}\big) = 1 \\ \text{and } \mathsf{Combine}(\mathbf{r})[j] = m \end{array}\right] \\
&= \varepsilon(\lambda).
\end{aligned}$$

This is a contradiction: $\varepsilon(\lambda)$ is non-negligible in λ, but by the soundness of the DPF authentication scheme, the above probability must be negligible. Therefore, Audit and Verify must be sound.

Privacy We want to show:

$$\big\{\{\mathbf{r}[i'] \mid i' \neq i\}, \boldsymbol{\beta}, \mathbf{k}\big\}_{\mathbf{r},\boldsymbol{\beta},\mathbf{k}} \approx_c \big\{\{\mathbf{r}'[i'] \mid i' \neq i\}, \boldsymbol{\beta}', \mathbf{k}\big\}_{\mathbf{r}',\boldsymbol{\beta}',\mathbf{k}}$$

where we sample the variables as

$$\begin{array}{lcl} \mathbf{r} \leftarrow \mathsf{Broadcast}(\mathbf{k}[j], j, \mathbf{m}[j]), & & \mathbf{r}' \leftarrow \mathsf{Cover}(), \\ \boldsymbol{\beta} \leftarrow \{\mathsf{Audit}(\mathbf{k}, r) \mid r \in \mathbf{r}\}, & \text{and} & \boldsymbol{\beta}' \leftarrow \{\mathsf{Audit}(\mathbf{k}, r) \mid r \in \mathbf{r}'\}. \end{array}$$

Replacing Broadcast, Cover, and Audit with their definitions, we want to show:

$$\begin{aligned}
&\big\{(\mathbf{d}[i], \mathbf{t}[i], \boldsymbol{\beta}[i]) \mid i \in I\big\}_{\substack{\mathbf{d} \leftarrow \mathsf{DPF.Gen}(j,m) \\ \mathbf{t} \leftarrow \mathsf{DPF.Tag}(\mathbf{d},j,\mathbf{k}[j]) \\ \boldsymbol{\beta} \leftarrow \{\mathsf{DPF.Audit}(\mathbf{k},\mathbf{d}[i'],\mathbf{t}[i']) \mid i' \in [n]\}}} \\
\approx_c &\big\{(\mathbf{d}[i], \mathbf{t}[i], \boldsymbol{\beta}[i]) \mid i \in I\big\}_{\substack{\mathbf{d} \leftarrow \mathsf{DPF.Empty}() \\ \mathbf{t} \leftarrow \mathsf{DPF.TagEmpty}(\mathbf{d}) \\ \boldsymbol{\beta} \leftarrow \{\mathsf{DPF.Audit}(\mathbf{k},\mathbf{d}[i'],\mathbf{t}[i']) \mid i' \in [n]\}.}}
\end{aligned}$$

By the definition of privacy for DPF authentication, the output distribution of DPF authentication simulator Sim_I is computationally indistinguishable from both of the above distributions. Therefore, they must be computationally indistinguishable from each other.

Since, by assumption, the DPF algorithms are correct, complete, sound, and private, Spectrum instantiated with these algorithms is correct, complete, sound, and private. $\square$

4 EVALUATION

In our evaluation, we demonstrate that Spectrum:

– performs comparably to prior works when the number of clients and the number of channels are the same—that is, *every* user broadcasts some message;

– outperforms those works when comparatively few clients upload—the setting for which Spectrum was designed;

– achieves throughput supporting live video streaming with a moderate number of users;

– scales by sharding each logical protocol server into several physical servers; and

– scales beyond two servers without sacrificing performance (after an initial penalty in switching to a protocol which can handle more than two servers).

4.1 IMPLEMENTATION AND DEPLOYMENT

We implement Spectrum in 8,000 lines of Rust (`2020-03-22` nightly build); the source is publicly available.[1] For the pseudorandom generator (PRG), we use AES-128 in CTR mode. For collision-resistant hashing, we use BLAKE3 [53]. For the multi-server setting, we require a seed-homomorphic PRG [10]; we instantiate this using the Jubjub [36] twisted Edwards curve.

For our experiments, we deploy Spectrum to Amazon Elastic Cloud Compute (EC2). We use one virtual machine (VM) per server (except in the horizontal sharding experiments), one VM for coordinating the experiment, and more VMs to simulate client traffic (between 250 and 500 clients per VM). Each server runs on a `m5.24xlarge`[2] EC2 instance (with smaller `m5.xlarge` machines for simulating client traffic and coordination). On the AES-NI enabled processors of these server machines, `openssl speed`, reports a throughput of 5.4 GB/s for AES-128-CTR. We run Ubuntu 18.04 LTS on each instance, and deploy our application behind Nginx. All servers run in the same region, and network round-trip times are about 1 ms. As of April 2020, each server VM costs $4.06 per hour.

[1] `https://www.github.com/znewman01/spectrum-impl`

[2] Full machine specifications are available at `https://aws.amazon.com/ec2/instance-types/m5/`.

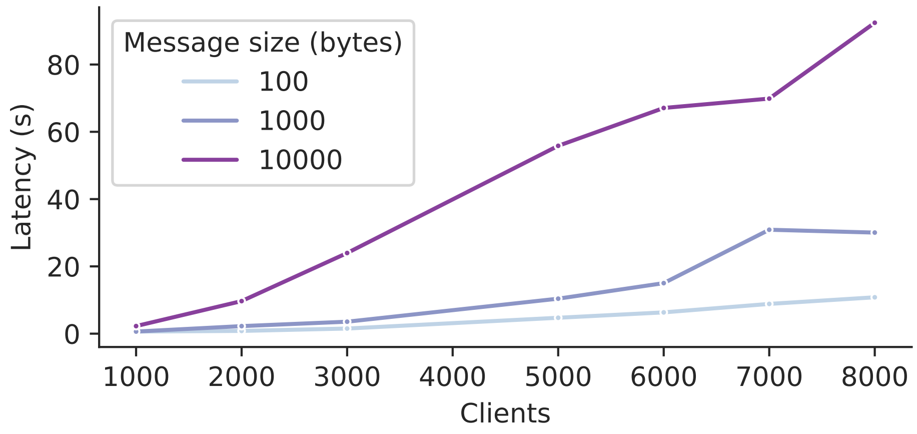

Figure 4-1: For direct comparison to prior works, we evaluate Spectrum with small messages (140 B–80 kbit) and an identical number of clients and channels: that is, *every* user is a publisher.

We measured the end-to-end time from the first client request sent to the time at which the servers recovered all messages, and use this to compute throughput and latency.

4.2 RESULTS

COMPARISON TO PRIOR WORKS Prior works assumed that *each* user broadcasts a message. We simulate this setting in Spectrum by setting the number of clients and servers equal. We compare with Riposte [22] and Blinder [2], as these systems have similar security guarantees and deployment configurations. We use the results reported by Abraham et al. [2] for these systems, and our results (fig. 4-1) for Spectrum.

For 10 kB messages, Spectrum measures 88 queries per second (QPS), holding its own against even the GPU-accelerated variant of Blinder. Blinder-CPU achieves 21 queries per second (QPS) and Blinder-GPU achieves 111 QPS. For smaller messages (100 B), Spectrum is better compared to Blinder-CPU: both systems achieve 1,000 QPS (double that of Riposte). Blinder-GPU outpaces both, measuring about 2,500 QPS. We conclude that Spectrum with two servers is not substantially slower than Blinder, even when all users broadcast. However, we predict a 3× performance hit when scaling beyond two servers (see below).

THE FEW-BROADCASTER SETTING On Twitch, 10,000 passive users watch each stream on average; other streaming platforms show a similar ratio [49, 71]. However, Blinder and Riposte are not equipped to leverage this asymmetry: as the number of users grows, so does the work needed per user. In contrast, Spectrum excels with relatively few publishers compared to the total number of users. Figure 4-2 shows Spectrum deployed with larger messages (between 10 kB and 8 Mbit). We

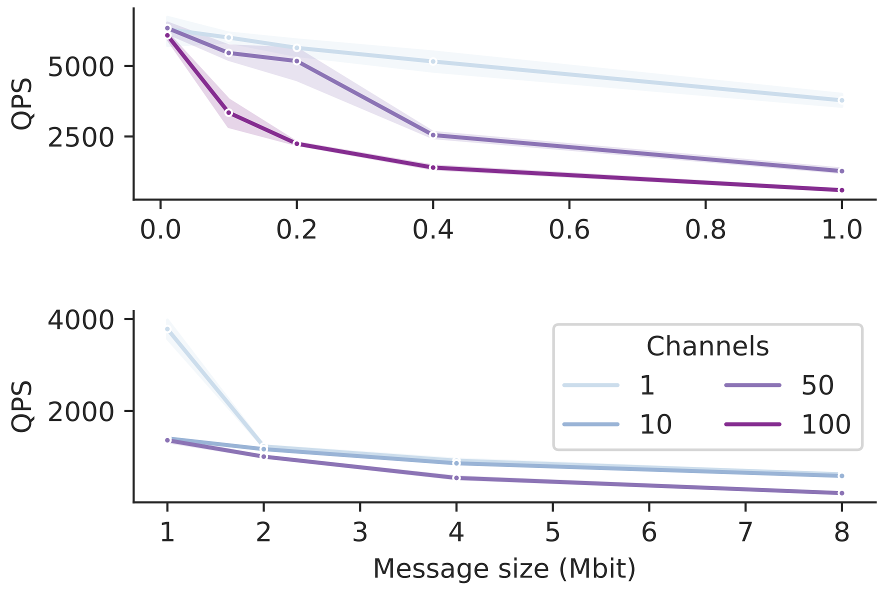

Figure 4-2: Here, we see Spectrum with fewer publishers than clients. We measure the rate of client query processing with one virtual machine with 1, 50, and 100 channels (upper graph) or 1, 10, and 50 channels (lower graph). The shaded regions indicate a 95% confidence interval over 5 trials.

observe an immediate 70× speed-up in the setting with 1 channel and 10 kB messages. While speed decreases as message sizes grow larger, we see favorable performance with messages up to 1 Mbit. These results suggest that Spectrum can support streaming live video with an anonymity set of about 5,000 users, or FHD video with about 1,000 users (table 4.1). Optimizing for throughput, Spectrum can support 3.4 GB/h with about 600 users.

Format		**Bandwidth** (Mbit/s)
240p		0.3 – 0.7
480p	(SD)	0.5 – 2.0
720p	(HD)	1.5 – 4.0
1080p	(FHD)	3.0 – 6.0

Table 4.1: Bandwidth required for video streaming [72]

We speculate that the sharp drop-off around 8 Mbit results from an issue with our experimental set-up: perhaps the client traffic simulators have saturated their network uplinks, or the overall bandwidth starts to cause TCP connections to be rejected.

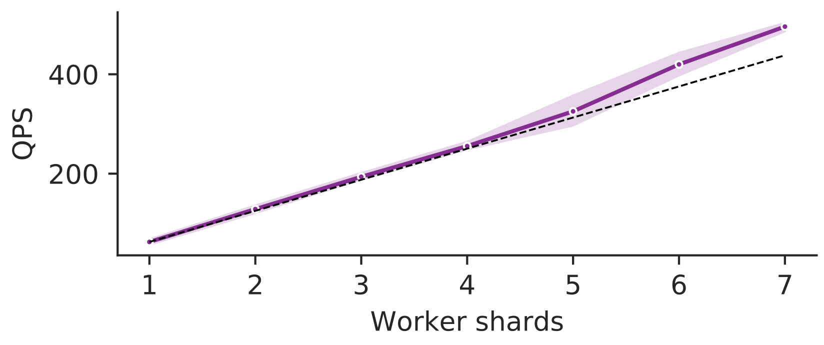

Figure 4-3: For a challenging workload (500 channels of 1 Mbit messages), sharding within each group increases throughput. The dashed line indicates "perfect scaling" (where doubling the servers doubles the QPS) based on the mean for 1 shard. The shaded regions indicate a 95% confidence interval over 5 trials.

Horizontal sharding As noted before, request processing for each client is embarrassingly parallel, and can be sharded among several physical worker servers composing a single logical protocol server. In theory, the speed-up from this should be near-perfect: only the `Combine` step needs to access data from multiple client requests. Further, `Combine` is a commutative and associative operation, so worker shards can combine their shares independently. Practice bears this out (fig. 4-3). With up to 7 worker shards, we see *over* 7× the throughput of 1 shard (we suspect this is due to inelegant failure behavior as we push the single worker to its limits).

With sharding, real-world deployments of Spectrum can use low-cost, commodity servers to scale to tens of thousands of users while supporting bandwidths supporting real-time video.

Beyond two servers We describe Spectrum in with symmetric distributed point functions (instantiated using *any* pseudorandom generator PRG) and a many-server construction (using seed-homomorphic PRGs) (section 3.4, respectively). If we can use any PRG, we can choose efficient constructions. As mentioned, we use AES-128 in CTR mode, which benefits from hardware support on AES-NI processors. Constructions of seed-homomorphic PRGs use groups in which the Decisional Diffie Hellman problem is hard—in practice, using elliptic curves. Public-key operations over these groups require an order of magnitude greater computation for the same quantity of data. Consequently, we would expect the seed-homomorphic version of the protocol to be much slower. In fig. 4-4, we see a 3× drop-off, though we observe a minimal cost to increase the number of servers beyond this initial penalty.

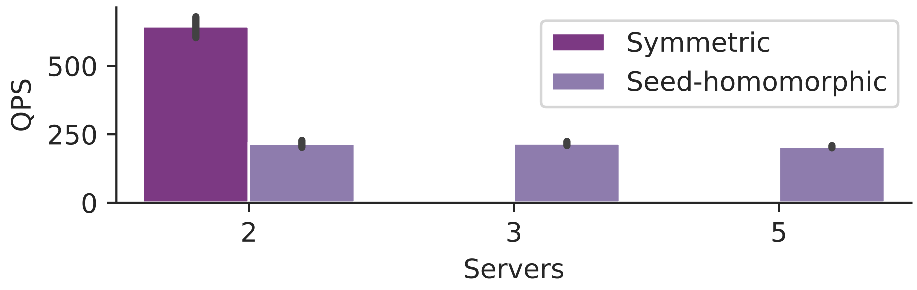

Figure 4-4: To scale beyond two servers, we must switch from the fast protocol based on symmetric pseudorandom generators to a "seed-homomorphic" protocol using public-key pseudorandom generators. With 100 channels of 1 Mbit messages, we observe a $3\times$ drop-off in throughput when switching to the slower protocol with only minor overhead for adding additional servers.

5 DISCUSSION

5.1 EXTENSIONS

Practical deployments of an anonymous broadcast system might benefit from additional features for managing content. Because these features are orthogonal to our primary concerns about scalability, we omit them in our implementation.

TIME-SLICED MULTIPLEXING While we described Spectrum with a one-to-one correspondence between publishers and channels, nothing prevents several users from sharing a channel. This would act as a restricted-access public bulletin board allowing, for instance, any member of an organization to post.

HANDLING DROPOUT There are two obvious ways to handle users who participate in one round of the protocol but not later rounds. First, we can halt the system. This allows any user to perform a denial-of-service attack by withdrawing from the protocol. Second, we can proceed with the protocol, as long as some quorum is met. This gives an active adversary that can terminate network connections some chance (bounded by the quorum size) of identifying users.

PRESERVING SUBSCRIBER ANONYMITY Techniques from private information retrieval (PIR) [17] underlie Spectrum: in particular, clients use "reverse PIR" or "PIR writing" to write to, in effect, a shared database. This preserves writer anonymity, but not necessarily reader anonymity: consuming sensitive content exposes a user to the server from which it ultimately downloads this content. This is exactly the setting for traditional PIR, which allows reading one of a number of database rows (that is, channels) while hiding *which row* was read. These techniques apply straightforwardly.

With a small number of channels, we can precompute responses to all possible PIR queries. For instance, using the Chor et al. [17] construction with 10 channels, we would have 1,024 possible queries sent to each server. If the number of clients exceeds this value, these servers can precompute the responses, and even distribute these using wide-scale content-distribution networks (CDNs), which efficiently serve static content.

Further, the access control technique we use for writing applies equally well to traditional PIR—allowing access controlled premium or private channels, potentially mixed with channels without access control.

Content moderation We note that Spectrum is *not* censorship-resistant: for users to download data, the servers must first recover that data; for cleartext messages, the servers can opt not to publish any data which they deem illegal or inappropriate. Typical moderation workflows apply for deployments with many channels: users can flag offending content (perhaps using an anonymous reporting system, such as Prio [20]).

5.2 future work

Eliminate pre-shared Keys The primary limitation of this work not shared with other anonymous broadcasting schemes is the setting: in particular, the requirement to pre-share a key. This sharing process must be similarly anonymous to the main protocol procedure; otherwise, intersection attacks apply. We leave as an open question the problem of anonymous broadcasting with per-user computation and bandwidth independent of the number of users, no pre-shared keys, and no interference between users.

Public-key authentication The access control scheme described in section 3.5.1 can be verified using asymmetric keys: given a generator g for a group (where the decisional Diffie-Hellman assumption is believed hard), we can perform the same arithmetic in the exponent of the generator. The publisher retains the exponent k, and sends a public key g^k. Then, even acting collectively, the servers cannot forge a tag given only g^k. We omit this from the primary description of the protocol, as it does not affect our security goals.

Optimizations Further optimizations might increase the utility of the system. In addition to the typical performance and robustness enhancements appropriate for any distributed system, protocol enhancements. As evaluated, the many-server version of the protocol requires an expensive seed-homomorphic pseudorandom generator (prg) expansion on each request, but seed-homomorphic prg seeds can efficiently aggregated without being expanded. The protocol could leverage distributed point functions (dpf) enhanced with an efficient key-aggregation mechanism built from seed-homomorphic prgs.

Currently, upload bandwidth presents the greatest bottleneck for clients, who may not have professional-grade internet connections. Users must send data at least the length of the message to *each server*. Further, this data is the *same* for each server. Assuming relatively faster inter-server network connections, clients might send this data to just one server, who can then share it with the others.

These, along with other enhancements (faster dpfs or prgs), would make Spectrum more practical.

Private access control We have already noted that the authentication mechanism used in Spectrum for writing applies equally well to reading. Our notions of privacy, completeness, and soundness apply to *any* oblivious operation.

5.3 CONCLUSION

This thesis describes Spectrum, a system for high-performance anonymous broadcasting. As evaluated, Spectrum running on two servers can stream video to 5,000 users, enabling whistleblowers and journalists to enjoy anonymity protections. For large documents, we see speeds of 8 Mbit/s (3.4 GB/h) for 600 clients. This would enable sharing a document like the *Pentagon Papers* or the "Cablegate" leaks in an hour or two. Faster speeds would have limited application, as many home and public internet connections top out between 5 and 10 Mbit/s; any broadcast system with similar performance guarantees is limited by the slowest upload speed among its users. Further, Spectrum can support more clients by sharding each server, allowing still-larger anonymity sets. Spectrum introduces authenticated anonymous broadcast and adds a high-throughput tool to the pantheon of technologies for anonymous communication.

BIBLIOGRAPHY

[1] Report of the Office of the Secretary of Defense Vietnam Task Force. URL https://www.archives.gov/research/pentagon-papers, 1969. Accessed: 2020-05-04.

[2] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder: MPC based scalable and robust anonymous committed broadcast. Cryptology ePrint Archive, Report 2020/248, 2020. URL https://eprint.iacr.org/2020/248.

[3] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. MCMix: Anonymous messaging via secure multiparty computation. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1217–1234, 2017.

[4] American Society for the Prevention of Cruelty to Animals. What is ag-gag legislation? URL https://www.aspca.org/animal-protection/public-policy/what-ag-gag-legislation, 2020. Accessed: 2020-04-17.

[5] 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.

[6] 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.

[7] Josh Cohen Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 251–260. Springer, 1986.

[8] Carl Bernstein and Bob Woodward. *All The President's Men*. Simon & Schuster, New York, 1974.

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

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

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

[12] Bryan Burrough, Sarah Ellison, and Suzanna Andrews. The Snowden saga: A shadowland of secrets and light. *Vanity Fair*, 23 April 2014. URL https://www.vanityfair.com/news/politics/2014/05/edward-snowden-politics-interview. Accessed: 2020-05-01.

[13] Ed Caesar. Bradley Manning: Wikileaker. *The Sunday Times*, 19 December 2010. URL https://web.archive.org/web/20130514070751/http://www.edcaesar.co.uk/article.php?article_id=53. Accessed: 2020-05-01.

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

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

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

[17] 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.

[18] Nicola Clark. How a cryptic message, "Interested in data?," led to the Panama Papers. *The New York Times*, 5 April 2016. URL https://www.nytimes.com/2016/04/06/business/media/how-a-cryptic-message-interested-in-data-led-to-the-panama-papers.html. Accessed: 2020-05-04.

[19] David Cole. "We kill people based on metadata". *The New York Review of Books*, 10 May 2014. URL https://www.nybooks.com/daily/2014/05/10/we-kill-people-based-metadata/. Accessed: 2020-05-04.

[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] Douglas Dalby and Amy Wilson-Chapman. Panama Papers helps recover more than $1.2 billion around the world. *International Consortium of Investigative Journalists Investigations*, 3 April 2019. URL https://www.icij.org/investigations/panama-papers/panama-papers-helps-recover-more-than-1-2-billion-around-the-world/. Accessed: 2020-05-04.

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

[27] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. *arXiv preprint arXiv:1911.09215*, 2019.

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

[29] Juliette Garside, Holly Watt, and David Pegg. The Panama Papers: how the world's rich and famous hide their money offshore. *The Guardian*, 3 April 2016. URL https://www.theguardian.com/news/2016/apr/03/the-panama-papers-how-the-worlds-rich-and-famous-hide-their-money-offshore. Accessed: 2020-05-04.

[30] Anita Gates and Katharine Q. Seelye. Linda Tripp, key figure in Clinton impeachment, dies. *The New York Times*, 8 April 2020. URL https://www.nytimes.com/2020/04/08/us/politics/linda-tripp-dead.html. Accessed: 2020-05-01.

[31] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 640–658. Springer, 2014.

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

[33] Hansard. House of Commons Debate. vol. 406 cols. 925–926. URL https://hansard.parliament.uk/commons/1944-12-08/debates/e8e76916-df42-4315-b2e7-f923d2181bc3/LiberatedEurope(BritishIntervention), 8 December 1944. Accessed: 2020-05-01.

[34] Nick Hopkins and Helena Bengtsson. What are the Paradise Papers and what do they tell us? *The Guardian*, 5 November 2017. URL https://www.theguardian.com/news/2017/nov/05/what-are-the-paradise-papers-and-what-do-they-tell-us. Accessed: 2020-05-04.

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

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

[37] Kevin Johnson. Letter shifts heat to FBI. *USA Today*, 28 May 2002. URL http://usatoday30.usatoday.com/news/nation/2002/05/28/letter-fbi.htm. Accessed: 2020-05-01.

[38] 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.

[39] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An efficient communication system with strong anonymity. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134, 2016.

[40] 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.

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

[42] Young Hyun Kwon. *Towards anonymous and metadata private communication at internet scale*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, October 2019.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] Ethan May. Streamlabs Q1 2019 live streaming industry report. *Streamlabs*, 18 April 2019. URL https://blog.streamlabs.com/youtube-contends-with-twitch-as-streamers-establish-their-audiences-6a53c7b28147. Accessed: 2020-04-10.

[50] 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.

[51] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, pages 215–226, 2011.

[52] Zachary Newman, Sacha Servan-Schreiber, and Srini Devadas. Spectrum: High-bandwidth, low-latency anonymous broadcasting. 2020.

[53] 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.

[54] John D. O'Connor. "I'm the guy they called Deep Throat". *Vanity Fair*, 17 October 2006. URL https://archive.vanityfair.com/article/2005/7/im-the-guy-they-called-deep-throat. Accessed: 2020-05-01.

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

[56] 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.

[57] 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.

[58] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) protocol version 1.3. 2018.

[59] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[60] 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.

[61] Elizabeth Stoycheff. Under surveillance: Examining Facebook's spiral of silence effects in the wake of NSA internet monitoring. *Journalism and Mass Communication Quarterly*, 93(2):296–311, 2016.

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

[63] The Tor Project. Tor metrics, 2019. URL https://metrics.torproject.org/. Accessed: 2019-09-15.

[64] US Department of Justice, Office of Public Affairs. Nine FIFA officials and five corporate executives indicted for racketeering conspiracy and corruption. Press Release, 2015. URL https://www.justice.gov/opa/pr/nine-fifa-officials-and-five-corporate-executives-indicted-racketeering-conspiracy-and. Accessed: 2020-05-04.

[65] Griswold v. Connecticut. 381 U.S. 479. URL https://www.law.cornell.edu/supremecourt/text/381/479, 1965. Accessed: 2020-05-01.

[66] Obergefell v. Hodges. 576 U.S. 644. URL https://www.law.cornell.edu/supremecourt/text/14-556, 2015. Accessed: 2020-05-01.

[67] Lawrence v. Texas. 539 U.S. 558. URL https://www.law.cornell.edu/supremecourt/text/02-102, 2003. Accessed: 2020-05-01.

[68] 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.

[69] 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.

[70] 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.

[71] 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? *StreamElements*, 12 July 2019. URL `https://blog.streamelements.com/state-of-the-stream-q2-2019-facebook-gaming-growth-gta-v-surges-and-twitch-influencers-get-more-529ee67f1b7e`. Accessed: 2020-04-10.

[72] YouTube. Choose live encoder settings, bitrates, and resolutions, 2020. URL `https://support.google.com/youtube/answer/2853702?hl=en`.

[73] 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.