Private Analytics via Streaming, Sketching, and Silently Verifiable Proofs
(Authors' version)

Mayank Rathee
UC Berkeley

Yuwen Zhang
UC Berkeley

Henry Corrigan-Gibbs
MIT

Raluca Ada Popa
UC Berkeley

***Abstract*—We present Whisper, a system for privacy-preserving collection of aggregate statistics. Like prior systems, a Whisper deployment consists of a small set of non-colluding servers; these servers compute aggregate statistics over data from a large number of users without learning the data of any individual user. Whisper's main contribution is that its server-to-server communication cost and its server-side storage costs scale *sublinearly* with the total number of users. In particular, prior systems required the servers to exchange a few bits of information to verify the well-formedness of each client submission. In contrast, Whisper uses *silently verifiable proofs*, a new type of proof system on secret-shared data that allows the servers to verify an arbitrarily large batch of proofs by exchanging a single 128-bit string. This improvement comes with increased client-to-server communication, which, in cloud computing, is typically cheaper (or even free) than the cost of egress for server-to-server communication. To reduce server storage, Whisper approximates certain statistics using small-space sketching data structures. Applying randomized sketches in an environment with adversarial clients requires a careful and novel security analysis. In a deployment with two servers and 100,000 clients of which 1% are malicious, Whisper can improve server-to-server communication for vector sum by three orders of magnitude while each client's communication increases by only 10%.**

1. Introduction

Private-aggregation systems make it possible to compute aggregate statistics about a population of devices, while revealing no information—beyond the aggregate statistic itself—about any device's data. These systems make it possible to privately collect information on user behavior [6], public health trends [10], [65], and device telemetry [90] at million-user scale.

In this paper, we focus on private-aggregation systems based on multi-party computation techniques [3], [13], [26], [43], [45], [48], [50], [57], [61], [63], [72], [81], [86], [89]. These systems require a small set of infrastructure providers ("servers"); the systems protect client privacy as long as an adversary cannot compromise all servers. Deployments of private aggregation at Apple [6], Google [10], Mozilla [90], and others [50] use this approach.

In a run of one such private-aggregation protocol, each user splits its data using a cryptographic secret-sharing scheme, and sends one share to each server. In addition, each user sends the servers a zero-knowledge proof attesting that its secret shares are well-formed. After receiving the data submissions and validity proofs from a large number of clients, the servers verify each proof, and then aggregate the valid submissions to compute the statistic of interest.

An annoyance in prior systems [3], [13], [43], [48], [50], [72] is that the servers must exchange messages to check *each client's* validity proof, so the server-to-server communication cost is linear in the number of clients. When the number of clients is in the millions, this server-to-server communication cost is significant.

More recent systems [26], [74], [89] support computing the "heavy-hitter" statistic: each client holds a string and the statistic computes the set of most popular client-held strings. This statistic is useful when the universe of possible client submission is large—for example, when computing the set of URLs that most often cause a user's browser to crash.

When computing heavy hitters, existing multiparty-computation-based systems [26], [74], [89] require the servers to store an amount of secret state that grows linearly with the number of clients. When the client submissions arrive in a stream [1], the servers cannot begin processing the first client submission until the last submission arrives. As a result, as the number of client uploads increases, the servers' memory and storage requirements balloon.

We present Whisper, a system for privacy-preserving collection of aggregate statistics that has server-to-server communication and server storage costs *sublinear* in the number of users. Whisper provides these properties while supporting the computation both of simple arithmetic statistics and of heavy hitters. Whisper uses the same deployment model as existing systems [43], [48], [74] (Figure 1) and provides the same privacy property: if there is at least one honest server, no adversarial coalition of malicious clients and servers can learn any information about honest clients' data, beyond what the aggregate statistics themselves leak.

Silently verifiable proofs. To reduce server-to-server communication in Whisper, we introduce *silently* verifiable proofs, a new type of zero-knowledge proof system on secret-shared data [25]. In a silently verifiable proof system, the verifiers can verify a batch of proofs by exchanging a single field element. This batch verification is possible even when the provers are mutually distrusting and when each prover is proving a different statement. Clients in Whisper

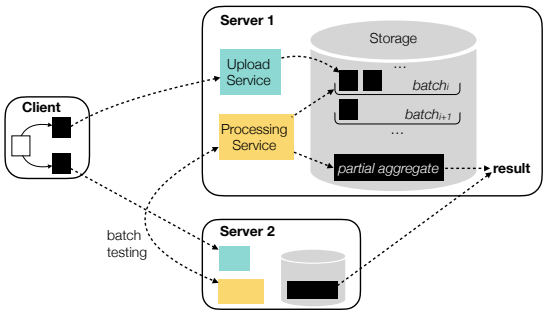

Figure 1: Whisper's server architecture. Clients split their data using a secret-sharing scheme and send one share to each server. The servers process submissions in batches.

use silently verifiable proofs to convince the servers that their data submissions are well formed; the servers can check arbitrarily large batches of proofs using only a few bits of server-to-server communication.

We show how to construct silently verifiable proofs in the random-oracle model from a variety of existing proof systems on secret-shared data [25], [27], [28], [43]. Our construction begins with a recent theoretical proof system [71], which (implicitly) gives a relatively inefficient silently verifiable proof using an MPC-in-the-head-like construction [75]. We then improve the concrete efficiency of their construction by replacing the traditional multiparty computation they use with a prover-aided multiparty computation [25], [80].

To sketch how our silently verifiable proofs work, consider a prover who wants to prove that its input x lies in some language $\mathcal{L}$. Each verifier holds a secret share of the input x. Furthermore, say that we have a zero-knowledge proof system $\Pi_{\mathcal{L}}$ on secret-shared data for the language $\mathcal{L}$ in which the verifiers communicate with each other over a broadcast channel (existing protocols satisfy this property [25], [48]).

To generate the silent proof, the prover locally simulates the execution of all of the parties (prover and verifiers) running the protocol $\Pi_{\mathcal{L}}$. The prover then sends to each verifier (1) a transcript of all messages that the simulated verifiers exchanged via the broadcast channel and (2) the view of each verifier in the simulated protocol. To check the proof, the silent verifiers only need to check that (a) their transcripts of the simulated broadcast channel are identical and (b) their simulated views are correct according to the protocol $\Pi_{\mathcal{L}}$. The verifiers can locally generate secret shares of a test value that is zero if and only if both of these checks pass (with high probability). To check a batch of proofs at once, the verifiers can publish a random linear combination of each proof's test value and accept if the resulting value sums to zero. We depict this construction in Figure 2.

Privately streaming heavy hitters. To avoid needing to store per-client state in our private heavy-hitters computation, we use a small-space sketching data structure [96]. In doing so, we give up on computing the exact heavy hitters, and instead settle for a good approximation—we expect this trade-off to be acceptable in many applications.

Applying a sketch in the context of private aggregation requires some care. In particular, existing sketching data structures can provide a good approximation of the true heavy hitters when the client's data values are chosen *independently* of the random hash functions used in the data structure [34], [40], [41], [82], [95], [96]. However, when clients can contribute maliciously-crafted data submissions that *depend* on the data structure's hash functions, the standard correctness analysis does not suffice [4], [18], [38], [69], [70], [88], [93]. We show how to use silently verifiable proofs, along with a refined analysis of the sketching data structures, to guarantee that we correctly approximate the heavy hitters, even in the face of malicious clients.

We implement Whisper on top of ISRG's `libprio-rs` library [51]. In our evaluation where two servers aggregate 1024-sized vectors across 100,000 clients of which 1% are malicious, each server in Whisper only sends 0.2 MB compared to 415 MB for state-of-the-art Prio3 [51]. In achieving this, our per-client communication increases to 303 KB from 274 KB for Prio3. This trade-off is most appealing in cloud deployments, where ingress is free and egress is costly. We estimate up to 3× reduction in server operating costs for certain statistics. When the same servers compute heavy hitters over a stream of 1.75 million client uploads, our baseline Poplar [26] overruns the 64 GB memory at the servers and takes four days to finish, while Whisper takes about two hours and recovers all the heavy hitters with probability at least 0.999. Streaming the heavy hitter computation in Whisper comes at a 14-17× increase in client communication over Poplar, however, it stays under 500 KB.

2. System Overview

In this section, we outline Whisper's system architecture, capabilities, and security properties.

2.1. System Model

A Whisper deployment consists of two or more logical servers, and a large number of clients. All the communication happens over TLS-protected network channels.

Servers. Each logical server in Whisper server runs in its own administrative domain, separate from all other servers in the system. A logical Whisper server can consist of a large number of physical servers or cloud instances. For conciseness, we use "server" to refer to a logical server. We assume that all participants in the system have the cryptographic public keys of each server in the system. The servers jointly compute the same set of aggregate statistics on the users' data. There are v servers ($v$erifiers).

Clients. Each client holds a piece of private data; the servers compute aggregate statistics over all clients' data. Clients in Whisper communicate with each of the Whisper servers and do not communicate with other clients. We assume that the clients have a means to authenticate to the servers [8].

Threat model and real-world deployments. While Whisper works with any number of servers, we evaluate our

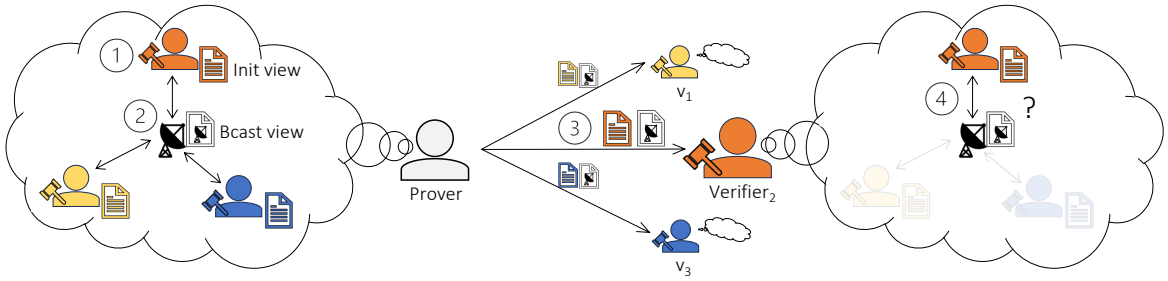

Figure 2: Overview: Constructing silently verifiable proofs from zero-knowledge proofs on secret-shared data. Given a zero-knowledge proof Π on secret-shared data, the prover, in its head, ① initializes each verifier's view, and ② simulates their interaction as per Π to generate the broadcast view. ③ It sends to each real verifier their initial view and the simulated broadcast view. ④ Each verifier locally verifies a part of the simulation to generate a share of the final decision.

system, and present additional optimizations, in the two-server setting. Moreover, real-world deployments [10], [50], [52], [90] of prior work [26], [43] with similar threat models typically adopt the two-server setting. This setting requires two non-colluding servers for privacy (more details in §2.4). The non-collusion assumption is unavoidable here and has to appear either between clients [16], [24], [84] or servers [26], [43]. In recent times, Mozilla [52], [90], NIH (with Apple and Google) [10], and Horizontal [52] have partnered with ISRG to run the second non-colluding server in their deployments. Through the Divvi Up project [50], any organization can request this service from ISRG. We make no synchrony requirement and the adversary can observe all network links.

2.2. Architecture

Whisper computes aggregate statistics in a sequence of time *epochs*: at the end of each epoch, the servers publish a set of aggregate statistics computed over the data of the clients participating during the just-completed epoch. The protocol flow in each epoch works in the following three steps, depicted in Figure 1.

Step 1: Client data submission. Each client authenticates to the *Upload Service* at each server, which associates this client with an id. The client uploads an encoding of its private data with a silently verifiable proof of valid encoding by sending a single message to each server.

The Upload Service associates each message with a specific batch of submissions, a batch corresponding to a time interval. We need to ensure that each client uploads to the *same* batch on each server. A malicious client can try to upload in v different batches at the v servers to cause v times more work for the servers. At the same time, we do not want the "silent" servers to communicate per client to reach consensus. To prevent this issue, Whisper has each client first submit its upload to the first server. This server will verify that this client id has not uploaded already. It will assign this message to a batch and will return a signed acknowledgment σ_{ack} that covers the batch identifier and the client id. The client will then upload to the other servers to the same batch by presenting σ_{ack}.

Step 2: Server data validation and aggregation. After receiving client submissions, the servers check that they are well formed using the silently verifiable proof in each submission. To keep the server state from growing, Whisper servers verify client submissions in batches as they arrive within the epoch. The *Processing Service* processes each batch. It first tests the validity of the submissions in the batch by running the batch-verification routine of the silently verifiable proofs. In the optimistic case—when all clients in a batch are honest—the entire validity check requires the servers to exchange a single short (128-bit) field element. If any proof in the batch is invalid, the servers will identify the failing proof via group-testing techniques [54]; they will discard the corresponding malformed submissions. These steps require interacting with the corresponding Processing Service on the other servers. It then aggregates the values in the batch into a running partial aggregate.

Step 3: Publishing the aggregate statistic. After the servers process all input batches, the Processing Service combines the resulting aggregate with the aggregates on the other servers to obtain the aggregate statistic.

2.3. Supported statistics

The configuration of a Whisper deployment specifies which aggregate statistic f the system will compute in each epoch. Following prior private-aggregation systems [26], [43], [48], Whisper supports any aggregate statistic that can be computed via a verifiable *additive encoding* of the client's data [43], [68]. We discuss additive encodings in more detail in §4. Using encodings from prior work [31], [41], [43], [57], [86], [97], Whisper supports the following statistics:

- *Basic statistics:* SUM, MEAN, VARIANCE, STDDEV, MIN/MAX (over small domains)
- *Counting:* FREQUENCY COUNT, APPROXIMATE FREQUENCY
- *Boolean operations:* AND, OR
- *Machine learning:* LINEAR REGRESSION, r^2 COEFFICIENT

As one of our technical contributions, we show that Whisper can also support computation of approximate *heavy hitters* (popular strings) via a new additive encoding (§5).

In many cases, Whisper reveals to the servers slightly more information about the client inputs $(x_1, \ldots, x_n)$ than the aggregate statistic $f(x_1, \ldots, x_n)$ itself. For example, practical private-aggregation schemes for VARIANCE additionally leak the mean [43]. As in prior work, we define the *leakage* $\hat{f}(x_1, \ldots, x_n)$ of the encoding to capture this extra information. In Whisper, the leakage function is always *symmetric* in its inputs—so the leakage reveals no information about which client i held which private input x_i.

2.4. Security properties

Whisper's security properties are similar to existing privacy-preserving systems for collecting aggregate statistics. We describe these properties in more detail in Sections 4.2 and 5; we sketch them here. All of the security properties are relative to an aggregate statistic f and an associated leakage function $\hat{f}$.

- *Privacy*. As long as one server is honest, no server or malicious client learns any information about the honest clients' data $x_1, \ldots, x_n$, except what can be inferred from the aggregate statistic $f(x_1, \ldots, x_n)$ and the leakage function $\hat{f}(x_1, \ldots, x_n)$. All the statistics f that Whisper supports and their leakage functions $\hat{f}$ are symmetric in their inputs, and therefore, the output reveals no information about which client submitted which input.
- *Correctness against malicious clients*. If all the servers are honest, then a small set of malicious clients can only affect the aggregate statistic f by misreporting their private data. When f computes heavy hitters, we allow malicious clients to introduce some small additional error in the output with low probability (Theorem 5.2).

For privacy, it is important that "enough" honest clients participate in each epoch. This ensures that $f(x_1, \ldots, x_n)$ and $\hat{f}(x_1, \ldots, x_n)$ don't reveal any private information about honest clients' inputs. For example, if there is a single honest client in an epoch, the output can trivially leak the client's data. Noising the aggregate statistic to provide differential privacy [56], [86] (§7) gives some protection in this case. To limit the number of malicious clients, as in prior works [25], [26], [43], [86], we assume that the servers employ Sybil-protection mechanisms [8], [9], [50], [100].

3. Silently Verifiable Proofs

A silently verifiable proof system is a new type of zero-knowledge proof system on *distributed* data that allows a set of verifiers to check an arbitrarily large batch of proofs, from independent provers, with verifier-to-verifier communication cost constant in the batch size.

We first recall the notion of distributed data [25] and the definition of a zero-knowledge proof on distributed data [25].

Distributed data [25]. A data item $x \in \mathbb{F}^n$ is distributed between a set of v parties (e.g., verifiers) if the i-th party holds a *piece* $x_i \in \mathbb{F}^{n_i}$ of x, where $x = (x_1 \| x_2 \| \ldots \| x_v)$,

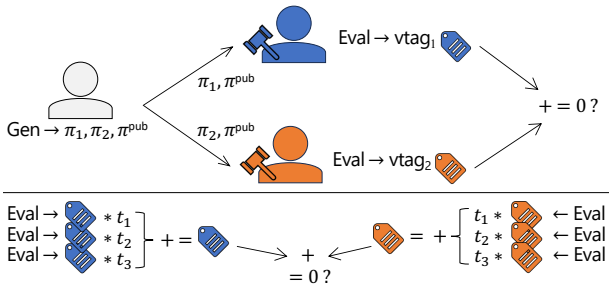

Figure 3: Silently verifiable proofs with batch verification.

and we use "‖" to denote concatenation. Secret-shared data is a special case of distributed data. For example, when x is additively secret-shared, it holds that $x = \sum_{i=1}^{v} x_i$ where $x \in \mathbb{F}^n$ and, for all $i \in [v]$, $x_i \in \mathbb{F}^n$.

Zero-knowledge proof on distributed data [25]. Such a proof system is a protocol that takes place between:

- a *prover*, holding an input $x \in \mathbb{F}^n$, for a finite field $\mathbb{F}$ and prover input size n,
- v *verifiers*, where the i-th verifier holds a piece $x_i \in \mathbb{F}^{\hat{n}}$ of the input x, for verifier input size $\hat{n}$.

The protocol allows the prover to convince the verifiers that the input $x = (x_1 \| \cdots \| x_v)$ satisfies a public predicate—i.e., that the input x is in some language $\mathcal{L} \subseteq \mathbb{F}^n$—while revealing nothing about the input x apart from the pieces of the input x that they already hold and the fact that $x \in \mathcal{L}$.

We consider a flavor of zero-knowledge proofs on distributed data that has a very simple communication pattern:

1) the prover sends each verifier a single message,
2) the verifiers each broadcast a single message to the other verifiers, and
3) each verifier runs some computation on the received messages to determine whether to accept or reject.

Many existing proof systems have this structure [25], [43], [48]. In practice, a designated verifier receives the messages from all verifiers and decides to accept or reject the proof.

A *silently verifiable proof* is a special zero-knowledge proof on distributed data where the verifiers' decision to accept or reject the proof is a *linear* function of the broadcasted messages. As we discuss in §3.2, silently verifiable proofs allow verifiers to check a large batch of proofs at once, with minimal verifier-to-verifier communication.

3.1. Definition

We define silently verifiable proof systems in the information-theoretic setting, i.e., we require the proof systems to be secure against computationally unbounded prover and verifiers. Later on, we will consider computationally-secure variants of these proof systems—in that setting, we consider infinite families of languages $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda=1}^{\infty}$, we require all algorithms to run in time $\text{poly}(\lambda)$, and we prove security against adversaries that run in time $\text{poly}(\lambda)$.

Our definition of zero-knowledge proofs on distributed data closely follows those in prior work [25], [43], [48]. The key differences are:

1) our proofs have a "public part" that the prover sends to all verifiers, in addition to a per-verifier "secret part," and
2) we only consider non-interactive proof systems—in which the prover sends a single message to each verifier.

Notation. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. For strings x and y, $x \| y$ denotes their concatenation.

Syntax: Zero-knowledge proof on distributed data. For a finite field $\mathbb{F}$, prover input size n, verifier input size $\hat{n}$, tag size q, and language $\mathcal{L} \subset \mathbb{F}^n$, a v-verifier zero-knowledge proof system on distributed data consists of the following algorithms:

$\mathsf{Gen}(x_1, \ldots x_v) \to (\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}})$. Randomized procedure takes as input $x_i \in \mathbb{F}^{\hat{n}}$ for each verifier $i \in [v]$ and outputs v private proofs π_i and one public proof π^{pub}.

$\mathsf{Eval}(x_i, \pi_i, \pi^{\mathsf{pub}}; r) \to \mathsf{vtag}_i \in \mathbb{F}^q$. Takes as input the i-th verifier input x_i, private proof π_i, the public proof π^{pub}, and a random tape r. Returns a proof tag vtag_i of size q.

$\mathsf{Ver}(\mathsf{vtag}_1, \ldots, \mathsf{vtag}_v) \in \mathbb{F}$. Takes as input the v verification tags and checks whether to accept or reject the proof. By convention, output $0 \in \mathbb{F}$ indicates acceptance.

Silent verification. We say that the proof system is *silently verifiable* if the verification predicate Ver computes a *linear* function (over field $\mathbb{F}$) of the verification tags it takes as input. The tag size q is one and Ver checks that the (scaled) verification tags sum to zero; both follow from the linearity of the verification predicate.

A zero-knowledge proof system on distributed data—whether silently verifiable or not—must satisfy the following completeness, soundness, and zero-knowledge properties.

Completeness. Completeness states that verification will always succeed if $x \in \mathcal{L}$ and all the parties are honest. Formally, we say that a v-verifier silently verifiable proof system for a language $\mathcal{L}$ has *completeness* if, for all input shares $x_1, \ldots, x_v$ such that $x = (x_1 || x_2 || \ldots || x_v) \in \mathcal{L}$ and for all choices of the verifiers' randomness r, if we compute

$$\begin{aligned}(\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}}) &\leftarrow \mathsf{Gen}(x_1, \ldots, x_v) \\ \mathsf{vtag}_i &\leftarrow \mathsf{Eval}(x_i, \pi_i, \pi^{\mathsf{pub}}; r), \text{ for } i \in [v],\end{aligned}$$

we have $\mathsf{Ver}(\mathsf{vtag}_1, \ldots, \mathsf{vtag}_v) = 0$.

Soundness. Soundness states that a prover trying to prove that x is in the language $\mathcal{L}$ for $x \notin \mathcal{L}$ will fail the verification at the v honest verifiers. Formally, we say that a v-verifier silently verifiable proof system for a language $\mathcal{L}$ has *soundness error* ϵ if, for all adversaries $\mathcal{A}$, it holds that:

$$\Pr_r \left[\begin{array}{c} \mathsf{Ver}(\mathsf{vtag}_1^*, \ldots, \mathsf{vtag}_v^*) = 0 \text{ and } (x_1^* || \ldots || x_v^*) \notin \mathcal{L}: \\ (x_1^*, \ldots, x_v^*, \pi_1^*, \ldots, \pi_v^*, \pi^{\mathsf{pub}^*}) \leftarrow \mathcal{A}() \\ \mathsf{vtag}_i^* \leftarrow \mathsf{Eval}(x_i^*, \pi_i^*, \pi^{\mathsf{pub}^*}; r), \text{ for all } i \in [v] \end{array} \right] \leqslant \epsilon.$$

In the information-theoretic setting, the adversary $\mathcal{A}$ can run in unbounded time. When we use the random-oracle model [17], all algorithms have access to a common random oracle; soundness holds only against adversaries making a bounded number of random-oracle queries.

Zero knowledge. Any strict subset of the verifiers does not learn any information about the prover's private input x other than what can be inferred from the pieces they hold and the fact that $x \in \mathcal{L}$. In our private analytics use case where the input is additively secret-shared, this property guarantees that the client leaks no information about its private data to an adversarial coalition of up to $v - 1$ out of the v servers.

Formally, we say that a v-verifier silently verifiable proof system has *δ-statistical zero knowledge* if, for every strict subset $A \subset [v]$, there exists a simulator $\mathcal{S}$ such that for all $x_1, \ldots, x_v \in \mathbb{F}^{\hat{n}}$, where $(x_1 \| x_2 \| \ldots \| x_v) \in \mathcal{L}$, the following distributions are δ-close in statistical distance, where ρ is an upper bound on the number of bits of randomness that Eval uses:

$$\mathcal{D}_{\text{real}} = \left\{ \begin{array}{l} \{\mathsf{view}_i\}_{i \in A}: \\ \quad (\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}}) \leftarrow \mathsf{Gen}(x_1, \ldots, x_v) \\ \quad r \xleftarrow{R} \{0,1\}^\rho \\ \quad \mathsf{vtag}_i \leftarrow \mathsf{Eval}(x_i, \pi_i, \pi^{\mathsf{pub}}; r), \text{for } i \in A \\ \quad \mathsf{view}_i \leftarrow \begin{pmatrix} x_i, \pi_i, \pi^{\mathsf{pub}}, r \\ \mathsf{vtag}_1, \ldots, \mathsf{vtag}_v \end{pmatrix}, \text{for } i \in A \end{array} \right\}$$

and $\mathcal{D}_{\text{ideal}} = \mathcal{S}(\{x_i\}_{i \in A})$.

If a proof system has δ-statistical zero knowledge with $\delta = 0$, we say that it has *perfect* zero knowledge. In the computational setting, we require that the simulator runs in probabilistic polynomial time, in the security parameter. Furthermore, we can relax the definition to allow the simulation to be computational, rather than statistical.

Efficiency metrics. The most important efficiency metric in a silently verifiable proof system is the *proof size*—the number of bits output by Gen. It dictates the number of bits the prover must send to the verifiers during an interaction.

Generalizations. Following prior work [25], we can generalize the definition of silently verifiable proof systems:

- we can require completeness and/or soundness to hold when only a subset of the verifiers is honest,
- we can require zero knowledge to hold when x is secret-shared and the simulator is given no explicit input.

Moreover, we can further generalize the definition by:

- relaxing the notion of silent verification to consider proof systems on distributed data where the verifier-to-verifier communication to check a batch of proofs grows sublinearly (not just constant) in the batch size.
- considering non-linear verification predicate Ver.
- considering asymmetric input sizes, or asymmetric tag sizes (especially when Ver is not linear) for the verifiers.
- considering verification over point-to-point channels instead of broadcast.

Since our simpler definition suffices for our application, we defer these generalizations to future work.

3.2. Features of silently verifiable proofs

We now mention two useful properties of our proofs:

Batch checking. A set of verifiers can check an arbitrarily large batch of silently verifiable proofs at the same communication cost as checking a single proof. Recall that, to verify a silently verifiable proof, the verifiers

- each compute a verification tag from their input, and
- check that their verification tags sum to zero.

To verify a batch of B proofs, the verifiers compute the verification tags for each of the B proofs as before. Rather than broadcasting the verification tags for each proof separately, the verifiers can agree on a shared random test vector $t \in \mathbb{F}^B$. Each verifier i publishes the inner product of their B verification tags (as a vector in $\mathbb{F}^B$) with the shared random vector t (Figure 3). If any set of verification tags in the batch sums to a non-zero value, then the combined verification tag will be non-zero with probability at least $1 - \frac{1}{|\mathbb{F}|}$.

Zero-knowledge against malicious verifiers. By definition, silently verifiable proof systems provide zero-knowledge even if a subset of the verifiers is malicious. This strong privacy guarantee comes for free because each verifier just sends a single message. Given an honest prover, the messages sent by honest verifiers are independent of the error introduced by malicious verifiers in their messages. Therefore, malicious verifiers learn no additional information about the prover's private input by deviating from the protocol.

3.3. General construction: silently verifiable proofs

We now show how to construct silently verifiable proof systems. We only focus on the case where the input is additively secret-shared between the verifiers, and not otherwise arbitrarily distributed. We refer to the zero-knowledge proof systems in this setting as *zero-knowledge proofs on secret-shared data*. This suffices for the applications that we consider in this work.

Our strategy is to start with a conventional zero-knowledge proof system on secret-shared data that is *not* silent—that is, the verification predicate may compute a non-linear function [25], [43], [48]. Then we show how to convert any non-silent proof into a silent one in the random-oracle model [17].

Batching via deterministic verification. At a high-level, batched verification can be achieved given a (non-silent) zero-knowledge proof system on secret-shared data for which the verification is *deterministic*. When the verification is deterministic, the prover can locally compute the verification transcript and send it to the verifiers. The verifiers can then do a simple consistency check on the transcript to verify the proof, and the check can be batched to verify multiple transcripts (one for each proof to be verified) at once.

An approach to get deterministic verification is to build zero-knowledge proofs on secret-shared data using the "MPC-in-the-head" paradigm [71], [75]. To prove that some $x \in \mathcal{L}$, the high-level idea is for the prover to run an MPC protocol (a.k.a. *base* protocol) that computes the predicate "$x \in \mathcal{L}$" in its head—meaning, playing the role of each party in the MPC. The transcript from this emulated execution is then used to prove the validity of the statement. The recent theoretical work of Hazay et al. [71] adapts this paradigm to build batch-verifiable zero-knowledge proofs on secret-shared data from an MPC protocol that computes the predicate in question. However, their proofs assume that a majority of the verifiers are honest.

Silently verifiable proofs, on the other hand, need to guarantee zero-knowledge even when a majority of the verifiers are dishonest. In this setting, we consider efficient and specialized base protocols [25], [27], [43] where the verifiers need to sample a random challenge after the prover commits to the proof. Therefore, to make the verification deterministic, a transform similar to the Fiat-Shamir transform [25], [62], [104] is required alongside a tailored application of MPC-in-the-head.

Delegating verification via MPC-in-the-head and Fiat-Shamir. Figure 4 shows our construction, which we now explain. It starts with a non-silent zero-knowledge proof system on secret-shared data. Our idea is to have the prover run the non-silent proof system, playing the role both of the prover and of all v verifiers in that proof system.

In our construction, rather than delegating the MPC protocol to the prover, we delegate the verification protocol of non-silent zero-knowledge proofs on secret-shared data. This not only makes silent verification possible but also leads to shorter proofs compared to the conventional use of MPC-in-the-head [71].

In our setting, the prover simulates the process of:

- the non-silent prover sending a proof to each verifier,
- the verifiers sampling random coins and using these to generate verification tags, and
- each verifier broadcasting a verification tag.

In the second step, the prover uses a Fiat-Shamir-like transformation [25], [62], [104] to derive the randomness that the (simulated) verifiers use to check the non-silent proofs.

For $i \in [v]$, the prover writes down the view of the i-th verifier as the private proof π_i. The prover writes down all of the public messages (i.e., the non-silent public proof and broadcasted verification tags) as the public proof π^{pub}.

Upon receipt of the proof, each verifier first checks that the simulated view of the protocol, contained in their private proof π_i, is correct. Then, the verifiers must check that the simulated broadcast messages in the public proof π^{pub} are consistent with their simulated local views. The verifiers do this by taking an appropriate random linear combination of the elements in their simulated views and public proofs.

Putting all of this together, we have:

Theorem 3.1 (Silently verifiable proofs)**.** *The construction of Figure 4, when instantiated with a zero-knowledge proof on secret-shared data for a language $\mathcal{L}$ with soundness error ϵ, produces a silently verifiable proof for $\mathcal{L}$ in the random-oracle model with soundness error $\epsilon' = \frac{vT^2+1}{|\mathbb{F}|} + (\epsilon +$*

Proof system	Prover to verifier	Verifier to verifier	
		All good	d bad
Non-silent	$\|\pi\|$	pq	pq
Silent	$\|\pi\| + v + q$	1	$d \log_2 \frac{p}{d}$

TABLE 1: Communication in field elements for silently verifiable proofs (Figure 4) and the underlying non-silent proof system. There are p provers and a small set of v verifiers. The non-silent proof system has tag size q. Entries represent the comm. from each prover to each verifier, and verifier to verifier comm. to verify the batch of p proofs. The proofs are either all honestly generated or d out of p are malicious. $O(\cdot)$ notation is suppressed for readability.

$\frac{1}{|\mathbb{F}|})\frac{T}{\Delta}$ *against adversaries making at most* T *random-oracle queries, where* $\Delta = (1 - \frac{T}{|\mathbb{F}|})^v$. *This transformation preserves the zero-knowledge property of the underlying proof system.*

We prove this theorem in §A.1. The soundness error ϵ' simplifies to $\approx \frac{vT^2}{|\mathbb{F}|} + \epsilon T$ when the adversary's running time $T \ll |\mathbb{F}|$. When T is $\mathsf{poly}(\lambda)$, $|\mathbb{F}| \approx 2^\lambda$, and $\epsilon = \mathsf{negl}(\lambda)$, the soundness error ϵ' of the silently verifiable proof stays $\mathsf{negl}(\lambda)$. Instantiating this template with standard zero-knowledge proofs on secret-shared data [25], we have:

Corollary 3.2 (Arithmetic circuit satisfiability)**.** *For an arithmetic circuit* $C\colon \mathbb{F}^n \to \mathbb{F}$, *let* $\mathcal{L}_C$ *be the language of vectors in* $\mathbb{F}^n$ *such that* $C(x) = 0 \in \mathbb{F}$. *Let* $M \geqslant 1$ *denote the number of multiplication gates in* C. *Then there exists a* v*-verifier silently verifiable proof for* $\mathcal{L}_C$ *in the random oracle model, with soundness error* $\epsilon \leqslant \frac{vT^2+2^vT+1}{|\mathbb{F}|} + \frac{2^{v+1}MT}{|\mathbb{F}|-M}$ *against adversaries making at most* $T < |\mathbb{F}|/2$ *random-oracle queries, perfect zero-knowledge, and proof size* $2M + 5v + 3$ *field elements to each verifier.*

Shorter public proof. In existing zero-knowledge proofs on secret-shared data [25], [43], [48], the $\mathsf{Ver}(\cdot)$ algorithm uses the sum of its inputs to compute its output, i.e., $\mathsf{Ver}(\mathsf{vtag}_1, \ldots, \mathsf{vtag}_v) = \mathsf{Ver}_{\mathsf{Inner}}(\mathsf{vtag})$, where $\mathsf{Ver}_{\mathsf{Inner}}(\cdot)$ is the main decision algorithm and $\mathsf{vtag} = \mathsf{vtag}_1 + \cdots + \mathsf{vtag}_v \in \mathbb{F}^q$. We use this structure to shrink the public proof size from Figure 4 as follows. The public proof π^{pub} includes the sum $\mathsf{vtag}_{\mathsf{NS},1} + \cdots + \mathsf{vtag}_{\mathsf{NS},v}$ of non-silent verification tags rather than including all tags separately and all the verifier checks stay linear as before. This reduces the proof size from Corollary 3.2 to $2M + v + 7$ field elements. Table 1 shows our proof size and verification communication.

3.4. Extensions: Sublinear proof size

Let $\mathcal{L}$ be the language of vectors that satisfy an arithmetic circuit with M multiplication gates. Our proofs so far (Corollary 3.2) for the language $\mathcal{L}$, have proof size linear in M. We now show how, when the language $\mathcal{L}$ is more structured, we can make the proof size sublinear in M.

Repeated instances of a subcircuit. If the circuit is made up of just affine combinations of M_G instances of a single subcircuit $G : \mathbb{F}^L \to \mathbb{F}$ of algebraic degree d, then instantiating Theorem 3.1 with prior zero-knowledge proofs on secret-shared data [25], we can achieve the proof size $2L + dM_G + v + 3$ with soundness error $\approx (vT + dM_G)\frac{T}{|\mathbb{F}|}$ (when $M_G, T \ll |\mathbb{F}|$) and zero-knowledge parameter $\delta = 0$.

Constant-degree languages. Languages with a constant degree (typically $d = 2$) define the valid submissions for many statistics like (vector) SUM, MEAN, VARIANCE and FREQUENCY COUNT. For these languages, prior work [25] constructs and implements [51] zero-knowledge proofs on secret-shared data with proof size $O(\sqrt{M})$. Using these non-silent proof systems, we can directly generate silently verifiable proofs with proof size $O(\sqrt{M})$ via Theorem 3.1. Moreover, $O(\log M)$-round interactive proofs from prior work [25] achieve proof size $\tilde{O}(\log M)$, and can be compiled to non-interactive proofs (although with a soundness loss [12], [19], [25]). Using these non-interactive proofs, we get silently verifiable proofs with $\tilde{O}(\log M)$ proof size.

Language of vectors of Hamming-weight one. When computing the FREQUENCY COUNT and APPROXIMATE FREQUENCY statistics [43], and sketching for heavy hitters (§5), each client must prove to the servers that it has secret-shared a vector of Hamming-weight one. Prior work on arithmetic-sketching schemes [26]–[28] gives protocols for this with constant server-to-server communication. We can compile them into a silently verifiable proof following Figure 4.

4. Collecting Aggregate Statistics

4.1. Preliminaries: Additive encodings

We recall additive encodings, as used in Prio [43] and other private-aggregation systems [26], [29], [48], [57], [74], [79], [81], [86], [97]. For an input space $\mathcal{X}$, an output space $\mathcal{Y}$, and number of inputs n, let $f\colon \mathcal{X}^n \to \mathcal{Y}$ be an aggregation function. For a finite field $\mathbb{F}$, encoding length ℓ, and a function $\hat{f} : \mathcal{X}^n \to \{0,1\}^*$, a private additive encoding for f with leakage $\hat{f}$ consists of three efficient algorithms:

- **Encoder** $E(x) \to e$. Outputs an encoding $e \in \mathbb{F}^\ell$ of input $x \in \mathcal{X}$.
- **Verifier** $V(e) \to \{0,1\}$. Verifies an encoding $e \in \mathbb{F}^\ell$.
- **Decoder** $D(e) \to y$. Outputs the decoding $y \in \mathcal{Y}$ of its input $e \in \mathbb{F}^\ell$.

These algorithms must satisfy the following properties:

- *Completeness*: For all $x_1, \ldots, x_n \in \mathcal{X}$ and for all $i \in [n]$, then it holds that $V(E(x_i)) = 1$ and $D(E(x_1) + \cdots + E(x_n)) = f(x_1, \ldots, x_n)$.
- *Soundness*: If $V(e) = 1$, then there exists $x \in \mathcal{X}$ such that $E(x) = e$. Moreover, there is an efficient algorithm that computes such an x.
- *Privacy*: There exists an efficient simulator $\mathcal{S}$ such that for all $x_1, \ldots, x_n \in \mathcal{X}$, $\mathcal{S}\big(f(x_1, \ldots, x_n), \hat{f}(x_1, \ldots, x_n)\big)$ outputs an identical distribution as $E(x_1) + \cdots + E(x_n)$.

Silently verifiable proofs. Construction of silently verifiable proofs over the finite field $\mathbb{F}$ with input size n for the language $\mathcal{L} \subseteq \mathbb{F}^n$ with v verifiers from a (non-silent) proof system $(\mathsf{Gen}_{\mathsf{NS}}, \mathsf{Eval}_{\mathsf{NS}}, \mathsf{Ver}_{\mathsf{NS}})$. The construction uses a hash function $H\colon \{0,1\}^* \to \{0,1\}^\rho$, which we model as a random oracle, where ρ random bits are used by $\mathsf{Eval}_{\mathsf{NS}}$.

Proof generation.
$\mathsf{Gen}(x_1, \ldots, x_v) \to (\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}})$:

- Generate the non-silent proof:

$$(\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}}_{\mathsf{NS}}) \leftarrow \mathsf{Gen}_{\mathsf{NS}}(x_1, \ldots, x_v)$$

- Use multiparty Fiat-Shamir [25] to derive the randomness r_{NS} that the verifiers will use to verify the proof:

$$c_i \leftarrow H(i, x_i, \pi_i, \pi^{\mathsf{pub}}_{\mathsf{NS}}), \text{ for } i \in [v]$$
$$r_{\mathsf{NS}} \leftarrow H(c_1, \ldots, c_v).$$

- Derive the verification tags that the verifiers would broadcast to verify the non-silent proof with randomness r_{NS}. That is, for $i \in [v]$, set

$$\mathsf{vtag}_{\mathsf{NS},i} \leftarrow \mathsf{Eval}_{\mathsf{NS}}(x_i, \pi_i, \pi^{\mathsf{pub}}_{\mathsf{NS}}; r_{\mathsf{NS}}) \in \mathbb{F}^q.$$

  Then compute:

$$\pi^{\mathsf{pub}} \leftarrow (\pi^{\mathsf{pub}}_{\mathsf{NS}}, c_1, \ldots, c_v, \mathsf{vtag}_{\mathsf{NS},1}, \ldots, \mathsf{vtag}_{\mathsf{NS},v}).$$

- Output $(\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}})$.

Proof evaluation.
$\mathsf{Eval}(x_i, \pi_i, \pi^{\mathsf{pub}}; r) \to \mathsf{vtag}_i$:

- Parse $\pi^{\mathsf{pub}} \to (\pi^{\mathsf{pub}}_{\mathsf{NS}}, c^*_1, \ldots, c^*_v, \mathsf{vtag}^*_{\mathsf{NS},1}, \ldots, \mathsf{vtag}^*_{\mathsf{NS},v})$.
- Rederive the randomness r^*_{NS} that the non-silent verifiers would have used to verify the proof:

$$r^*_{\mathsf{NS}} \leftarrow H(c^*_1, \ldots, c^*_v).$$

- Derive the verification tags that the verifiers would broadcast to verify the non-silent proof with randomness r^*_{NS}. That is, set

$$\mathsf{vtag}_{\mathsf{NS},i} \leftarrow \mathsf{Eval}_{\mathsf{NS}}(x_i, \pi_i, \pi^{\mathsf{pub}}_{\mathsf{NS}}; r^*_{\mathsf{NS}}) \in \mathbb{F}^q$$

- The verifiers must check that:
  - the prover computed the challenge value correctly: $c^*_i = H(i, x_i, \pi_i, \pi^{\mathsf{pub}}_{\mathsf{NS}})$,
  - all verifiers' π^{pub} values are identical,
  - the prover provided correct verification tags, i.e., verifier i checks that $\mathsf{vtag}^*_{\mathsf{NS},i} = \mathsf{vtag}_{\mathsf{NS},i}$, and
  - verification of the non-silent proof succeeds: $\mathsf{Ver}_{\mathsf{NS}}(\mathsf{vtag}^*_{\mathsf{NS},1}, \ldots, \mathsf{vtag}^*_{\mathsf{NS},v}) = 0$.

  Each verifier i uses the shared randomness r to form a value $\mathsf{vtag}_i \in \mathbb{F}$ such that $\mathsf{vtag}_1 + \cdots + \mathsf{vtag}_v = 0 \in \mathbb{F}$ if and only if these checks all succeed, with high probability over the choice of r.
- Return $\mathsf{vtag}_i \in \mathbb{F}$.

Figure 4: General construction of silently verifiable proofs.

For example, there is a folklore additive encoding for VARIANCE. Let $\mathbb{F} = \{0, 1, 2, 3, \ldots\}$ be a finite field and let n be the number of inputs. The input space of the encoding is $\mathcal{X} \in \{0, \ldots, B\} \subseteq \mathbb{F}$, for some bound B where $\sqrt{B} < n|\mathbb{F}|$. The output space of the encoding is $\mathcal{Y} = \mathbb{Q}$. The encoding routine outputs $E(x) = (x, x^2) \in \mathbb{F}^2$, so the encoding length is $\ell = 2$. The verification routine $V(a, b)$ accepts if $0 \leqslant a, b \leqslant B$ and $a^2 = b \in \mathbb{F}$. The decoding routine outputs $D(a, b) = (\frac{b}{n} - (\frac{a}{n})^2) \in \mathbb{Q}$.

Recent work proposes powerful theoretical generalizations of additive encodings [68]. These encodings do not natively provide verifiability—taking advantage of these more recent encodings is an exciting direction for future work.

4.2. Private-aggregation scheme

We now use silently verifiable proofs to construct a private-aggregation scheme where servers communicate and store sublinear in the number of clients (in optimistic case).

The protocol is essentially that of prior work [3], [43], [45], [48], [50], [74], [81], [86], except with silently verifiable proofs. Prior work formalizes its security properties [43], [48], [50], [74], so we only sketch them here. In the following, we say that a client's submission is "valid" if there exists an input $x \in \mathcal{X}$ such that an *honest* client produces the same submission on input x. The protocol is defined relative to a public aggregation function $f\colon \mathcal{X}^* \to \mathcal{Y}$.

- *Correctness against malicious clients*: If all servers are honest, then the servers output f applied to the inputs of clients whose submissions were valid.
- *Privacy*: As long as one server is honest, the protocol only reveals the value of the aggregation function f and leakage function $\hat{f}$ applied to the inputs of clients whose submissions were valid.

Building blocks. The private-aggregation protocol with n clients and v servers for the function f works over a finite field $\mathbb{F}$ and requires two building blocks:

- A private additive encoding (E, V, D) over $\mathbb{F}$ with input space $\mathcal{X}$, output space $\mathcal{Y}$ and encoding length ℓ for the aggregation function f with leakage $\hat{f}$.
- A silently verifiable proof system $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Ver})$ over $\mathbb{F}$ for the language $\mathcal{L} = \{e \mid V(e) = 1 \text{ and } e \in \mathbb{F}^\ell\}$ with v verifiers, where ℓ is the encoding length and V is the additive-encoding verifier.

Protocol. At a high level, the protocol proceeds as follows: Each client $i \in [n]$ performs the following steps:

- On input x_i, generate an additive encoding $e \leftarrow E(x_i) \in \mathbb{F}^\ell$ of the input. Split the encoding into v additive shares: $e = e_1 + \cdots + e_v \in \mathbb{F}^\ell$.
- Generate a silently verifiable proof that the encoding is well formed: $(\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}}) \leftarrow \mathsf{Gen}(e_1, \ldots, e_v)$.
- For $j \in [v]$, send (e_j, π_j) to server j and π^{pub} to all.

Next, each server $j \in [v]$ performs the following steps:

- For each client $i \in [n]$, generate a verification tag $\mathsf{vtag}_i \in \mathbb{F}$ to verify that client i's submission is valid.
- Take a random linear combination (using randomness shared across all servers) of the n verification tags (one per client) to construct a batched verification tag $\mathsf{vtag}^\star_j \in \mathbb{F}$. Send this tag to the first server.

Finally, the servers perform the following steps:

- The first server checks that $\sum_{j \in [v]} \mathsf{vtag}^\star_j = 0 \in \mathbb{F}$ and broadcasts the result to all the other servers.
- If the check fails, the servers jointly employ group testing (§4.3) to identify the failing proofs and weed out the malformed inputs.
- Each server $j \in [v]$ adds its shares of each (valid) encoding to generate a share $e^\star_j \in \mathbb{F}^\ell$ of the sum of encodings. Server j sends $e^\star_j$ to the first server.
- The first server computes the sum $e^\star \leftarrow \sum_{j \in [v]} e^\star_j \in \mathbb{F}^\ell$ and computes the final output $\mathsf{out} \leftarrow D(e^\star) \in \mathcal{Y}$, i.e., the aggregate statistic over the clients' secret inputs.

We argue in Appendix B that this protocol provides correctness against malicious clients. The argument for privacy follows exactly as in prior work [43]. As mentioned in §2.2, the servers in this protocol can verify the submissions in batches of size n_b each and locally aggregate their shares of passing submissions as they go. In the end, each server $j \in [v]$ sends its $e^\star_j$ to the first server.

Efficiency. The server storage while running the protocol is essentially just a vector in $\mathbb{F}^\ell$. The server-to-server communication depends on the number of malicious clients (§4.3).

Supported statistics. Using existing additive encodings [31], [41], [43], [57], [86], [97], Whisper can compute the following aggregation functions:

- *Basic statistics:* SUM, MEAN, VARIANCE, STDDEV, MIN/MAX (over small domains)
- *Counting:* FREQUENCY COUNT, APPROXIMATE FREQUENCY
- *Boolean operations:* AND, OR
- *Machine learning:* LINEAR REGRESSION, r^2 COEFFICIENT

4.3. Finding failing proofs

In our private-aggregation protocol, when malicious clients submit invalid proofs, the servers' batch-verification check fails. To identify the failing proofs with little server-to-server communication, the servers can use standard algorithms for group testing [53], [54]; similar approach has been adopted in prior work on anonymous communication [2] and private analytics [89]. Whisper uses the binary-splitting algorithm [54], [73]: with a rough estimate on the upper bound of the number of "defective" uploads d in each batch of n_b clients, the servers first split the batch into d non-overlapping batches of n_b/d clients each and compute $\mathsf{vtag}^\star_i$ for $i \in [v]$ for each such batch. They exchange these verification tags to find which batches contain defective uploads. For each defective batch, they recursively search for defective clients within each batch in parallel. They continue recursing until they are left with defective singleton batches—these are the malicious clients. This requires $1 + \log \frac{n_b}{d}$ rounds of server interaction and $O(vd \log \frac{n_b}{d})$ field elements in total communication per batch of n_b clients.

This strategy only works if the servers split each batch into consistent sub-batches even though some uploads could be in some servers' batches but not in others. We do not want extra server communication to reach a consensus on these sub-batches. Instead, during setup, the servers share a key for a pseudorandom function and use it to map each client's id (from §2.2) to a random and deterministic sub-batch.

In our approach, the additional server overhead to identify failing proofs allows malicious clients to degrade the system's performance. However, we show in our evaluation (§6) that the servers remain efficient and available (unlike a DDoS attack) even in the presence of a large number of malicious clients. Moreover, we discuss a strategy in §7 to permanently ban malicious clients and get rid of the overhead of repeatedly identifying the same offenders.

5. Sketching for heavy hitters

The heavy-hitters aggregate statistic takes as input a set of n strings, each L bits long. It returns the set of strings that appear more than a certain number of times in the input. Prior work [26], [89] has proposed custom protocols for efficient computation of *exact* heavy hitters. A limitation of these protocols is that they require $O(L)$ rounds and they do not support streaming computation (i.e., the servers must store and repeatedly compute over all client submissions).

In this section, we consider the relaxed problem of computing *approximate* heavy hitters—we tolerate a small probability of failure in outputting the heavy hitters. The benefit is that we get a streaming-friendly protocol with round complexity constant in the string length L.

Our approach, following prior work on private aggregation [86], is to use linear sketches [34], [40], [41], [82], [96]. For our purposes, a linear sketch is just an additive encoding (in the sense of §4.1) for the *approximate* heavy-hitters function. Prior private-aggregation schemes using sketching for heavy hitters [86] require the servers to run in time 2^L on string length L. Instead, we identify as fitting a more advanced sketching construction due to Pagh et al. [96], that reduces the servers' work to polynomial in the string length L. (We could also have used the count-min sketch with the "dyadic trick" [41]. The sketch of Pagh et al. is better for our application, since the clients can compress their secret-shared submissions to servers using distributed point functions [28], [49].) We compare with

the approach of computing approximate heavy hitters using Local Differential Privacy in §8.

Notation. In this section, all arithmetic happens over a finite field $\mathbb{F}$ with size $|\mathbb{F}|$, which we assume to be $\mathbb{Z}_p$ for a prime modulus p. Positive elements are $\{1, \ldots, \lfloor \frac{p}{2} \rfloor\}$. The value $-x$ represents the field element $p - x$. We use the total ordering $-\lfloor \frac{p}{2} \rfloor < \cdots < -1 < 0 < 1 < \cdots < \lfloor \frac{p}{2} \rfloor$.

5.1. Building block: Bucketed string counting

Our private-heavy-hitters construction uses the private-aggregation scheme of §4 as a subroutine. In particular, we instantiate that private-aggregation protocol with an aggregation function that we call "bucketed-string-counting."

The aggregation function is parameterized by a number of buckets B, number of client inputs n, and a string length L. Each client holds a pair of a bucket ID in $\{1, \ldots, B\}$ and an L-bit string σ. For each bucket $b \in [B]$, the aggregation function puts the "average" of the strings in bucket b. That is, if we view each string as a vector $\hat{\sigma} \in \{-1, 1\}^L \subseteq \mathbb{F}$, then for each bucket $b \in [B]$, the aggregation function sums up the values in each bucket.

Private-aggregation for bucketed string counting. Figure 14 of Appendix C gives a simple additive encoding (E, V, D) for bucketed string counting with encoding length $\ell = (L + 1) \cdot B$ and no leakage. For each bucket and for each bit of each string, we use one instance of the additive encoding for the SUM function. The validity predicate just ensures that the client only inserted a string into a single bucket (i.e., that there is only one bucket-aligned run of non-zero values) and that the string is encoded in $\{-1, 1\}^L$.

We use arithmetic sketching [27], [28] to construct silently verifiable proofs for the language of valid encodings (see §3.4). The encoding and the proof system then, via the private-aggregation protocol of Section 4.2, yield a private-aggregation scheme for bucketed string counting.

Optimization in the two-server case. When there are only two servers, we can use verifiable distributed point functions (VDPFs) [28], [49] to compress the secret-shared additive encodings sent by each client. A VDPF gives a succinct way to secret share a weight-one vector. The verifiability property means that two servers, each holding a purported succinct share of a weight-one vector, can tell that their shares are well formed by performing an equality check on a short string. As with our silently verifiable proofs, it is possible for the servers to batch-verify a large number of VDPFs by exchanging a short string. VDPFs thus can replace silently verifiable proofs in the two-server setting for heavy hitters.

5.2. Our heavy-hitters protocol

In our protocol (Figure 5), each client first hashes its input string $x \in \{0, 1\}^L$ into one of B buckets, where B is a protocol parameter. The client and servers then run the private-aggregation protocol for bucketed string counting to compute the "average" of the strings in each bucket.

Heavy hitters protocol. Parameters: a number of clients n, a string length L, a number of buckets B, hash functions $H_{\text{bucket}} \colon \{0, 1\}^L \to [B]$ and $H_{\text{sign}} \colon \{0, 1\}^L \to \{0, 1\}$, and a heavy-hitter threshold T.

Client input preparation. Given a string $x \in \{0, 1\}^L$ as input:

- The client hashes the string to get a bucket ID b and sign bit β:

  $$b \leftarrow H_{\text{bucket}}(x) \in [B] \quad \text{and} \quad \beta \leftarrow H_{\text{sign}}(x) \in \{0, 1\}.$$

- If $\beta = 0$, the client complements its bitstring $x \leftarrow \bar{x}$.
- The client participates in the secure-aggregation protocol for bucketed string counting using input $(b, \beta \| x) \in [B] \times \{0, 1\}^{L+1}$.

Output decoding. The output of the secure-aggregation protocol is, for each bucket, (1) the number of strings in that bucket and (2) the sum over $\mathbb{F}^{L+1}$ of all strings in that bucket. This output-decoding procedure recovers the set of approximate heavy hitters from this output.

Initialize a set $H = \emptyset$ of heavy hitters. Then, for each bucket $b \in [B]$ containing at least T strings:

- Let $s \in \mathbb{F}^{L+1}$ be the sum of the strings in bucket b.
- "Round" s to a bitstring $\hat{\sigma} \in \{0, 1\}^{L+1}$ by mapping each value in $\{-n, \ldots, 0\} \subseteq \mathbb{F}$ to 0 and all other values to 1.
- Parse $(\beta, \sigma) \leftarrow \hat{\sigma} \in \{0, 1\} \times \{0, 1\}^L$.
- If $\beta = 0$, complement the bits of σ: $\sigma \leftarrow \bar{\sigma}$.
- Add σ to the set of heavy hitters H.

Finally, output H as the set of heavy hitters.

Figure 5: Our protocol for approximate heavy hitters.

If there were no collisions—i.e., distinct strings hash to distinct buckets—the output of the bucketed string counting function would exactly give the set of all heavy hitters. However, since multiple distinct strings may fall into the same bucket, we need to recover heavy hitters despite collisions.

The delicate part of the analysis is showing that, for the purposes of finding heavy hitters, these collisions do not matter too much. If a string is a heavy hitter, it is unlikely that it will fall into a bucket containing so many non-heavy-hitters that we cannot recover the original heavy hitter. To recover a heavy hitter from a bucket, we just round each bit of the bucket's counter either up or down to determine whether the corresponding bit of the string is either 0 or 1.

5.3. Security analysis

Since our heavy-hitters protocol only uses our private-aggregation scheme (§4) as a subroutine, the security of that scheme does not imply the security of our heavy-hitters protocol. For example, the client in our heavy-hitters

Heavy-hitters correctness game. Parameters: a number of buckets B, string length L, encoding length ℓ, number of honest clients n, number of malicious clients m, heavy-hitter threshold T, and an adversary $\mathcal{A}$.

1) The adversary outputs a list of strings, each in $\{0,1\}^L$, along with a state st:
$$(\mathsf{st}, x_1, \ldots, x_n) \leftarrow \mathcal{A}()$$
corresponding to the honest parties' inputs.
2) The challenger chooses random hash functions
$$H_{\text{sign}}\colon \{0,1\}^L \to \{0,1\} \quad \text{and} \quad H_{\text{bucket}}\colon \{0,1\}^L \to [B].$$
3) The adversary outputs m encodings, each in $\mathbb{F}^\ell$:
$$(y_1, \ldots, y_m) \leftarrow \mathcal{A}^{H_{\text{sign}}, H_{\text{bucket}}}(\mathsf{st}).$$
If, for any $i \in [m]$, the encoding y_i is invalid—i.e., $V(y_i) = 0$, the experiment output is "0".
4) For each $i \in [n]$, the challenger runs the client-input-preparation step of our heavy-hitters protocol (Figure 5) with inputs $x_1, \ldots, x_n$ to get encodings $z_1, \ldots, z_n \in \mathbb{F}^\ell$.
5) The challenger computes the sum $S \in \mathbb{F}^\ell$ of all encodings:
$$S \leftarrow (y_1 + \cdots + y_m) + (z_1 + \cdots + z_n) \in \mathbb{F}^\ell.$$
6) The challenger runs the output-decoding step of our heavy-hitters protocol (Figure 5) using the sum S as input. The output-decoding procedure produces a set H of heavy hitters.
7) If there is a string $\sigma \in \{0,1\}^L$ that:
   - appears more than T times in the list $(x_1, \ldots, x_n)$ of the honest clients' inputs, and
   - does not appear in the set of heavy hitters H,

   then the challenger outputs "1".

Figure 6: Security game for our heavy-hitters protocol.

protocol (Figure 5) is supposed to hash its string into a bucket—a malicious client may hash its string using some adversarial strategy with the goal of corrupting the system's output. In addition, our security definitions in Figure 5 did not handle approximate statistics or randomized encodings.

We now prove a notion of correctness against malicious clients, and later show privacy against malicious servers.

Correctness against malicious clients. We formally define our notion of correctness against malicious clients in Figure 6. Intuitively, we would like to guarantee that the servers recover all of the heavy hitters, even in the presence of an adversary that controls some number of malicious clients and may choose the strings that all honest parties will submit. The only guarantees we have are that (a) malicious clients' must submit valid additive encodings to the bucketed-string-counting protocol and (b) the adversary must choose the honest parties' strings independently of the random hash functions used in our sketching data structures. In a real run of our protocol, our private-aggregation scheme's input-validity checks enforce restriction (a), and the fact that honest clients will choose their strings independently of the sketch's randomness enforce restriction (b).

At a high level, we say that adversary wins the game if the system's output fails to find any heavy hitter among the honest clients' inputs. As long as we can show that the probability of winning for the adversary is small, the protocol will be useful in the face of malicious clients.

We now analyze the winning probability.

Definition 5.1 (Heavy hitters: correctness against malicious clients)**.** For a finite field $\mathbb{F}$, string length L, bucket count B, number of honest clients n, number of malicious clients m, heavy-hitter threshold T and adversary $\mathcal{A}$, let $\mathsf{HHAdv}_{L,B,n,m,T}[\mathcal{A}]$ denote the probability that the experiment of Figure 6 outputs "1" when instantiated with these parameters. We say that the heavy-hitters protocol is ϵ-*correct* against malicious clients if, for all adversaries $\mathcal{A}$, $\mathsf{HHAdv}_{L,B,n,m,T}[\mathcal{A}] \leqslant \epsilon$.

We prove the following theorem in Appendix C:

Theorem 5.2 (Heavy hitters: correctness against malicious clients)**.** *For a finite field $\mathbb{F}$, string length L, bucket count B, heavy-hitter threshold T, number of honest clients n, number of malicious clients $m < T$, if:*

- *the field characteristic of $\mathbb{F}$ satisfies* $\text{char}(\mathbb{F}) > 2(n+m)$,
- *there are k distinct heavy-hitting strings among the honest parties' inputs, and*
- *$\bar{h}$ is the total number of non-heavy-hitter occurrences among the honest parties' inputs,*

then for all adversaries $\mathcal{A}$, it holds that

$$\mathsf{HHAdv}_{L,B,n,m,T}[\mathcal{A}] \leqslant \frac{k^2}{B} + \frac{k \cdot \bar{h}}{B \cdot (T-m)}.$$

Privacy against malicious servers. The only interaction that happens in our heavy-hitters scheme is one execution of the private-aggregation scheme for bucketed string counting. Therefore, all that our protocol execution leaks (provided that at least one server is honest) is the output of bucketed-string-counting function computed over all clients' inputs. The leakage function here is symmetric in its inputs—so the association of clients to strings is completely hidden. At the same time, the output of the string-counting function does leak something beyond the heavy hitters, as in prior work [26], [86]. The leakage is small in absolute terms, since the number of buckets will be far smaller than the number of clients. Even so, it would be possible to further obscure the leakage using differential-privacy (§7).

6. Evaluation

We implement Whisper in Rust on top of the `libprio-rs` library [51]. Implementations of both Whisper and the com-

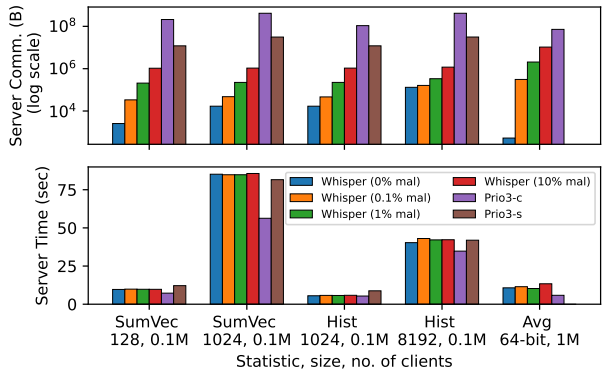

Figure 7: Server-to-server communication and time of each server for verification and aggregation of common statistics.

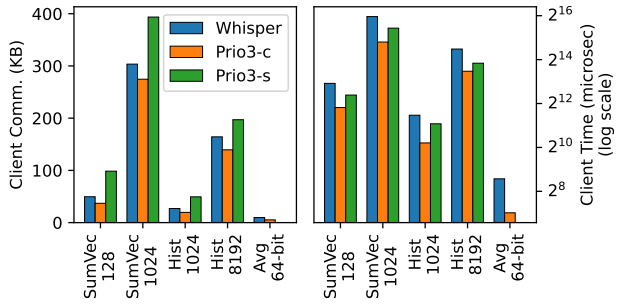

Figure 8: Communication and proof generation time per client for common statistics.

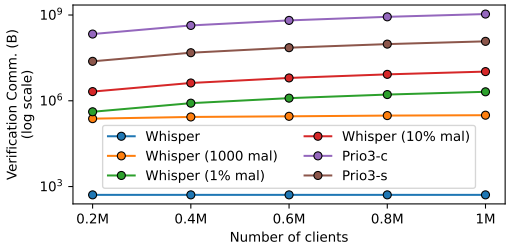

Figure 9: Verification communication per server with an increasing number of clients for Hist 1024.

parison systems are multithreaded. For heavy hitters, we implement our two-server optimization (§5) that uses VDPFs and given their compatibility with rings $\mathbb{Z}_{2^k}$ for $k \in \mathbb{N}$, our heavy hitters code runs over $\mathbb{Z}_{2^{16}}$ and $\mathbb{Z}_{2^{32}}$ (depending on the number of clients) for faster arithmetic. We use SHA-256 to batch multiple verification tags for VDPFs. Our VDPF code borrows from Poplar's codebase [42]. To be sound against adversaries that run in time at most $\approx 2^{128}$, for general statistics, we perform two parallel runs with 128-bit field each (details deferred to §D), and for heavy hitters, we set $\lambda = 128$ for VDPFs. Our code is available online at `https://github.com/ucbsky/whisper`.

Evaluation setup. We use two servers to mirror existing deployments [6], [10], [65], [90]: one in Iowa (`us-central1-a`) and the other in Virginia (`us-east4-c`). Both have 32 vCPUs and 64 GB memory. We use a MacBook Pro as a client.

6.1. General statistics

Baseline. We first compare with the state-of-the-art system Prio3 [51], [74]. For some statistics, Prio3 has a "chunk-size" parameter that trades client-to-server communication for server-to-server communication. We call the client-optimized configuration *Prio3-c* and the server-optimized configuration *Prio3-s*. We compare with both.

Statistics. We consider three main statistics supported by Prio3: VECTOR SUM ("sumvec"), FREQUENCY COUNT ("hist"), and MEAN ("avg"). For vector sums, we consider vector sizes 128 and 1024, and 16-bit entries. For frequency count, we consider 1024 and 8192 bins. Means are over 64-bit values.

Server performance. Figure 7 shows the server-to-server communication and server time (after submissions are received) for Whisper and Prio3. Increasing the number of malicious clients barely affects Whisper's server time. However, as we discuss in §4.3, finding d malicious clients requires communication $O(d \log \frac{n}{d})$, and therefore, server communication increases as the number of malicious clients increases. The server-to-server communication remains up to two orders of magnitude lower than the Prio3-c baseline. The communication-cost improvement comes at an average cost of roughly a 1.4× increase in server time. For MEAN with 10 million clients (10% malicious), server time and verification communication are 116 sec and about 10^8 bytes (linear scaling, as expected), respectively.

Client performance. Figure 8 compares Whisper with Prio3 on client communication (encoding + proof size) and client time (proof generation). Whisper has roughly 1.4× more client communication than Prio3-c. As the size of the statistics increases, the increase in our client communication relative to Prio3 goes down. Our client time is at most a few milliseconds and about 2-3× higher than baseline.

Server-optimized Prio3. Whisper improves server-to-server communication by up to 30× over Prio3-s. Whisper outperforms Prio3-s in *both* client and server communication, and the server time is comparable.

Total communication. The total communication (clients and servers combined) in Whisper is at most 2× over Prio3-c and is up to 2× better than Prio3-s when 1% clients are malicious.

Dollar cost. Using Google Cloud's pricing model [64], [66], we estimate up to 3× reduction in the cost of running the servers (about 2× reduction on average) over our baseline.

Silently verifiable proofs. Figure 9 shows batch verifiability of our silently verifiable proofs. When the number of malicious clients is fixed, verification communication stays constant as clients increase. Prio3's proof verification communication scales linearly with clients. Our batch verifica-

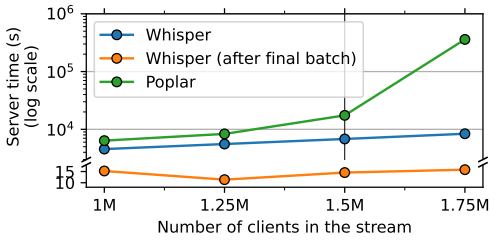

Figure 10: Server time to compute heavy hitters over a stream of client submissions for 0.1% threshold and 0.05% malicious clients. Poplar runs out of the main memory at the vertical line.

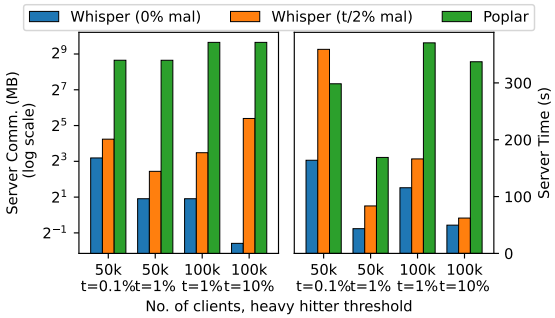

Figure 11: Server-to-server communication and time of each server to compute heavy hitters.

tion comes at some increase in proof size, proof generation time, and proof verification time (Figures 7 and 8).

6.2. Heavy hitters

Baseline. We compare with Poplar [26], the state-of-the-art system for private heavy hitters in the two-server setting.

Parameters. We sample 256-bit client inputs from a Zipf distribution with parameter 1.03 and support 10,000, as in Poplar's evaluation [26]. We configure Poplar as in their evaluation. For Whisper, we set the parameters such that the probability of finding all the heavy hitters is at least 0.999. We consider three heavy hitter thresholds 10%, 1%, and 0.1% of the total number of clients. When using the 10% threshold, we use (via Theorem 5.2 and Corollary C.1) a sketch with 32 buckets and 10 sketching instances. When using the 1% threshold, we use 256 buckets and 14 sketching instances. When using the 0.1% threshold, we use 1024 buckets and 17 sketching instances. We set the number of malicious clients as half the heavy hitter threshold, and to maintain our success probability, we double the number of buckets in our experiment with malicious clients. For our streaming experiment, we form batches of 3,000 clients.

Streaming. Figure 10 shows server runtime to process large streams with millions of clients. The computation in Poplar cannot be streamed and all submissions stay in memory. At around 1.5M clients, it exceeds the server's memory, and swapping to disk degrades its performance. Mitigating this slowdown would require using larger, more expensive servers. Whisper uses streaming to avoid this slowdown. Moreover, with the fixed batch size, Whisper's server time after the last submission is independent of the stream size. We scale Whisper to 10 million clients and observe server time of 44063 sec (linear scaling, as expected).

Other metrics (server). Whisper's server time is typically lower than Poplar (Figure 11) and server-to-server communication is up to two orders of magnitude lower (Figure 11). This translates to up to 3.8× reduction in the dollar cost to run the servers based on Google Cloud's pricing [64], [66].

Other metrics (client). Whisper's client communication is 10-17× larger than Poplar, and the client is up to 2× slower. Concretely, our client communication is less than 500 KB for all the three heavy-hitter thresholds. Moreover, historically, cloud providers don't charge for ingress communication.

Total communication. Whisper's larger client communication compared to Poplar shows up in the total communication (clients and servers combined). The total communication in Whisper is 8-13× (50k clients with $t = 0.1\%$ being the worst) more than Poplar.

7. Extensions

Eliminating bad clients. When the servers repeatedly compute aggregate statistics over the same user population [1], [65], [76], the servers may want to *permanently* exclude malicious clients from participating in this system. Before removing a client permanently from the system, the honest servers may want to ensure that the client indeed acted maliciously—i.e., it was not being framed by a malicious server. We can accomplish this additional security goal if we assume that all the servers have cryptographic public keys of each client. Each client signs the messages that it sends to the servers. For all the clients with valid signatures, the servers proceed as before. For the clients whose submissions are identified as invalid in the current and past sessions, each server can use zero-knowledge proofs [67], [103] to prove that they did the correct computation on their shares and that they have the client's signature on those shares. This step is relatively expensive, but the servers only need to do it rarely. For such a defense to be useful, the servers must employ some Sybil defense [5] to make it harder for malicious users to reenter the system by creating new accounts.

Differential privacy in Whisper. Like prior work [26], [43], [89], Whisper provides MPC-style privacy: the only information leaked about clients' inputs $(x_1, \ldots, x_n)$ is via the aggregate statistic $f(x_1, \ldots, x_n)$ that the system outputs. In some cases, this statistic itself could implicitly reveal sensitive data, e.g., via intersection attacks [20], [26], [43], [46]. To mitigate this leakage, Whisper can achieve differential privacy [56] where the servers publish noisy shares of the statistic, as in prior work [26], [43], [86]. For our heavy hitters protocol, if the ℓ_∞ norm of the total noise is bounded by Δ with high probability, then we can still recover heavy hitters that occur more than $T + \Delta$ times with

similar success probability as before (Corollary C.2). When the number of clients in the system is large, the accuracy remains high [26].

8. Related Work

Single-server model. There is a rich literature on systems for private collection of aggregate statistics via local differential privacy [7], [14], [15], [33], [86], often using sketching algorithms as Whisper does. These systems provide an incomparable privacy property to Whisper: we aim for an MPC-style privacy property—nothing leaks beyond the aggregate statistic—while these systems provide user-level differential privacy (and they do leak information about each user's data beyond the aggregate statistic itself). A secondary distinction is that these systems do not necessarily protect correctness against malicious clients [7], [14], [15], [24], [33], [39], [55], [78], [86], [107]; adding such defenses can be expensive [16], [24], [57], [81], [84]–[86], [92].

Private analytics. Another common approach in private analytics systems is to split trust across multiple non-colluding servers [3], [13], [43], [45], [57], [61], [63], [72], [74], [81], [86], [99], [102]. These protocols [13], [43], [63], [72], [74], [99], [102] offer MPC-style privacy protection. As we have discussed, to defend against malicious clients, these systems tend to require server-to-server communication linear in the number of clients. Systems relying on hardware trust [105] are vulnerable to side-channel attacks on enclaves [36], [91].

Private heavy hitters. Mix-net [35], [94] and other anonymous communication systems [44], [59], [60] can be used to compute heavy hitters from the multiset of clients' strings while providing anonymity, however, the entire multiset of the client inputs leaks in the process. In the distributed-trust setting, existing protocols incur high server-to-server communication [11], [23], [77], cannot compute heavy hitters over a stream leading to large server-side storage [89] or both [26]. Star [47] considers a different setting with an aggregation server with a separate randomness server and doesn't hide the identity of clients with the same input. Except for Plasma [89], the server egress in all these works scales linearly with the number of clients. Plasma works in a different threat model than Whisper assuming an honest majority among three servers which can be challenging to find in the real-world [83]. Moreover, Plasma and the two-server state-of-the-art Poplar [26] cannot stream heavy hitter computation leading to large server storage and require the servers to interact over multiple rounds.

Differentially private aggregate statistics. There is a long line of work [7], [14], [15], [30], [32], [33], [58], [98], [106] on computing aggregate statistics over randomized responses collected from the clients. The noise added by the clients provides differential privacy, however, it leads to a loss in the accuracy of the output and makes it challenging to filter malformed submissions [21]. Moreover, noisy submissions per client leak some information about the client's private input. Whisper and related systems [3], [26], [43], [89] provide a different privacy guarantee where only the output and a modest leakage function are seen by the servers without any information on an individual client's contribution, and the accuracy of the output is preserved. However, as mentioned in §7, when the leakage from the output is a concern, Whisper uses differential privacy where similar to [26], [37], [43], [86], the noise is added directly to the aggregate [101]. This maintains higher accuracy compared to local differential privacy. Zhu et al. [39], [107] develop a trie-based heavy hitters protocol where subsampling the clients provides meaningful differential privacy without requiring additional noise. Prochlo [22] requires a trusted shuffler.

Batch verifiable proofs on secret-shared data. Zero-knowledge proofs on secret-shared data supporting batch verification are implicit in recent work by Hazay et al. [71] where the proof sizes are at least linear in the size of the predicate. Our silently verifiable proofs provide batch verification with sublinear-sized proofs for structured languages common in private analytics. For the language of one-hot vectors, verifiable distributed point functions [49] offer batch verification and succinct key sizes.

Acknowledgements

We thank the anonymous reviewers, the shepherd, and Yuval Ishai for their helpful feedback. We thank Matei Zaharia for references on discretized streams, and Jelani Nelson for references on sketching schemes. We also thank students in the Sky security group for feedback that improved the presentation of this paper. This work is supported by NSF CAREER 1943347, and gifts from Accenture, AMD, Anyscale, Capital One, Facebook, Google, IBM, Intel, Mohamed bin Zayed University of Artificial Intelligence, Mozilla, Samsung, SAP, and VMware. This work was funded in part by the FinTech@CSAIL Initiative and NSF grant CNS-2054869.

References

[1] "Mozilla Telemetry Data Documentation: Raw Ping Data, Ping Types," https://docs.telemetry.mozilla.org/datasets/pings.html.

[2] I. Abraham, B. Pinkas, and A. Yanai, "Blinder: MPC based scalable and robust anonymous committed broadcast," *IACR Cryptol. ePrint Arch.*, p. 248, 2020.

[3] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou, "Prio+: Privacy preserving aggregate statistics via boolean shares," in *SCN*, 2022.

[4] M. Ajtai, V. Braverman, T. S. Jayram, S. Silwal, A. Sun, D. P. Woodruff, and S. Zhou, "The white-box adversarial data stream model," in *PODS*, 2022.

[5] Apple, "Assessing fraud risk request and analyze risk data using server-to-server calls," "https://developer.apple.com/documentation/devicecheck/assessing_fraud_risk".

[6] ——, "Learning iconic scenes with differential privacy," https://machinelearning.apple.com/research/scenes-differential-privacy.

[7] ——, "Learning with privacy at scale," "https://docs-assets.developer.apple.com/ml-research/papers/learning-with-privacy-at-scale.pdf".

[8] ——, "Managed device attestation for apple devices," https://support.apple.com/guide/deployment/managed-device-attestation-dep28afbde6a/web.

[9] ——, "Mitigate fraud with AppAttest and DeviceCheck," *WWDC21*, 2021.

[10] Apple and Google, "Exposure notification privacy-preserving analytics (enpa) white paper," https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf.

[11] G. Asharov, K. Hamada, D. Ikarashi, R. Kikuchi, A. Nof, B. Pinkas, K. Takahashi, and J. Tomida, "Efficient secure three-party sorting with applications to data analysis and heavy hitters," in *CCS*, 2022.

[12] T. Attema, S. Fehr, and M. Klooß, "Fiat-shamir transformation of multi-round interactive proofs," in *TCC (1)*, 2022.

[13] L. Bangalore, M. H. F. Sereshgi, C. Hazay, and M. Venkitasubramaniam, "Flag: A framework for lightweight robust secure aggregation," in *AsiaCCS*, 2023.

[14] R. Bassily, K. Nissim, U. Stemmer, and A. G. Thakurta, "Practical locally private heavy hitters," in *NIPS*, 2017.

[15] R. Bassily and A. D. Smith, "Local, private, efficient protocols for succinct histograms," in *STOC*, 2015.

[16] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun, "ACORN: Input validation for secure aggregation," in *USENIX Security*, 2023.

[17] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *CCS*, 1993.

[18] O. Ben-Eliezer, R. Jayaram, D. P. Woodruff, and E. Yogev, "A framework for adversarially robust streaming algorithms," *J. ACM*, vol. 69, 2022.

[19] E. Ben-Sasson, A. Chiesa, and N. Spooner, "Interactive oracle proofs," in *TCC (B2)*, 2016.

[20] O. Berthold and H. Langos, "Dummy traffic against long term intersection attacks," in *PETS*, 2002.

[21] A. Biswas and G. Cormode, "Interactive proofs for differentially private counting," in *CCS*. ACM, 2023.

[22] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnés, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *SOSP*, 2017.

[23] J. Böhler and F. Kerschbaum, "Secure multi-party computation of differentially private heavy hitters," in *CCS*, 2021.

[24] K. A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *CCS*, 2017.

[25] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Zero-knowledge proofs on secret-shared data via fully linear PCPs," in *CRYPTO (3)*, 2019.

[26] ——, "Lightweight techniques for private heavy hitters," in *IEEE S&P*, 2021.

[27] ——, "Arithmetic sketching," in *CRYPTO (1)*, 2023.

[28] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *CCS*, 2016.

[29] A. Broadbent and A. Tapp, "Information-theoretic security without an honest majority," in *ASIACRYPT*, 2007.

[30] M. Bun, J. Nelson, and U. Stemmer, "Heavy hitters and the structure of local privacy," *ACM Trans. Algorithms*, vol. 15, 2019.

[31] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics," in *USENIX Security*, 2010.

[32] K. N. Chadha, J. Chen, J. C. Duchi, V. Feldman, H. Hashemi, O. Javidbakht, A. McMillan, and K. Talwar, "Differentially private heavy hitter detection using federated analytics," *CoRR*, vol. abs/2307.11749, 2023.

[33] T. H. Chan, M. Li, E. Shi, and W. Xu, "Differentially private continual monitoring of heavy hitters from distributed streams," in *PETS*, 2012.

[34] M. Charikar, K. C. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theor. Comput. Sci.*, vol. 312, 2004.

[35] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, 1981.

[36] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "SGXPECTRE: Stealing intel secrets from SGX enclaves via speculative execution," in *EuroS&P*, 2019.

[37] S. G. Choi, D. Dachman-Soled, M. Kulkarni, and A. Yerukhimovich, "Differentially-private multi-party sketching for large-scale statistics," *PETS*, 2020.

[38] D. Clayton, C. Patton, and T. Shrimpton, "Probabilistic data structures in adversarial environments," in *CCS*, 2019.

[39] G. Cormode and A. Bharadwaj, "Sample-and-threshold differential privacy: Histograms and applications," in *AISTATS*, 2022.

[40] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *Proc. VLDB Endow.*, 2008.

[41] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, 2005.

[42] H. Corrigan-Gibbs, "heavyhitters," https://github.com/henrycg/heavyhitters.

[43] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *NSDI*, 2017.

[44] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *IEEE S&P*, 2015.

[45] G. Danezis, C. Fournet, M. Kohlweiss, and S. Z. Béguelin, "Smart meter aggregation via secret-sharing," in *SEGS@CCS*, 2013.

[46] G. Danezis and A. Serjantov, "Statistical disclosure or intersection attacks on anonymity systems," in *Information Hiding*, 2004.

[47] A. Davidson, P. Snyder, E. B. Quirk, J. Genereux, B. Livshits, and H. Haddadi, "STAR: Secret sharing for private threshold aggregation reporting," in *CCS*, 2022.

[48] H. Davis, C. Patton, M. Rosulek, and P. Schoppmann, "Verifiable distributed aggregation functions," *PETS*, 2023.

[49] L. de Castro and A. Polychroniadou, "Lightweight, maliciously secure verifiable function secret sharing," in *EUROCRYPT (1)*, 2022.

[50] Divvi Up, https://divviup.org/.

[51] ——, https://github.com/divviup/libprio-rs.

[52] ——, "A year-end letter from our vice president," https://divviup.org/blog/eoy-letter-2023/.

[53] R. Dorfman, "The Detection of Defective Members of Large Populations," *The Annals of Mathematical Statistics*, vol. 14, no. 4, 1943.

[54] D. Du, F. K. Hwang, and F. Hwang, *Combinatorial group testing and its applications*. World Scientific, 2000, vol. 12.

[55] Y. Duan, N. Youdao, J. F. Canny, and J. Z. Zhan, "P4P: Practical large-scale privacy-preserving distributed computation robust against malicious users," in *USENIX Security*, 2010.

[56] C. Dwork, "Differential privacy: A survey of results," in *TAMC*, 2008.

[57] T. Elahi, G. Danezis, and I. Goldberg, "Privex: Private collection of traffic statistics for anonymous communication networks," in *CCS*, 2014.

[58] Ú. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: Randomized aggregatable privacy-preserving ordinal response," in *CCS*, 2014.

[59] S. Eskandarian and D. Boneh, "Clarion: Anonymous communication from multiparty shuffling protocols," in *NDSS*, 2022.

[60] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh, "Express: Lowering the cost of metadata-hiding communication with cryptographic privacy," in *USENIX Security*, 2021.

[61] M. Faisal, J. Zhang, J. Liagouris, V. Kalavri, and M. Varia, "TVA: A multi-party computation system for secure and expressive time series analytics," in *USENIX Security*, 2023.

[62] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986.

[63] D. Froelicher, J. R. Troncoso-Pastoriza, J. S. Sousa, and J. Hubaux, "Drynx: Decentralized, secure, verifiable system for statistical queries and machine learning on distributed datasets," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, 2020.

[64] Google, "All networking pricing," https://cloud.google.com/vpc/network-pricing.

[65] ——, "Exposure notifications: Help slow the spread of covid-19, with one step on your phone," "https://www.google.com/covid19/exposurenotifications".

[66] ——, "VM instance pricing," https://cloud.google.com/compute/vm-instance-pricing.

[67] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT (2)*, 2016.

[68] S. Halevi, Y. Ishai, E. Kushilevitz, and T. Rabin, "Additive randomized encodings and their applications," in *CRYPTO (1)*, 2023.

[69] M. Hardt and D. P. Woodruff, "How robust are linear sketches to adaptive inputs?" in *STOC*, 2013.

[70] A. Hassidim, H. Kaplan, Y. Mansour, Y. Matias, and U. Stemmer, "Adversarially robust streaming algorithms via differential privacy," *J. ACM*, vol. 69, 2022.

[71] C. Hazay, M. Venkitasubramaniam, and M. Weiss, "Your reputation's safe with me: Framing-free distributed zero-knowledge proofs," *IACR Cryptol. ePrint Arch.*, p. 1523, 2022.

[72] T. Humphries, R. A. Mahdavi, S. Veitch, and F. Kerschbaum, "Selective MPC: distributed computation of differentially private key-value statistics," in *CCS*, 2022.

[73] F. K. Hwang, "A method for detecting all defective members in a population by group testing," *Journal of the American Statistical Association*, 1972.

[74] IETF, "Verifiable distributed aggregation functions draft-irtf-cfrg-vdaf-07," https://datatracker.ietf.org/doc/draft-irtf-cfrg-vdaf/.

[75] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Zero-knowledge from secure multiparty computation," in *STOC*, 2007.

[76] ISRG, "Introducing ISRG Prio services for privacy respecting metrics," "https://www.abetterinternet.org/post/introducing-prio-services".

[77] P. Jangir, N. Koti, V. B. Kukkala, A. Patra, B. R. Gopal, and S. Sangal, "Poster: Vogue: Faster computation of private heavy hitters," in *CCS*, 2022.

[78] M. Jawurek and F. Kerschbaum, "Fault-tolerant privacy-preserving statistics," in *PETS*, 2012.

[79] A. F. Karr, X. Lin, A. P. Sanil, and J. P. Reiter, "Regression on distributed databases via secure multi-party computation," in *DG.O*, 2004.

[80] J. Katz, V. Kolesnikov, and X. Wang, "Improved non-interactive zero knowledge with applications to post-quantum signatures," in *CCS*, 2018.

[81] K. Kursawe, G. Danezis, and M. Kohlweiss, "Privacy-friendly aggregation for the smart-grid," in *PETS*, 2011.

[82] K. G. Larsen, J. Nelson, H. L. Nguyen, and M. Thorup, "Heavy hitters via cluster-preserving clustering," in *FOCS*, 2016.

[83] Y. Lindell, D. Cook, T. Geoghegan, S. Gran, R. Schmidt, E. Kret, D. Kaviani, and R. A. Popa, "The deployment dilemma: Merits & challenges of deploying MPC," https://mpc.cs.berkeley.edu/blog/deployment-dilemma.

[84] H. Lycklama, L. Burkhalter, A. Viand, N. Küchler, and A. Hithnawi, "Rofl: Robustness of secure federated learning," in *IEEE S&P*, 2023.

[85] E. Margolin, K. Newatia, T. Luo, E. Roth, and A. Haeberlen, "Arboretum: A planner for large-scale federated analytics with differential privacy," in *SOSP*, 2023.

[86] L. Melis, G. Danezis, and E. D. Cristofaro, "Efficient private statistics with succinct sketches," in *NDSS*, 2016.

[87] G. T. Minton and E. Price, "Improved concentration bounds for count-sketch," 2013.

[88] I. Mironov, M. Naor, and G. Segev, "Sketching in adversarial environments," *SIAM J. Comput.*, vol. 40, 2011.

[89] D. Mouris, P. Sarkar, and N. G. Tsoutsos, "Plasma: Private, lightweight aggregated statistics against malicious adversaries with full security," Cryptology ePrint Archive, Paper 2023/080.

[90] Mozilla, "Next steps in privacy-preserving telemetry with Prio," https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/.

[91] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against Intel SGX," in *IEEE S&P*, 2020.

[92] M. Naor, B. Pinkas, and E. Ronen, "How to (not) share a password: Privacy preserving protocols for finding heavy hitters with adversarial behavior," in *CCS*, 2019.

[93] M. Naor and E. Yogev, "Bloom filters in adversarial environments," in *CRYPTO (2)*, 2015.

[94] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *CCS*, 2001.

[95] J. Nelson, H. L. Nguyen, and D. P. Woodruff, "On deterministic sketching and streaming for sparse recovery and norm estimation," *Linear Algebra and its Applications*, vol. 441, 2014.

[96] R. Pagh, "Compressed matrix multiplication," *ACM Trans. Comput. Theory*, vol. 5, 2013.

[97] R. A. Popa, A. J. Blumberg, H. Balakrishnan, and F. H. Li, "Privacy and accountability for location-based aggregate statistics," in *CCS*, 2011.

[98] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren, "Heavy hitter estimation over set-valued data with local differential privacy," in *CCS*, 2016.

[99] M. Rathee, C. Shen, S. Wagh, and R. A. Popa, "ELSA: Secure aggregation for federated learning with malicious actors," in *IEEE S&P*, 2023.

[100] M. Rosenberg, J. White, C. Garman, and I. Miers, "zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure," Cryptology ePrint Archive, Paper 2022/878.

[101] T. Steinke, "Multi-central differential privacy," *CoRR*, vol. abs/2009.05401, 2020.

[102] K. Talwar, "Differential secrecy for distributed data and applications to robust differentially secure vector summation," *CoRR*, vol. abs/2202.10618, 2022.

[103] K. Yang, P. Sarkar, C. Weng, and X. Wang, "Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field," in *CCS*, 2021.

[104] K. Yang and X. Wang, "Non-interactive zero-knowledge proofs to multiple verifiers," in *ASIACRYPT (3)*, 2022.

[105] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *NSDI*, 2017.

[106] M. Zhou, T. Wang, T. H. Chan, G. Fanti, and E. Shi, "Locally differentially private sparse vector aggregation," in *IEEE S&P*, 2022.

[107] W. Zhu, P. Kairouz, B. McMahan, H. Sun, and W. Li, "Federated heavy hitters discovery with differential privacy," in *AISTATS*, 2020.

Appendix A.
Supplemental material for Section 3

A.1. Proof of Theorem 3.1

Denote the non-silent proof system by Π_{NS} and the silent proof system by Π. Completeness of Π follows from the completeness of Π_{NS}.

Zero-knowledge. Denote the simulator for Π_{NS} by $\mathcal{S}_{\mathsf{NS}}$. Construct a simulator $\mathcal{S}$ for Π as follows:

- On input $\{x_i\}_{i\in A}$, invoke $\mathcal{S}_{\mathsf{NS}}(\{x_i\}_{i\in A})$ and receive $\{\mathsf{view}_i^*\}_{i\in A}$.
- For $i \in A$, parse $\mathsf{view}_i^* \to (x_i, \pi_i, \pi_{\mathsf{NS}}^{\mathsf{pub}}, r_{\mathsf{NS}}, \mathsf{vtag}_{\mathsf{NS},1}^*, \ldots, \mathsf{vtag}_{\mathsf{NS},v}^*)$.
- Randomly sample $\{c_i\}_{i\in[v]\setminus A} \in \{0,1\}^\rho$ and for $i \in A$, set $c_i \leftarrow H(i, x_i, \pi_i, \pi_{\mathsf{NS}}^{\mathsf{pub}})$
- Set $\pi^{\mathsf{pub}} \leftarrow (\pi_{\mathsf{NS}}^{\mathsf{pub}}, c_1, \ldots, c_v, \mathsf{vtag}_{\mathsf{NS},1}^*, \ldots, \mathsf{vtag}_{\mathsf{NS},v}^*)$.
- Program the random oracle as: $r_{\mathsf{NS}} \leftarrow H(c_1, \ldots, c_v)$.
- Sample a random r.
- For $i \in [v]$, use the randomness r to form $\mathsf{vtag}_i \in \mathbb{F}$ such that $\mathsf{vtag}_1 + \cdots + \mathsf{vtag}_v = 0$ if and only if π^{pub} in each view view_i is identical with high probability over the choice of r.
- For $i \in A$, set $\mathsf{view}_i \leftarrow (x_i, \pi_i, \pi^{\mathsf{pub}}, r, \mathsf{vtag}_1, \ldots, \mathsf{vtag}_v)$.
- Output $\{\mathsf{view}_i\}_{i\in A}$.

For statistical distance, observe that $\mathcal{S}$ perfectly simulates 1) the view of $\{c_i\}_{i\in[v]\setminus A}$ by randomly sampling them because H is a random oracle, and 2) the view of $\mathsf{vtag}_1, \ldots, \mathsf{vtag}_v$ because the local checks done by the verifiers in $\mathsf{Eval}(\cdot)$ pass (from completeness), and therefore contribute zero to $\mathsf{vtag}_1, \ldots, \mathsf{vtag}_v$ which are just a function of π^{pub} and r.

Soundness. We split the proof of soundness into two parts:

- Claim 1: Assuming that multiparty Fiat-Shamir is sound, the rest of the protocol is sound. We consider an intermediate proof system $\Pi_{\mathsf{Interactive}}$ in the $\mathcal{F}_{\mathsf{coin}}$-hybrid model. $\mathcal{F}_{\mathsf{coin}}$ returns common fresh randomness to the set of parties that invoked it. We claim that the soundness error of $\Pi_{\mathsf{Interactive}}$ is at most $\frac{1}{|\mathbb{F}|}$ worse than Π_{NS}.
- Claim 2: Loss in soundness error from multiparty Fiat-Shamir [25], [62] over $\Pi_{\mathsf{Interactive}}$ is polynomial in λ.

Claim 1. The construction of $\Pi_{\mathsf{Interactive}}$ is presented in Figure 12. If any of the following happens, by construction we have that $\mathsf{vtag}_1 + \cdots + \mathsf{vtag}_v \neq 0$ except with probability $\frac{1}{|\mathbb{F}|}$: 1) the prover equivocates about $\{\mathsf{vtag}_{\mathsf{NS},i}^*\}_{i\in[v]}$, or 2) any $\mathsf{vtag}_{\mathsf{NS},i}^*$ is incorrectly computed by prover, or 3) $\mathsf{Ver}_{\mathsf{NS}}(\mathsf{vtag}_{\mathsf{NS},1}^*, \ldots, \mathsf{vtag}_{\mathsf{NS},v}^*) \neq 0$. Therefore, the soundness error of $\Pi_{\mathsf{Interactive}}$ is $\epsilon_{\mathsf{Int}} \leqslant \epsilon + \frac{1}{|\mathbb{F}|}$.

Claim 2. Consider the adversarial strategies defined in Definition A.1 for generating proofs for silent proof system Π. We now analyze the loss in advantage for the adversary $\mathcal{A}$ if restricted to $\mathcal{A}_{\mathsf{FS1}}$ and $\mathcal{A}_{\mathsf{FS2}}$. If the adversary equivocates or $c_i \neq H_i(x_i, \pi_i, \pi_{\mathsf{NS}}^{\mathsf{pub}})$, then verifiers accept with probability at most $\frac{1}{|\mathbb{F}|}$. If the adversary correctly guesses c_i without making the call $H_i(x_i, \pi_i, \pi_{\mathsf{NS}}^{\mathsf{pub}})$, then it can simply make this call at the very end to satisfy the restriction without affecting its advantage. Therefore, the adversary's advantage is reduced at most by $\frac{1}{|\mathbb{F}|}$. Now, for $\mathcal{A}_{\mathsf{FS2}}$, no additional advantage is lost because $\mathcal{A}$ can always make the outer call $H(c_1, \ldots, c_v)$ at the end if not made before and it can always pad to T queries with dummies.

We now have two mutually exclusive adversarial strategies $\mathcal{A}_{\mathsf{FSRev}}$ and $\mathcal{A}_{\mathsf{FSFwd}}$ whose union represents all of $\mathcal{A}_{\mathsf{FS2}}$. $\mathcal{A}_{\mathsf{FSRev}}$ produces an output with probability at most $\frac{vT^2}{|\mathbb{F}|}$ because the expected number of collisions between the i-th input of any of the T outer H calls and the output of any of the T inner H_i calls which are made afterward is at most $\frac{T^2}{|\mathbb{F}|}$, and the probability of this happening for any $i \in [v]$ is at most $\frac{vT^2}{|\mathbb{F}|}$. We analyze $\mathcal{A}_{\mathsf{FSFwd}}$ in Lemma A.2.

Interactive proof system $\Pi_{\mathsf{Interactive}}$. $\Pi_{\mathsf{Interactive}}$ from non-silent proof system $\Pi_{\mathsf{NS}} = (\mathsf{Gen}_{\mathsf{NS}}, \mathsf{Eval}_{\mathsf{NS}}, \mathsf{Ver}_{\mathsf{NS}})$.

- Prover does $(\pi_1, \ldots, \pi_v, \pi_{\mathsf{NS}}^{\mathsf{pub}}) \leftarrow \mathsf{Gen}_{\mathsf{NS}}(x_1, \ldots, x_v)$ and sends $(\pi_i, \pi_{\mathsf{NS}}^{\mathsf{pub}})$ to i-th verifier.
- All parties invoke $\mathcal{F}_{\mathsf{coin}}$ and receive r_{NS}.
- Prover does $\mathsf{vtag}_{\mathsf{NS},i}^* \leftarrow \mathsf{Eval}_{\mathsf{NS}}(x_i, \pi_i, \pi_{\mathsf{NS}}^{\mathsf{pub}}; r_{\mathsf{NS}}), \forall i$.
- Prover sends $\mathsf{vtag}_{\mathsf{NS},1}^*, \ldots, \mathsf{vtag}_{\mathsf{NS},v}^*$ to all verifiers.
- The verifiers invoke $\mathcal{F}_{\mathsf{coin}}$ and receive r, and then for $i \in [v]$, i-th verifier does:
  - Derives tag $\mathsf{vtag}_{\mathsf{NS},i} \leftarrow \mathsf{Eval}_{\mathsf{NS}}(x_i, \pi_i, \pi_{\mathsf{NS}}^{\mathsf{pub}}; r_{\mathsf{NS}})$.
  - Outputs vtag_i s.t. $\mathsf{vtag}_1 + \cdots + \mathsf{vtag}_v = 0$ iff these checks pass with prob. $\geqslant 1 - \frac{1}{|\mathbb{F}|}$ over r's choice:
    - All verifiers received the same $\{\mathsf{vtag}_{\mathsf{NS},i}^*\}_{i\in[v]}$.
    - $\mathsf{vtag}_{\mathsf{NS},i}^* = \mathsf{vtag}_{\mathsf{NS},i}$.
    - $\mathsf{Ver}_{\mathsf{NS}}(\mathsf{vtag}_{\mathsf{NS},1}^*, \ldots, \mathsf{vtag}_{\mathsf{NS},v}^*) = 0$.

Figure 12: Interactive proof system $\Pi_{\mathsf{Interactive}}$

Definition A.1. Consider an adversarial prover $\mathcal{A}$ for the silently verifiable proof system from Figure 4. The adversary is allowed random oracle queries of two types. Denoted by $H_i(a, b, c)$, the "inner" queries are of the form $H(i, a, b, c)$ for $i \in [v]$, and the rest of the queries to H are the "outer" queries. We define four adversarial strategies:

- $\mathcal{A}_{\mathsf{FS1}}$: If the adversary doesn't equivocate and outputs $(x_1, \ldots, x_v)$, $(\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}})$ where $\pi^{\mathsf{pub}} = (\pi_{\mathsf{NS}}^{\mathsf{pub}}, c_1, \ldots, c_v, \mathsf{vtag}_{\mathsf{NS},1}, \ldots, \mathsf{vtag}_{\mathsf{NS},v})$, then for all $i \in [v]$, it receives c_i as the output of $H_i(x_i, \pi_i, \pi_{\mathsf{NS}}^{\mathsf{pub}})$.
- $\mathcal{A}_{\mathsf{FS2}}$: Same as $\mathcal{A}_{\mathsf{FS1}}$ but also makes the outer call $H(c_1, \ldots, c_v)$. Moreover, all calls to random oracles are distinct, and exactly T calls are made to each H_i and H.
- $\mathcal{A}_{\mathsf{FSRev}}$: $\mathcal{A}_{\mathsf{FS2}}$ when there exists $i \in [v]$ such that the invocation $c_i \leftarrow H_i(x_i, \pi_i, \pi_{\mathsf{NS}}^{\mathsf{pub}})$ is made *after* $H(c_1, \ldots, c_v)$.

Multiparty Fiat-Shamir reduction. Prover P^*_{Int} for $\Pi_{\mathsf{Interactive}}$ from black-box access to a prover $\mathcal{A}$ for the silent proof system Π. $\mathcal{A}$ makes T queries each to the v inner random oracles $H_1, \ldots, H_v$ and the outer random oracle H. H_i denotes queries of form $H(i, \cdot, \cdot, \cdot)$.

- Pick a random integer $i^* \in [T]$.
- For $i \in [v]$, initialize an empty list L_i.
- Run $\mathcal{A}$ and respond to it as follows except the i^*-th outer query made by $\mathcal{A}$:
  - For inner query $H_i(y, y^\pi, y^{\mathsf{pub}})$: pick a random z, add "$(y, y^\pi, y^{\mathsf{pub}}) : z$" to L_i and return z.
  - For outer query $H(z_1, \ldots, z_v)$: return a random value.
- For the i^*-th outer query $H(z_1^*, \ldots, z_v^*)$:
  - For each z_i^*, scan the list L_i and from all the entries which map to z_i^*, randomly pick an entry $(y_i^*, y_i^{\pi *}, y^{\mathsf{pub}*})$ such that $y^{\mathsf{pub}*}$ is common in all the v entries picked. If no entry exists, abort.
  - For $i \in [v]$, send $(y_i^*, y_i^{\pi *}, y^{\mathsf{pub}*})$ to i-th verifier.
  - Receive r from the verifiers and return r to $\mathcal{A}$.
- Receive $(x_1, \ldots, x_v, \pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}})$ from $\mathcal{A}$, where $\pi^{\mathsf{pub}} \to (\pi^{\mathsf{pub}}_{\mathsf{NS}}, c_1, \ldots, c_v, \mathsf{vtag}_{\mathsf{NS},1}, \ldots, \mathsf{vtag}_{\mathsf{NS},v})$.
- Check the following and abort if any fails: $\pi^{\mathsf{pub}}_{\mathsf{NS}} = y^{\mathsf{pub}*}$, $x_i = y_i^*$, $\pi_i = y_i^{\pi *}$ and $c_i = z_i^*$ for $i \in [v]$.
- Send $(\mathsf{vtag}_{\mathsf{NS},1}, \ldots, \mathsf{vtag}_{\mathsf{NS},v})$ to all the verifiers.

Figure 13: Malicious prover for $\Pi_{\mathsf{Interactive}}$

- $\mathcal{A}_{\mathsf{FSFwd}}$: $\mathcal{A}_{\mathsf{FS2}}$ where for all $i \in [v]$, invocations $c_i \leftarrow H_i(x_i, \pi_i, \pi^{\mathsf{pub}}_{\mathsf{NS}})$ are made *before* $H(c_1, \ldots, c_v)$.

Lemma A.2. *In a silently verifiable proof system Π, all adversaries with strategy $\mathcal{A}_{\mathsf{FSFwd}}$ (Definition A.1) who generate the output $(x_1, \ldots, x_v, \pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}})$ with T random oracle calls, where $x = \sum_{i\in[v]} x_i \notin \mathcal{L}$, the v honest verifiers accept with probability at most $\frac{\epsilon_{\mathsf{Int}} T}{(1-\frac{T}{|\mathbb{F}|})^v}$, where ϵ_{Int} is the probability that the honest verifiers in $\Pi_{\mathsf{Interactive}}$ (Figure 12) accept any interaction for $x \notin \mathcal{L}$.*

Proof. We construct a malicious prover P^*_{Int} in Figure 13 such that for any adversary $\mathcal{A}$ with strategy $\mathcal{A}_{\mathsf{FSFwd}}$ making the verifiers accept an incorrect statement in Π with probability p, P^*_{Int} makes the verifiers in $\Pi_{\mathsf{Interactive}}$ accept on the same statement with probability $\frac{p}{T} \cdot \left(1 - \frac{T}{|\mathbb{F}|}\right)^v$.

An accepting proof generated via $\mathcal{A}_{\mathsf{FSFwd}}$ leads to an accepting interaction for P^*_{Int} as long as the guess i^* is correct and the correct entry $(y_i^*, y_i^{\pi *}, y^{\mathsf{pub}*})$ is picked from the list L_i for all $i \in [v]$.

We first look at the guess i^*. By definition, $\mathcal{A}_{\mathsf{FSFwd}}$ makes T calls to the outer random oracle H and one of these calls corresponds to the output it generates. The probability that the guess i^* is correct is $\frac{1}{T}$. Now, we look at the entry picked by P^*_{Int} from the list L_i when responding to the i^*-th outer query. From the restriction on $\mathcal{A}_{\mathsf{FSFwd}}$, we know that the correct entry exists in the list L_i if guess i^* is correct, but there can be other entries that map to the same z_i^*. The probability of collisions from other $T-1$ inner H_i queries is $\leqslant \frac{T}{|\mathbb{F}|}$. The probability that all the guesses are correct is $\geqslant \frac{1}{T} \cdot \left(1 - \frac{T}{|\mathbb{F}|}\right)^v$.

Wrapping it up, P^*_{Int} generates an accepting interaction with probability $\geqslant p \cdot \frac{1}{T} \cdot \left(1 - \frac{T}{|\mathbb{F}|}\right)^v$. Since $\epsilon_{\mathsf{Int}} \geqslant \frac{p}{T} \cdot \left(1 - \frac{T}{|\mathbb{F}|}\right)^v$, we have that $p \leqslant \frac{\epsilon_{\mathsf{Int}} \cdot T}{(1-\frac{T}{|\mathbb{F}|})^v}$. □

Appendix B. Supplemental material for Section 4

Correctness against malicious clients. We prove that our private-aggregation scheme (§4.2) satisfies correctness against malicious clients. As long as the servers remove all invalid submissions before aggregation, from the correctness of the additive encoding, the output will be $f(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are valid submissions. Consider a submission $(e_1, \ldots, e_v)$, $(\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}})$. Let $e = \sum_{i\in[v]} e_i$ and for $i \in [v]$, vtag_i is the verification tag generated by i-th server from $e_i, \pi_i, \pi^{\mathsf{pub}}$. For an invalid encoding, i.e., $V(e) \neq 1$, the probability that $\sum_{i\in[v]} \mathsf{vtag}_i = 0$ is $\leqslant \epsilon$ where ϵ is the soundness error of our silently verifiable proof system. When $\sum_{i\in[v]} \mathsf{vtag}_i \neq 0$, the probability that batch verification passes $(\mathsf{vtag}_1, \ldots, \mathsf{vtag}_v)$ is $\leqslant \frac{1}{|\mathbb{F}|}$ (from random linear combination). Since servers test at most $\log \frac{n}{d}$ batches per client where d is an upper bound on the number of invalid submissions, the probability that the invalid submission isn't caught in any batch is $\leqslant \frac{1}{|\mathbb{F}|} \cdot \log \frac{n}{d}$. Therefore, the probability that all d invalid submissions are caught by the servers is $\geqslant 1 - d \cdot \epsilon - \frac{d}{|\mathbb{F}|} \cdot \log \frac{n}{d}$.

Appendix C. Supplemental material for Section 5

Proof of Theorem 5.2. For the challenger to output "1", all the encodings $y_1, \ldots, y_m$ should be valid. In a valid encoding, the magnitude at each coordinate is at most 1, therefore, $\|y_1 + \cdots + y_m\|_\infty \leqslant m$. Consider a heavy-hitting string x which appears $T_x \geqslant T$ times in $x_1, \ldots, x_n$ and let $b = H_{\text{bucket}}(x)$. Parse the sum of all encodings $S \to ((c_1||s_1), \ldots, (c_B||s_B))$. We focus on the b-th bucket: $c_b||s_b \in \mathbb{F} \times \mathbb{F}^{L+1}$. Since all the encodings are valid, the counter field c_b is a sum of values in $\{0, 1\}$ with at least T_x of them being 1 (from all the occurrences of x). Therefore, $c_b \geqslant T$. This satisfies the condition that rounding will be performed on the b-th bucket during output decoding (Figure 5). What's left to argue is when rounding and subsequent complementing recovers x.

The output of the rounding operation is invariant to the magnitude of each coordinate in a bucket; it only reads the sign. As long as the sign of each coordinate in s_b matches that of an encoding of x, the output will be correct. Let z_x be

Additive encoding for bucketed string counting.
Additive encoding for the aggregation function $f\colon \mathcal{X}^n \to \mathcal{Y}$, where:

- the input space is $\mathcal{X} = ([B] \times \{0,1\}^L)$, pairs of a bucket ID and an L-bit string, and
- the output space is $\mathcal{Y} = \mathbb{F}^B \times (\mathbb{F}^L)^B$, for each bucket, the number of strings in the bucket and the sum (over $\mathbb{F}^L$) of the strings in that bucket.

The scheme takes as parameters: a finite field $\mathbb{F}$, a number of buckets B, the bitlength L of the input strings. The encoding length is $\ell = (L+1) \cdot B$.

Encoding $E(b \in [B], \sigma \in \{0,1\}^L)$**:**

- Represent σ as a vector $\hat{\sigma} \in \{-1,1\}^L$ over $\mathbb{F}$.
- For $i = 1, \ldots, B$:
  - If $i = b$, set $s_i \leftarrow (1 \| \hat{\sigma}) \in \mathbb{F}^{L+1}$.
  - Otherwise, set $s_i \leftarrow \mathbf{0} \in \mathbb{F}^{L+1}$.
- Output $(s_1, \ldots, s_B) \in (\mathbb{F}^{L+1})^B$.

Verification $V(y \in \mathbb{F}^\ell)$**:**

- Parse $((c_1 \| s_1), \ldots, (c_B \| s_B)) \leftarrow y \in (\mathbb{F}^{L+1})^B$.
- Check that there is a unique $i \in [B]$ such that $(c_i || s_i)$ is a non-zero vector.
- Check that $c_i = 1 \in \mathbb{F}$ and $s_i \in \{-1,1\}^L \subseteq \mathbb{F}^L$.

Decoding $D(y \in \mathbb{F}^\ell)$**:**

- Parse $((c_1 \| s_1), \ldots, (c_B \| s_B)) \leftarrow y \in (\mathbb{F}^{L+1})^B$.
- Output $(c_1, \ldots, c_B), (s_1, \ldots, s_B) \in \mathbb{F}^B \times (\mathbb{F}^L)^B$.

Figure 14: Additive encoding for bucketed string counting.

the encoding of x generated by the client input preparation step (Figure 5). We split S as follows:

$$S = (y_1 + \cdots + y_m) + \Big((z_x + \cdots + z_x) + \sum_{i \in Z_{\text{Heavy}}} z_i + \sum_{i \in Z_{\text{Tail}}} z_i\Big)$$
$$= S_y + (S_x + S_{\text{Heavy}} + S_{\text{Tail}})$$

where $Z_{\text{Heavy}}, Z_{\text{Tail}}$ denote subsets of $[n]$ such that Z_{Heavy} contains indices i where z_i is an encoding of a heavy-hitting string other than x, while Z_{Tail} contains all the remaining (non-heavy) indices. In a similar way, the b-th bucket of S can be written as:

$$s_b = W_y + (W_x + W_{\text{Heavy}} + W_{\text{Tail}})$$

We now look at the event $\|W_y + W_{\text{Heavy}} + W_{\text{Tail}}\|_\infty < T$. This event leads to the correct recovery of x because each coordinate of $|W_x|$ is $\geqslant T$, and therefore, their signs are preserved in s_b.

Recall that we proved above that $\|W_y\|_\infty \leqslant m$. When $\|W_{\text{Heavy}}\|_\infty + \|W_{\text{Tail}}\|_\infty < T - m$ holds, it implies $\|W_y + W_{\text{Heavy}} + W_{\text{Tail}}\|_\infty < T$.

Bounding $\|W_{Heavy}\|_\infty$. Consider a heavy-hitting string $x' \neq x$. Since $x_1, \ldots, x_n$ (and therefore, x, x') are picked before the hash functions are sampled, we have that $\Pr[H_{\text{bucket}}(x') = b] \leqslant \frac{1}{B}$. In the flip case, when $H_{\text{bucket}}(x') \neq b$, the b-th bucket in the encoding of x' will be $\mathbf{0}$. Using triangle inequality and union bound over all $k-1$ heavy-hitting strings other than x, we get:

$$\Pr\left[\|W_{\text{Heavy}}\|_\infty > 0\right] \leqslant \frac{k}{B}$$

Bounding $\|W_{Tail}\|_\infty$. Recall that W_{Tail} represents the contents of the b-th bucket in $\sum_{i \in Z_{\text{Tail}}} z_i$. For $i \in Z_{\text{Tail}}$, denote the contents of the b-th bucket of encoding z_i by $s^{\text{Tail}}_{i,b} \in \mathbb{F}^{L+1}$. Using Markov's inequality,

$$\Pr\left[\|W_{\text{Tail}}\|_\infty \geqslant T - m\right] \leqslant \Pr\left[\sum_{i \in Z_{\text{Tail}}} \|s^{\text{Tail}}_{i,b}\|_\infty \geqslant T - m\right]$$
$$\leqslant \frac{\mathrm{E}\left[\sum_{i \in Z_{\text{Tail}}} \|s^{\text{Tail}}_{i,b}\|_\infty\right]}{T - m}$$

We know that $s^{\text{Tail}}_{i,b}$ is either $\mathbf{0}$ or $s^{\text{Tail}}_{i,b} \in \{-1,1\}^{L+1}$. The latter happens with $\frac{1}{B}$ probability because hash functions are randomly sampled after $x_1, \ldots, x_n$ are committed. Therefore,

$$\mathrm{E}\left[\sum_{i \in Z_{\text{Tail}}} \|s^{\text{Tail}}_{i,b}\|_\infty\right] = \sum_{i \in Z_{\text{Tail}}} \mathrm{E}\left[\|s^{\text{Tail}}_{i,b}\|_\infty\right] = \sum_{i \in Z_{\text{Tail}}} \frac{1}{B} = \frac{\bar{h}}{B}$$

Winding it up.

$$\Pr\left[x \notin H\right] \leqslant \Pr\left[\|W_{\text{Heavy}}\|_\infty + \|W_{\text{Tail}}\|_\infty \geqslant T - m\right]$$
$$\leqslant \Pr\left[\|W_{\text{Heavy}}\|_\infty > 0\right] + \Pr\left[\|W_{\text{Tail}}\|_\infty \geqslant T - m\right]$$
$$\leqslant \frac{k}{B} + \frac{\bar{h}}{B \cdot (T - m)}$$

Taking a union bound over all k heavy-hitting strings, we get $\mathsf{HHAdv}_{L,B,n,m,T}[\mathcal{A}] \leqslant \frac{k^2}{B} + \frac{k \cdot \bar{h}}{B \cdot (T-m)}$ □

Corollary C.1 (Heavy hitters: success amplification)**.** *For a finite field $\mathbb{F}$, a string length L, bucket count B, a number of honest clients n, a number of malicious clients m, a heavy hitter threshold T, a number of sessions R and adversary $\mathcal{A}$, let* $\mathsf{HHAdvMulti}_{L,B,n,m,T,R}[\mathcal{A}]$ *denote the probability that the experiment G1 of figure 15 outputs "1". Given same conditions as theorem 5.2,*

$$\mathsf{HHAdvMulti}_{L,B,n,m,T,R}[\mathcal{A}] \leqslant k \cdot \left(\frac{k}{B} + \frac{\bar{h}}{B \cdot (T-m)}\right)^R$$

Proof. Follows from the proof of theorem 5.2. □

Corollary C.2 (Heavy hitters: bounded adversarial error)**.** *For a finite field $\mathbb{F}$, a string length L, a bucket count B, a number of honest clients n, a number of malicious clients m, a heavy hitter threshold T, a bound Δ on the error and adversary $\mathcal{A}$, let* $\mathsf{HHAdvDP}_{L,B,n,m,T,\Delta}[\mathcal{A}]$ *denote the probability that the experiment G2 of figure 15 outputs "1". Given same conditions as theorem 5.2 with* $\mathrm{char}(\mathbb{F}) > 2(n+m+\Delta)$ *and k' distinct heavy-hitting strings*

Variants of heavy-hitters game. The game is parameterized by the same parameters as figure 6.

G1: Heavy-hitters game with parallel sessions. Takes additional parameter R denoting the number of parallel sessions.

- Same as step 1 in figure 6 (shared across all sessions).
- The challenger initializes the set $H' \leftarrow \{\}$.
- For $r \in [R]$:
  - Same steps as figure 6 excluding steps 1, 7.
  - The challenger updates $H' \leftarrow H' \cup H$ where H is the result of output-decoding for the current session.
- If there is a string $\sigma \in \{0,1\}^L$ that:
  - appears more than T times in the list $(x_1, \ldots, x_n)$ of the honest clients' inputs, and
  - does not appear in the set of heavy hitters H',

  then the challenger outputs "1."

G2: Heavy-hitters game with error. Takes additional parameter Δ denoting the bound on error.

- First two steps same as steps 1 and 2 in figure 6.
- The adversary outputs m encodings along with an error vector δ, each in $\mathbb{F}^\ell$:

  $$(y_1, \ldots, y_m, \delta) \leftarrow \mathcal{A}^{H_{\text{sign}}, H_{\text{bucket}}}(\mathsf{st}).$$

  If, for any $i \in [m]$, the encoding v_i is invalid—i.e., $V(y_i) = 0$, or if $\|\delta\|_\infty > \Delta$, the experiment output is "0."
- Same as step 4 in figure 6.
- The challenger computes the sum $S \in \mathbb{F}^\ell$ of all encodings and δ:

  $$S \leftarrow (y_1 + \cdots + y_m) + (z_1 + \cdots + z_n) + \delta \in \mathbb{F}^\ell.$$

- Same as step 6 in figure 6.
- If there is a string $\sigma \in \{0,1\}^L$ that:
  - appears more than $T + \Delta$ times in the list $(x_1, \ldots, x_n)$ of the honest clients' inputs, and
  - does not appear in the set of heavy hitters H,

  then the challenger outputs "1."

Figure 15: Security games for our heavy-hitters protocol.

among honest parties' input each of which occurs $> T + \Delta$ times,

$$\mathsf{HHAdvDP}_{L,B,n,m,T,\Delta}[\mathcal{A}] \leqslant \frac{k' \cdot k}{B} + \frac{k' \cdot \bar{h}}{B \cdot (T - m)}$$

Proof. Follows from the proof of theorem 5.2. □

Lemma C.3 (Heavy hitters: correctness against malicious clients with ℓ_2 bound)**.** *Given the same parameters and conditions as Corollary C.1. Let $\bar{h}_2$ denote the ℓ_2 norm of the histogram of strings excluding the k heavy hitters. Then we have*

$$\mathsf{HHAdvMulti}_{L,B,n,m,T,R}[\mathcal{A}] \leqslant (L+1) \cdot \left(\frac{k^2}{B} + \frac{k \cdot \bar{h}_2^2}{B \cdot (T-m)^2}\right)^R$$

Proof. This follows similarly to the analysis of ℓ_2 error in count sketch [34], [87]. □

Appendix D. Smaller field for silently verifiable proofs

From Theorem 3.1, we have that silently verifiable proofs are sound against adversaries running in at most T time only if $|\mathbb{F}| \geqslant T^2$. In this section, we discuss an optimization which allows fields with $|\mathbb{F}| \geqslant T$, but requires concurrently proving the statement twice. The high-level idea is to ask the prover to provide two sets of proofs on the same statement. The randomness for each proof is different, but comes from the same random oracle call. In particular, each random oracle call outputs double the bits, one half for each proof.

We now describe the changes to the proof generation step in Figure 4; the proof evaluation step requires similar changes.

- The first step remains the same. Generate the non-silent proof:

  $$(\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}}_{\mathsf{NS}}) \leftarrow \mathsf{Gen}_{\mathsf{NS}}(x_1, \ldots, x_v)$$

- Use multiparty Fiat-Shamir [25] to derive the randomness $r_{\mathsf{NS}}, r'_{\mathsf{NS}}$ that the verifiers will use to verify the proof:

  $$c_i, c'_i \leftarrow H(i, x_i, \pi_i, \pi^{\mathsf{pub}}_{\mathsf{NS}}), \text{ for } i \in [v]$$
  $$r_{\mathsf{NS}}, r'_{\mathsf{NS}} \leftarrow H(c_1, c'_1, \ldots, c_v, c'_v).$$

- Derive the verification tags that the verifiers would broadcast to verify the non-silent proof with randomness $r_{\mathsf{NS}}, r'_{\mathsf{NS}}$. That is, for $i \in [v]$, set

  $$\mathsf{vtag}_{\mathsf{NS},i} \leftarrow \mathsf{Eval}_{\mathsf{NS}}(x_i, \pi_i, \pi^{\mathsf{pub}}_{\mathsf{NS}}; r_{\mathsf{NS}}) \in \mathbb{F}^q.$$
  $$\mathsf{vtag}'_{\mathsf{NS},i} \leftarrow \mathsf{Eval}_{\mathsf{NS}}(x_i, \pi_i, \pi^{\mathsf{pub}}_{\mathsf{NS}}; r'_{\mathsf{NS}}) \in \mathbb{F}^q.$$

  Then compute:

  $$\pi^{\mathsf{pub}} \leftarrow (\pi^{\mathsf{pub}}_{\mathsf{NS}}, c_1, c'_1, \ldots, c_v, c'_v, \mathsf{vtag}_{\mathsf{NS},1}, \mathsf{vtag}'_{\mathsf{NS},1}, \ldots, \mathsf{vtag}_{\mathsf{NS},v}, \mathsf{vtag}'_{\mathsf{NS},v}).$$

- Output $(\pi_1, \ldots, \pi_v, \pi^{\mathsf{pub}})$.

If the non-silent proof generation uses Fiat-Shamir, then recursively apply the changes to the non-silent proof as well.

Appendix E. Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

E.1. Summary

The paper presents Whisper, a design for privacy-preserving collection aggregate statistics. As part of the design, they propose "silently verifiable proofs" based on MPC-in-the-head, that allows a set of anytrust servers to verify large batch of proofs on secret shared data by exchanging a single 128-bit string. Their system significantly reduces the server-to-server communication overhead by reasonably increasing the communication overhead for the client.

E.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Provides a Valuable Step Forward in an Established Field

E.3. Reasons for Acceptance

1) Addresses an important problem of calculating statistics over private client data securely with different applications.
2) Uses a new construct called silently verifiable proofs that builds upon a new ZK proof system that allows batch checking and zero-knowledge against malicious verifiers.
3) The authors implement Whisper and provide extensive benchmarks and open source their code.

E.4. Noteworthy Concerns

The prover time of the Whisper system is slower than previous systems which might put more computation overhead on the clients.

Appendix F. Response to the Meta-Review

In our experiments, the Whisper prover time is indeed 2–3× that of the baseline, as we discuss in §6. At the same time, the absolute cost is at most 64 milliseconds for the applications we discuss, which may not be a concern in practice. Reducing the prover time is a worthwhile direction for future work.