

MDSS-STAR: Private Heavy-Hitters through Multi-Dealer Secret Sharing

Harry Eldridge¹, Aditya Hegde¹, Brennon Brimhall¹, Gabrielle Beck², and Matthew Green¹

¹Johns Hopkins University, {hme, ahegde, brimhall, mgreen}@cs.jh.edu

²CNRS, LIRMM, gabrielle.beck@lirmm.fr

Abstract

We propose a new private telemetry system for computing t -heavy hitters in the STAR [22] and POPSTAR [31] model. In this setting, each client generates a report with the assistance of a lightweight Randomness Server and submits it to a central Aggregation Server, which can then locally compute only the heavy hitters. As compared to STAR and POPSTAR—which reveal either the full (pseudonymized) frequency histogram or a complex function of it—our protocol reduces leakage: the Aggregation Server learns non-heavy-hitter values only if their frequency exceeds a well-defined threshold. Additionally, while STAR and POPSTAR are insecure in the face of an Aggregation Server that colludes with clients, our protocol provides optimal security against such a colluding server.

To achieve these privacy guarantees, our protocol efficiently adapts multi-dealer secret sharing [25] to the STAR/POPSTAR model and introduces a novel oblivious secret-share sampling protocol to ensure security against a colluding Aggregation Server. We implement and benchmark the performance of our protocol and find that it is practical for a number of use cases. Moreover, we show that it supports a tunable three-way tradeoff between correctness, efficiency and privacy.

1 Introduction

Modern software development often requires the collection of *telemetry*, typically information gathered from users that reveals how a product is used. This information is invaluable for identifying software defects (e.g. via crash or bug reports) and for guiding product development efforts, such as by tracking how and when users interact with deployed features.

However, unrestricted telemetry access can lead to privacy concerns. For example, a system that records searched URLs may inadvertently expose sensitive user information, especially in cases where a URL is accessed by only a small number of users, such as a webpage related to a specific medical condition. The need to respect privacy concerns has spurred

substantial work in the area of privacy-preserving statistics calculations [1, 3, 5–10, 13, 15–17, 20, 23, 30, 32–35, 38, 39, 42] including real-world deployments [2] and recent standardization efforts by the IETF [26]. One particular area that has received significant attention in the private telemetry space is the *private t -heavy-hitters problem* [4, 14, 16, 22, 23, 28, 31, 38], where a data aggregator only learns responses that are reported by at least t clients. In the previous example, this would allow the service provider to obtain a list of popular URLs while keeping rarer and thus potentially more sensitive searches private.

In previous work this problem has seen two approaches. One direction focuses on concretely efficient constructions from highly optimized two-party computation (2PC) protocols [4, 16, 28]. While efficient, these works require multiple servers, and the strong assumptions that (1) the enlisted servers will not collude, despite being in constant interactive communication, and (2) that it will be possible for a data aggregator to identify an honest third-party that is willing to expend computational resources equal to the data aggregator for little to no benefit. Some earlier works [4, 16] are also impacted by inherent leakage from the MPC functionality, which they reduce using differential privacy [24].

An alternative model In CCS ’22, Davidson et al. proposed STAR [22]. STAR introduces a new paradigm with two non-colluding servers between which no interaction needs to occur. Critically, one server can even be relatively lightweight. In this model, clients report their inputs in two phases: first, they run a protocol with a *Randomness Server* to compute a *report*. The Randomness Server has minimal resource requirements: its interaction with each client consumes only small amounts of bandwidth, is usually computationally inexpensive, and results in no increased storage for the server. After interacting with the Randomness Server, clients transmit their report to an independent *Aggregation Server*, which uses the collected reports to *locally* compute the t -heavy-hitters. [Figure 1](#) provides an overview of the STAR model.

The STAR protocol is extremely efficient, with the cloud-

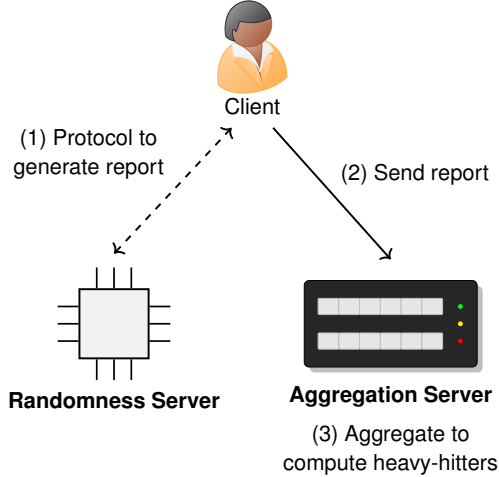


Figure 1: The STAR model. Note that there is an implicit asymmetry in the computational resources of the randomness server and the Aggregation Server.

compute cost of aggregating 100,000 client reports estimated to be a fraction of a cent. Unfortunately, the protocol suffers from substantial privacy leakage: the Aggregation Server learns a pseudonomized version of the *complete frequency histogram* of client inputs. More specifically, while reports do not include the exact client input used to create them, they do embed a pseudonymous “label” that allows the Aggregation Server to bucket all reports made on the same input. The STAR protocol effectively decrypts this label for only reports that correspond to the heavy-hitters. In recognition of this privacy limitation, a follow-up work called POPSTAR (Li et al., USENIX 2024) reduces the leakage to a partial histogram. This protocol uses a *prefix tagging* technique to replace STAR’s uniquely-identifying label with a short tag, such that high-count reports will be uniquely identifiable, but low-count reports will experience tag collisions. While all reports in POPSTAR leak some pseudonym of the client’s input, for low-count reports this leakage can be as little as one or two bits.¹

More critically, both STAR and POPSTAR are extremely fragile in the face of an Aggregation Server that behaves dishonestly. Specifically, an Aggregation Server that makes a *single query* to the Randomness Server (e.g., by pretending to be a client) can identify *all* reports of a target input value v , even if v is not a heavy-hitter.² While this attack was acknowledged in prior work, it was explicitly defined to be outside the threat model [22, 31], despite being a realistic threat to any deployment.

¹We provide a more detailed description of the POPSTAR leakage, as well as a comparison to our scheme, in Section 2.1.

²From a cryptographic perspective, we emphasize that this attack is allowed even in the *honest-but-curious* setting, as it only requires the Aggregation Server to learn a client’s view during the report generation phase.

Our approach In this work, we propose a new paradigm for t -heavy-hitters in the STAR model. Our new protocol MDSS-STAR provides reduced leakage relative to STAR and POPSTAR, and contains strong new protections against corrupted clients.

As our protocol is inspired by the STAR scheme, we begin by briefly recalling their construction. In STAR, clients send the Aggregation Server a t -out-of- n Shamir secret share of their input value v . The Aggregation Server then receives a collection of shares for many different secrets, and uses them to reconstruct the secrets with at least t shares. The main limitation of this approach is that a traditional secret sharing scheme does not permit the intermingling of shares of different secrets during reconstruction. To avoid this, the Randomness Server provides each client with a *tag* computed deterministically from the value v , and the client appends this tag to its report. The Aggregation Server then reconstructs each group of shares that contain the same tag. Since it is these tags that lead to the leakage inherent in STAR, we would like to remove the tags but still allow for secret reconstruction, i.e. use an algorithm that can simultaneously reconstruct secrets from a collective pool of shares. Fortunately, such a scheme was recently defined in the work of Eldridge et al. under the name Multi-Dealer Secret Sharing (MDSS) [25].

Thus, as a first step, our MDSS-STAR construction replaces the Shamir shares in client reports with MDSS shares, allowing us to eliminate the need for tags while preserving the Aggregation Server’s ability to recover secrets. Performing this substitution while maintaining efficiency is a challenge, and we do so by optimizing the MDSS reconstruction algorithm. We validate our results by showing that heavy hitters can be recovered from a large numbers of reports.

As a second contribution, we design a new protocol that greatly enhances security against a dishonest Aggregation Server. Concretely, we exploit the structure of MDSS shares to design an efficient *oblivious share sampling* (OSS) protocol. This protocol ensures that a party issuing a query to the Randomness Server learns only their MDSS share and nothing else, removing the ability to monitor a value of interest after a single query. Our OSS protocol is in turn built on an efficient *doubly oblivious polynomial evaluation* (DOPE) protocol, which may be of independent interest. We describe these primitives and our overall approach in more detail in Section 2.

Finally, we implement and benchmark our algorithms to evaluate them against the state-of-the-art, and give some exemplary parameter configurations that show that our protocol is practically efficient. Aggregation on a collection of 100,000 client reports with privacy guarantees comparable to POPSTAR takes approximately 10 hours on standard hardware. Report generation is efficient: both the client and Randomness server have a computation time of approximately 1 second, and the total communication between the parties is approximately 50 kilobytes.

2 Technical Overview

Overview of the STAR protocol Our proposed construction is based on the STAR protocol, which we briefly recall here. In STAR, a client with value v computes a t -out-of- n secret sharing of their value $\langle v \rangle = (\langle v \rangle_1, \dots, \langle v \rangle_n)$, samples an index i , and sends $\langle v \rangle_i$ to the Aggregation Server. When ℓ clients report the same value v , the aggregation server learns $\langle v \rangle_{i_1}, \dots, \langle v \rangle_{i_\ell}$ for ℓ distinct indices $\{i_1, \dots, i_\ell\}$. If $\ell < t$, then v is not a heavy-hitter and the privacy of the t -out-of- n threshold sharing ensures that the aggregation server does not learn the corresponding input v reported by the clients. On the other hand, when $\ell > t$, the aggregation server possesses more than t shares with which it can reconstruct the heavy-hitter v .

However, the protocol as described does not ensure correctness, as the standard reconstruction algorithm for a t -out-of- n threshold secret sharing requires at least t shares from the *same* sharing instance. Specifically, this leads to two issues with the above protocol:

1. Clients with the same input do not report shares from the same instance.
2. Even if they did, the Aggregation Server receives only a list of shares and cannot determine which subset corresponds to the same value, making reconstruction infeasible.

STAR addresses the first issue by having the Randomness Server coordinate the randomness, so that clients with the same input generate correlated shares from the same sharing instance. In more detail, each client with input v obviously obtains randomness r_v associated with v —so that the Randomness Server does not learn v , and the client does not learn the randomness for other inputs³—and then uses r_v as the randomness to secret share v . Each client then sends the share at a distinct index to the Aggregation Server, ensuring that clients reporting the same value produce shares from the same sharing instance. To resolve the second issue, STAR requires each client to append a tag t_v , which the Randomness Server derives by applying a pseudorandom function to the client’s input v . Each client sends a report of the form $\text{rpt}_i = (\langle v \rangle_i, t_v)$ to the Aggregation Server, which then combines t reports with the same tag to reconstruct the heavy-hitters.

Reducing report leakage Unfortunately, the inclusion of the tags t_v leaks the complete histogram of client values. The first contribution of this work is an approach to removing this leakage from STAR. Rather than labeling each report, we turn to Multi-Dealer Secret Sharing (MDSS), originally described in [25]. MDSS extends traditional secret sharing by allowing for the simultaneous recovery of multiple secrets from a pool of unlabeled shares. In our setting this ensures that heavy-hitters can be extracted from a set of reports, without the need

³This can be done using an Oblivious Pseudorandom Function (OPRF) protocol.

to explicitly label reports that belong to a specific input value, and without revealing any information about non-heavy-hitter values. In principle, MDSS offers us a drop-in solution to the tag leakage problem: clients query the randomness server to obtain the secret-sharing parameters corresponding to their input, evaluate a single share, then send that share to the Aggregation Server without a tag. The Aggregation Server then runs the MDSS reconstruction algorithm over the entire pool of shares to learn the heavy hitters.

While MDSS can theoretically work as a drop-in solution, we stress that making this approach work efficiently in practice is non-trivial. One important concern is that existing MDSS constructions add a gap (or *ramp*) between the number of shares required for secret recovery and the number of shares that are guaranteed to protect privacy (throughout this work we refer to these thresholds as t_{rec} and t_{priv} respectively). This ramp can be minimized by increasing the number of polynomials c , although it cannot be practically eliminated. This requires some careful balancing of parameters, as well as a number of optimizations to ensure a practical level of efficiency. We describe these in [Section 5](#).

Addressing collusion. Both STAR and POPSTAR are extremely fragile in the event that an Aggregation Server is given the ability to issue even a single query to the Randomness Server. In either protocol, a single query on a candidate client input v will immediately reveal the tag corresponding to that input, which allows the Aggregation Server to identify all reports made for that input. Notably, this is a valid attack even in the *semi-honest* setting.

Our MDSS-based protocol partially alleviates this concern by simply removing the tag. Unfortunately, any protocol based on secret sharing still requires the Randomness Server to sample a distinct (pseudo)random polynomial for each distinct client input v . In STAR this is done in a two-step process: (1) the client and Randomness Server first conduct an oblivious PRF (OPRF) protocol on the client’s input and the Server’s key to obtain a random seed r_v , and then (2) this seed is used to sample the secret-sharing polynomial. This means that an Aggregation Server who issues a single client query on v to the Randomness Server will obtain the full polynomial associated with that input. This knowledge allows the Aggregation Server to identify all secret shares (and hence reports) that correspond to this value.

Addressing this problem requires a new means for clients to interact with the Randomness Server. At a high level, we require a 2PC protocol that deterministically maps a client input v to a seed value r_v , uses it to compute a secret sharing polynomial, and returns a single share to the client. While we could achieve this using generic two-party computation techniques, we would like to keep the Randomness Server interaction as lightweight as possible and generic 2PC introduces a significant overhead.

To overcome this hurdle we exploit the structure of MDSS

shares. An MDSS share is a point on a tuple of polynomials: $(x, p_1(x), \dots, p_c(x))$. Critically, the share algorithm consists only of sampling these polynomials uniformly at random, and then evaluating them on the same random point. We therefore give a protocol for *doubly oblivious polynomial evaluation* (DOPE) that maps an input to a seed, derives a polynomial, and evaluates that polynomial on a given point, all while keeping both parties ignorant of the intermediate steps.

To construct such a protocol, we use an additively homomorphic public key encryption scheme that allows for the oblivious sampling of ciphertexts. In such a scheme, parties can use a public key to sample valid ciphertexts without knowing the underlying plaintexts. The protocol then proceeds as follows. First, a client hashes their value v to obtain r_v . They then use the public key of the Randomness Server and r_v to obliviously sample t_{priv} ciphertexts $(c_1, \dots, c_{t_{\text{priv}}})$, with corresponding plaintexts $(m_1, \dots, m_{t_{\text{priv}}})$. The client then computes the polynomial evaluation $y = c_1x + c_2x^2 + \dots + c_{t_{\text{priv}}}x^{t_{\text{priv}}}$ for a value x . By the homomorphic property of the encryption scheme, the result is the encryption of $m_1x + m_2x^2 + \dots + m_{t_{\text{priv}}}x^{t_{\text{priv}}}$. Finally, the client samples a mask r^* , computes its encryption c^* , and sends $y + c^*$ to the Randomness Server. The Randomness Server then decrypts the ciphertext and sends the result to the client, who removes the mask. At the end of the protocol, the client is left with the evaluation of a polynomial for which they do not know any of the coefficients, and the Randomness Server learns nothing due to the single value it decrypts being masked. Knowledge of the value r_v is no longer useful to a malicious client, as it requires interaction with the Randomness Server to learn any information about additional shares.

There are several encryption schemes that are both (1) additively homomorphic and (2) allow for oblivious ciphertext sampling: these include the Paillier [36] and Goldwasser-Micali [27] schemes. We ultimately elect to use the Benaloh encryption scheme [11, 12], as its plaintext space aligns well with the share space required by MDSS. For more information see Section 4.1.

For our scheme, the interaction between the Randomness Server and client in the DOPE protocol remains lightweight. Notably, for each polynomial, the client’s DOPE message is a single Benaloh ciphertext, and the Server only performs a single decryption. Concretely, for the number of polynomials used in our work, both the client and Randomness Server computation can be performed in approximately 1 second. Total communication is in the range of 50 kilobytes. More detailed benchmarks can be found in Section 6.

2.1 Comparison to Prior Work

POPSTAR [31] is the state-of-the art for computing private heavy hitters in the STAR model, and we briefly describe the leakage induced by the scheme here. While the original STAR protocol reveals the client histogram, allowing the Aggrega-

tion Server to know exactly which reports are of the same value and which are not, the leakage in POPSTAR is akin to a *partial* revealing of the histogram. The Aggregation Server may learn the exact counts for some of the un-reconstructed values, but for others may only learn the *combined counts* of several values. For example, the Aggregation Server may learn that the total number of clients reporting some collection of values is 100, but it will be unable to learn how many values contribute to that collection, or which reports within it are for the same value. The reports that the aggregation server learns the most about are selected randomly, as determined by a short tag computed deterministically from the client’s initial randomness r_v .

More concretely, each value in POPSTAR is mapped to a short bit string ℓ , over which the server builds a prefix tree, revealing nodes in the tree as the counts of their parent node exceed the threshold t . At the start of aggregation, the server only learns the first bit of ℓ for each report. Then, if at least t reports have the prefix 0, the server learns the two bit prefix for all of those reports, and so on for all other prefixes that meet the threshold. When a given prefix ℓ has been fully revealed, STAR-like tags for reports with that prefix are revealed in turn, allowing the aggregation server to recover values as in STAR. The consequences of this are that some prefixes may not have over t reports associated with them, and therefore do not reveal how many values were mapped to that prefix. [31] refers to these prefixes and the data associated with them as *end nodes*. Prefixes with at least t associated reports reveal the *exact counts* of non-heavy-hitter values, as they return to a STAR-like tagging scheme.

MDSS-STAR leakage regime The protocol described in this work introduces a different leakage regime to the one considered in POPSTAR. Rather than leak *some* information about *every* report, we may leak *complete* information (including the underlying value) on *some* reports, those that exist in the gap between t_{priv} and t_{rec} . The scheme here guarantees privacy for any clients submitting rare values, meaning our approach could be valuable in applications where the confidentiality of such reports is paramount. We also note that the parameters of our scheme are customizable: the leakage can be reduced at the cost of increasing the runtime of the aggregation step. For further information about runtime and leakage calculations, see Section 5. For comparisons to other private aggregation systems, see Section 8.

3 Preliminaries

We use $[x, y]$ to denote the set $\{x, x+1, \dots, y\}$ where $x, y \in \mathbb{Z}$ and $x \leq y$. We use $[x]$ as a shorthand to denote $[1, x]$. We use $\mathbf{x} \in \mathcal{X}^\ell$ to denote a vector of ℓ elements in $\mathcal{X}$. For a vector $\mathbf{x} = (x_1, \dots, x_\ell)$, we use $\mathbf{x}^{(i)} = x_i$ to denote the i -th element in $\mathbf{x}$ where $i \in [\ell]$. We use $x||y$ to denote the concatenation of

two strings x and y . For any algorithm $\mathcal{A}$, we use $y := \mathcal{A}(x; r)$ to denote that the algorithm runs on input x and random tape r and produces output y . In most cases we do not denote the random tape explicitly and simply write $y \leftarrow \mathcal{A}(x)$. We use λ as the computational security parameter, $\text{negl}(\lambda)$ to denote negligible functions, and $\text{poly}(\lambda)$ to denote polynomials in λ . For two probability ensembles $\{A_i\}_i$ and $\{B_i\}_i$, we use $\{A_i\}_i \equiv \{B_i\}_i$ to denote that the ensembles are identical, $\{A_i\}_i \approx \{B_i\}_i$ to denote that the ensembles are statistically close and $\{A_i\}_i \approx_c \{B_i\}_i$ to denote that the ensembles are computationally indistinguishable. We use $\Pr[E : A]$ to denote the probability of an event E in an experiment defined by executing A . For a rank $k \in [N]$ and exponent $s > 0$, the Zipf power-law (or “Zipfian”) distribution [37] with parameters N , s is defined by

$$\Pr[X = k] = \frac{1/k^s}{H_{N,s}}, \quad \text{where} \quad H_{N,s} = \sum_{k=1}^N \frac{1}{k^s}.$$

$[N]$ is the support of the distribution, while the s parameter controls the concentration of values: the larger s is, the more concentrated the distribution is in the top ranks. We will sometimes abuse notation and refer to a Zipfian distribution with support N .

Security model We define and prove the security of our protocol in the global universal composability (GUC) framework [19], which extends the universal composability (UC) framework [18] to support global setup—allowing, for instance, modeling random oracles. We use $\mathcal{A}$ to denote the real world adversary, $\mathcal{S}$ for the ideal world adversary, and $\mathcal{Z}$ for the environment. We use S_{rand} and S_{agg} to denote the Randomness Server and Aggregation Server, respectively, and U to denote a user. We use $\mathcal{F}$ to denote (standard) ideal functionalities that are invoked by parties of the given protocol execution, and $\bar{\mathcal{F}}$ to denote global ideal functionalities which can be invoked by any party and even directly by the environment. All functionalities are standard PID-wise corruption unless stated otherwise. Our protocol is proved secure against a static, honest-but-curious adversary that corrupts either S_{rand} or S_{agg} , along with an arbitrary subset of users. We refer the reader to [18, 19] for further details on UC and GUC.

3.1 Multi-Dealer Secret Sharing

We will make use of a $(t_{\text{rec}}, t_{\text{priv}}, d, B)$ -multi-dealer-secret sharing scheme (MDSS). We use a weaker variant of MDSS than the one defined in [25]. Modifications are indicated in footnotes.

Definition 1 (Multi-Dealer Secret Sharing). A $(t_{\text{rec}}, t_{\text{priv}}, d, B)$ -multi-dealer secret sharing scheme (MDSS) with security parameter λ is parameterized by $(t_{\text{rec}}, t_{\text{priv}}, d, B)$ which are integer-valued polynomials in λ , is defined over

a secret space $\mathcal{H}$, and consists of the following algorithms, which are expected to run in time polynomial in λ .

- $\text{Share}(1^\lambda, s, n) \rightarrow \{\langle s \rangle_i\}_{i \in [n]}$, takes as input a secret $s \in \mathcal{H}$ and an integer n and outputs a set of n shares.
- $\text{Reconstruct}(\langle s \rangle_1, \dots, \langle s \rangle_w) =: \{s_1, \dots, s_m\}$, takes as input shares $\langle s \rangle_1, \dots, \langle s \rangle_w$ where $w \leq B$, and outputs a (potentially empty) set of secrets $\{s_1, \dots, s_m\}$.

We now define some notational shorthand. For a secret s , an integer n , and a list of unique indices $I \subseteq [n]$, define $\text{Proj}(s, n, I)$ as follows:

1. Compute $\{\langle s \rangle_1, \dots, \langle s \rangle_n\} \leftarrow \text{Share}(s, n)$.
2. Return $\{\langle s \rangle_i\}_{i \in I}$.

Definition 2 (Unlinkability). We say an MDSS scheme is ϵ -unlinkable if for all admissible adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$:

$$\Pr \left[\mathcal{A}(\mathbf{sh}'_1, \dots, \mathbf{sh}'_L) = b : \begin{cases} s_i, (n_{i,0}, I_{i,0}), \\ (n_{i,1}, I_{i,1}) \end{cases}_{i=1}^L \leftarrow \mathcal{A}(1^\lambda) \right. \\ \left. \begin{matrix} b \leftarrow \{0, 1\} \\ \forall i \in [L] : \mathbf{sh}_i \leftarrow \text{Proj}(s_i, n_{i,b}, I_{i,b}) \end{matrix} \right] \leq \frac{1}{2} + \epsilon(\lambda),$$

where $\mathcal{A}$ is *admissible* if it is non-uniform polynomial time and satisfies:

1. $\forall i \in [L], |I_{i,0}| \leq t_{\text{priv}}, |I_{i,1}| \leq t_{\text{priv}}$
2. $|I_{0,0}| \dots |I_{L,0}| = |I_{0,1}| \dots |I_{L,1}|$ ⁴

Definition 3 (MD-Correctness). We say that a $(t_{\text{rec}}, t_{\text{priv}}, d, B)$ -MDSS scheme is ϵ -correct if for all $\lambda \in \mathbb{N}$ and for all $\{(s_i, n_i, I_i)\}_{i \in [h]}$ where the following hold:

- $\forall i \in [h], I_i \subseteq [n_i]$
- $\sum_{i \in [h]} |I_i| \leq B$
- $|\{I_i \text{ s.t. } |I_i| \geq t_{\text{rec}}\}| \leq d$
- $\bigcap_{i,j \in [h] \text{ s.t. } i \neq j} I_i \cap I_j = \emptyset$ ⁵

It holds that:

$$\Pr \left[\begin{matrix} \text{Reconstruct}(S) \neq \\ \{s_i \text{ s.t. } |I_i| \geq t_{\text{rec}}\} \end{matrix} : S := \bigcup_{i=1}^h \text{Proj}(s_i, n_i, I_i) \right] \leq \epsilon(\lambda)$$

We will use a modified version of the MDSS construction from [25], shown in Figure 2. As in [25], we provide proofs that the construction show in Figure 2 satisfies unlinkability and *heuristically* satisfies md-correctness. Our proofs proceed similarly to the proofs in [25], and can be found in Appendix A.

⁴ [25] requires unlinkability as long as the total *number* of indices is the same across the two games, even if the indices themselves differ. In contrast, our definition only requires unlinkability when the set of indices is *identical*.

⁵ In contrast to [25], we require that correctness only hold when all indices are unique.

Construction MDSS _c	
Share(<i>s</i> , <i>n</i>)	
1:	Sample <i>c</i> polynomials $p_1, \dots, p_c$, each of degree $t_{\text{priv}} + 1$ and with leading coefficient 1, where $s = p_1(0) \dots p_c(0)$
2:	return $\{(i, p_1(i), \dots, p_c(i))\}_{i \in [n]}$
Reconstruct($sh_1, \dots, sh_w$)	
1:	return CH-MDSS($sh_1, \dots, sh_w$)

Figure 2: MDSS Construction. For a full description of the CH-MDSS algorithm see [25].

Theorem 1. *For all $c \geq 1$ and finite fields $\mathbb{F}$, the construction MDSS_c shown in Figure 2 satisfies ϵ -unlinkability for $\epsilon = 0$.*

Theorem 2 (Informal). *For all sets of parameters satisfying $t_{\text{rec}} \geq \frac{1}{1+c}(B + (c \cdot (t_{\text{priv}} + 1)))$ and finite fields $\mathbb{F}$, the construction shown in Figure 2 heuristically satisfies ϵ -correctness for a negligibly small $\epsilon(\lambda)$.*

For a formalization of the heuristic assumption underlying Theorem 2, which is a slight modification of the one used in [25], see Appendix A.

4 Security Analysis

In this section we formally define our ideal functionalities, which capture our leakage profile and correctness guarantees, and provide protocols that achieve them.

The MDSS-STAR protocol, which we denote by $\Pi_{\text{report}}^{t_{\text{rec}}, t_{\text{priv}}, B}$, is detailed in Figure 11 and the ideal functionality it realizes is described in Figure 10. The protocol is described in the $(\mathcal{F}_{\text{oss}}^{\text{MDSS}}, \mathcal{F}_{\text{mix}}^B)$ -hybrid model. $\mathcal{F}_{\text{oss}}^{\text{MDSS}}$ (Figure 7) is an ideal functionality that allows clients to obliviously sample an MDSS share, which will subsequently serve as the client’s report. The $\mathcal{F}_{\text{mix}}^B$ functionality aggregates and permutes client reports before forwarding them to the Aggregation Server, thereby ensuring that the reports received by the Aggregation Server cannot be linked to the identities of the clients who generated them.

We note that MDSS-STAR can be instantiated with any MDSS scheme that meets the requirements outlined in Section 3.1; the $\mathcal{F}_{\text{oss}}^{\text{MDSS}}$ functionality abstracts away the details of the MDSS scheme and captures the fact that the protocol only requires a way for clients to obliviously sample a share. We present an efficient protocol $\Pi_{\text{oss}}^{\text{MDSS}_c}$ in Figure 8 that realizes this functionality for the MDSS scheme described in Figure 2. To simplify the presentation, we present $\Pi_{\text{oss}}^{\text{MDSS}_c}$ in the $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$ -hybrid model, where $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$ (Figure 3) is an ideal functionality for doubly-oblivious polynomial evaluation (DOPE). Since under the MDSS construction in Figure 2 a share is the evaluation of several polynomials, $\Pi_{\text{oss}}^{\text{MDSS}_c}$ will mostly

be a pass-through to $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$. The $\Pi_{\text{dope}}^{\mathcal{E}, \ell}$ protocol described in Figure 4 realizes $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$.

Our doubly-efficient polynomial evaluation protocol can be seen in Figure 4, our oblivious share sampling protocol can be seen in Figure 8 and our secure aggregation protocol MDSS-STAR can be seen in Figure 11.

We now provide theorems and proofs of our protocols’ security.

4.1 DOPE Protocol

Functionality $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$	
This functionality is parameterized by a field $\mathbb{F}$ and an integer ℓ . For a field $\mathbb{F}$, integer ℓ , and field element $v \in \mathbb{F}$, let $\mathcal{P}_{\mathbb{F}, \ell, v}$ be the set of all polynomials over $\mathbb{F}$ of degree ℓ with leading coefficient 1 where the constant term is v .	
Ignore all messages associated with <i>sid</i> before receiving (<i>init</i> , <i>sid</i>) from S_{rand} .	
1:	Upon receiving (<i>init</i> , <i>sid</i>) from S_{rand} , initialize an empty table <i>T</i> and store <i>T</i> .
2:	Upon receiving (<i>eval</i> , <i>sid</i> , <i>v</i> , <i>x</i>) from a user <i>U</i> , send (<i>eval</i> , <i>sid</i> , <i>U</i>) to $\mathcal{S}$. Then, if $v \notin T$, sample $p \leftarrow \mathcal{P}_{\mathbb{F}, \ell, v}$ and set $T[v] := p$. Finally, retrieve $p := T[v]$, and return $p(x)$ to <i>U</i> .
3:	Upon receiving (<i>observe</i> , <i>sid</i> , <i>v</i>) from $\mathcal{S}$, ignore the message if S_{rand} is not corrupt. Else, retrieve <i>T</i> and if $v \notin T$, sample $p \leftarrow \mathcal{P}_{\mathbb{F}, \ell, v}$ and set $T[v] := p$. Finally, retrieve $p := T[v]$, and return <i>p</i> to $\mathcal{S}$.

Figure 3: Ideal functionality for DOPE.

The ideal functionality for doubly-oblivious polynomial evaluation is presented in Figure 3 and the corresponding protocol is described in Figure 4. Our protocol uses an encryption scheme with certain key properties. We first define the encryption scheme and its properties and then proceed to describe the protocol.

Encryption scheme We will use a *linearly homomorphic* IND-CPA secure public-key encryption scheme $\mathcal{E} = (\text{Gen}, \text{Enc}, \text{Dec})$ with message space $\mathcal{M}$, key space $\mathcal{K}$, and ciphertext space $\mathcal{C}$ ⁶. We require that the ciphertext space is *dense*—that is, for any public key *pk*, a uniformly random element of the ciphertext space $\mathcal{C}$ is a valid ciphertext under *pk* and corresponds to an encryption of a uniformly random

⁶Formally, these are a sequence of message, key and ciphertext spaces indexed by the security parameter λ , but we abuse notation and directly refer to the λ -th element of the sequences.

Protocol $\Pi_{\text{dope}}^{\mathcal{E}, \ell}$

Let $\mathbb{F}$ be a finite field. Let $\mathcal{E}$ be a linearly homomorphic (Definition 4) public-key encryption scheme with dense ciphertext space (Definition 5), and with message space $\mathbb{F}$.

When a user U is activated with input $(\text{eval}, \text{sid}, v, x)$, it proceeds as follows:

- 1: U invokes $\overline{\mathcal{F}}_{\text{setup}}$ to receive pk . Then, for $i \in [\ell]$, it sends $(\text{query}, \text{sid}, v \| i)$ to $\overline{\mathcal{F}}_{\text{ro}}$ and receives ct_i .
- 2: U samples $\alpha \leftarrow \mathbb{F}$ uniformly at random and computes $\text{ct}_0 \leftarrow \mathcal{E}.\text{Enc}(\text{pk}, v + \alpha)$.
- 3: U computes

$$\text{ct} := \mathcal{E}.\text{Eval}\left(\text{pk}, (\text{ct}_0, \dots, \text{ct}_\ell), (1, x, \dots, x^\ell)\right)$$

and sends ct to S_{rand} .

- 4: Upon receiving ct from U , S_{rand} invokes $\overline{\mathcal{F}}_{\text{setup}}$ to receive (pk, sk) , computes $\gamma := \mathcal{E}.\text{Dec}(\text{sk}, \text{ct})$, and sends γ to U .
- 5: Upon receiving γ from S_{rand} , U computes $y := \gamma - \alpha$ and outputs y .

Figure 4: Protocol for Doubly-Oblivious Polynomial Evaluation.

message. We omit the standard definition of IND-CPA security (see, e.g., [29, Definition 11.5]) and instead define linear homomorphism and dense ciphertext space below.

Definition 4 (Linear Homomorphism). Let $\mathcal{E}$ be a public-key encryption scheme such that the message space $\mathcal{M}$ is a ring. $\mathcal{E}$ is said to be *linearly homomorphic* if there exists a polynomial time algorithm $\mathcal{E}.\text{Eval}$ such that for all integer-valued polynomials $\ell := \ell(\lambda)$, all $\lambda \in \mathbb{N}$, and all $\mathbf{m}, \mathbf{v} \in \mathcal{M}^\ell$ we have

$$\Pr \left[\mathcal{E}.\text{Dec}(\text{sk}, \mathcal{E}.\text{Eval}(\text{pk}, \mathbf{ct}, \mathbf{v})) = \sum_{i=1}^{\ell} \mathbf{v}^{(i)} \cdot \mathbf{m}^{(i)} \right] = 1$$

where $(\text{pk}, \text{sk}) \leftarrow \mathcal{E}.\text{Gen}(1^\lambda)$, each $\mathbf{ct}^i \leftarrow \mathcal{E}.\text{Enc}(\text{pk}, \mathbf{m}^{(i)})$, and the probability is over the randomness used in Gen and Enc .

Definition 5 (Dense Ciphertext Space). A public-key encryption scheme $\mathcal{E}$ is said to have *dense ciphertext space* if there exists a PPT algorithm to sample uniformly random elements from $\mathcal{C}$ and if for all non-uniform polynomial time adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, we have

$$\Pr \left[\mathcal{A}^{O_b}(\text{pk}, \text{sk}) = b : \begin{array}{l} (\text{pk}, \text{sk}) \leftarrow \mathcal{E}.\text{Gen}(1^\lambda) \\ b \leftarrow \{0, 1\} \\ \leq \frac{1}{2} + \text{negl}(\lambda), \end{array} \right]$$

where O_b samples $\text{ct} \leftarrow \mathcal{C}$ uniformly at random and returns ct , while O_1 samples $m \leftarrow \mathcal{M}$ uniformly at random, computes $\text{ct} \leftarrow \mathcal{E}.\text{Enc}(\text{pk}, m)$, and returns ct .

Several encryption schemes—such as Paillier [36], Damgård-Jurik [21], Goldwasser-Micali [27], and Benaloh [12]—are linearly homomorphic and have dense ciphertext space. In this work, we use the Benaloh encryption scheme, which we describe in Appendix B. We use this scheme as it supports having a polynomial order finite field as the message space, which is what is required by the MDSS scheme described in Figure 2.

Protocol We describe the DOPE protocol $\Pi_{\text{dope}}^{\mathcal{E}, \ell}$ in Figure 4. The protocol uses a setup functionality $\overline{\mathcal{F}}_{\text{setup}}^{\mathcal{E}}$ (Figure 5) and a random oracle $\overline{\mathcal{F}}_{\text{ro}}^{\mathcal{C}}$ (Figure 6). Informally, the setup functionality provides the public key of the Randomness Server to all clients, while the random oracle helps non-interactively sample random ciphertexts.

Functionality $\overline{\mathcal{F}}_{\text{setup}}^{\mathcal{E}}$

This functionality is parameterized by a public-key encryption scheme $\mathcal{E}$.

On the first activation, run $(\text{pk}, \text{sk}) \leftarrow \mathcal{E}.\text{Gen}(1^\lambda)$ and store (pk, sk) . Upon activation by a user U , return pk to U . Upon activation by S_{rand} , return (pk, sk) to S_{rand} .

Figure 5: Shared setup functionality.

Functionality $\overline{\mathcal{F}}_{\text{ro}}^{\mathcal{C}}$

This functionality is parameterized by a set $\mathcal{C}$ and an algorithm to sample elements uniformly at random from $\mathcal{C}$.

Upon first activation, initialize an empty table T . Upon receiving $(\text{query}, \text{sid}, x)$ from a party, if $(\text{sid}, x) \notin T$, sample $\text{ct} \leftarrow \mathcal{C}$ uniformly at random, and set $T[\text{sid}, x] := \text{ct}$. Return $T[\text{sid}, x]$ to the activating party.

Figure 6: Shared random oracle functionality.

Lemma 3. Let $\mathbb{F}$ be a finite field and let $\mathcal{E}$ be any linearly homomorphic (Definition 4) public-key encryption scheme with dense ciphertext space $\mathcal{C}$ (Definition 5) and with message space $\mathbb{F}$. Then, for any integer-valued polynomial $\ell := \ell(\lambda)$, $\Pi_{\text{dope}}^{\mathcal{E}, \ell}$ (Figure 4) securely realizes $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$ (Figure 3) in the $(\overline{\mathcal{F}}_{\text{setup}}^{\mathcal{E}}, \overline{\mathcal{F}}_{\text{ro}}^{\mathcal{C}})$ -hybrid model.

The proof for the above lemma can be found in [Appendix C](#).

4.2 OSS Protocol

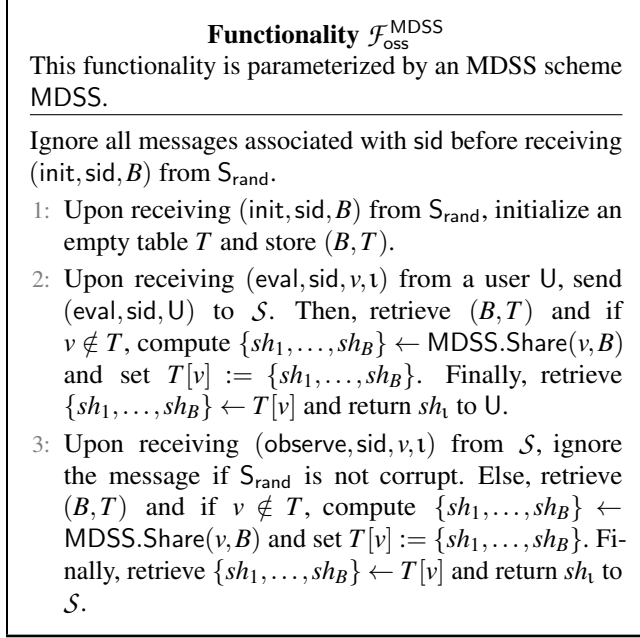


Figure 7: Ideal functionality for Oblivious Share Sampling (OSS)

As discussed earlier, the OSS functionality, described in [Figure 7](#), allows a client to sample a single MDSS share of its input, with the help of the Randomness Server. The OSS protocol presented in [Figure 8](#) uses the DOPE functionality ([Figure 3](#)) to realize oblivious share sampling for the MDSS scheme from [Figure 2](#).

Theorem 4. *For all positive integers c , the protocol $\Pi_{\text{oss}}^{\text{MDSS}_c}$ ([Figure 8](#)) UC-realizes the functionality $\mathcal{F}_{\text{oss}}^{\text{MDSS}_c}$ in the $(\mathcal{F}_{\text{dope}}^{\mathbb{F}, t_{\text{priv}}+1})$ -hybrid model.*

The proof of [Theorem 4](#) follows directly from UC-security under parallel composition.

4.3 Report Protocol

The ideal functionality for our overall scheme can be seen in [Figure 10](#). The honest interface supports clients reporting their values v , with the interface totaling the count of each. Honest collection by the aggregation server returns all values which have been reported at least t_{rec} times, along with the number of times each has been reported. The corrupt interface returns values with counts that meet the lower t_{priv} threshold instead.

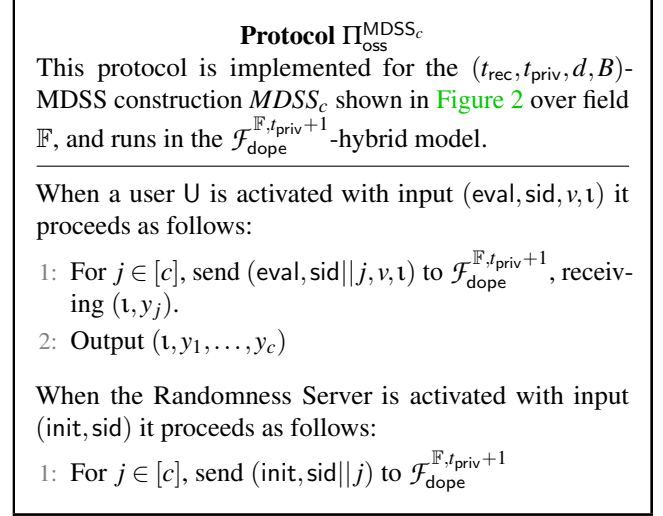


Figure 8: Protocol for Oblivious Share Sampling for Construction MDSS_c .

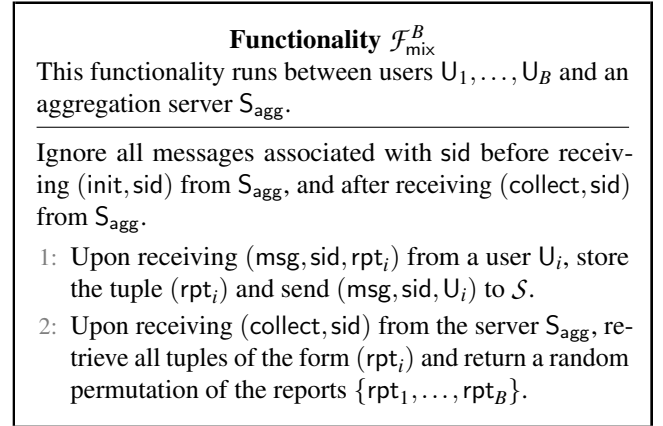


Figure 9: Ideal functionality for Mixing

Theorem 5. *For all $(d, t_{\text{rec}}, t_{\text{priv}}, B)$ -MDSS schemes MDSS that satisfy ϵ -unlinkability for a negligible $\epsilon(\lambda)$, the protocol $\Pi_{\text{report}}^{t_{\text{rec}}, t_{\text{priv}}, B}$ ([Figure 11](#)) UC-realizes the functionality $\mathcal{F}_{\text{report}}^{t_{\text{rec}}, t_{\text{priv}}, B}$ in the $(\mathcal{F}_{\text{oss}}^{\text{MDSS}}, \mathcal{F}_{\text{mix}}^B)$ -hybrid model, in the presence of semi-honest adversaries who statically corrupt at most one of the servers $S_{\text{agg}}, S_{\text{rand}}$ and any number of clients.*

A proof of [Theorem 5](#) can be found in [Appendix C](#).

5 Implementation

We now describe the implementation of the protocol given in [Figure 11](#). We instantiate the linearly homomorphic encryption scheme with dense ciphertext space required by $\Pi_{\text{dope}}^{E, t_{\text{priv}}}$ with the Benaloh encryption scheme described in [Appendix B](#).

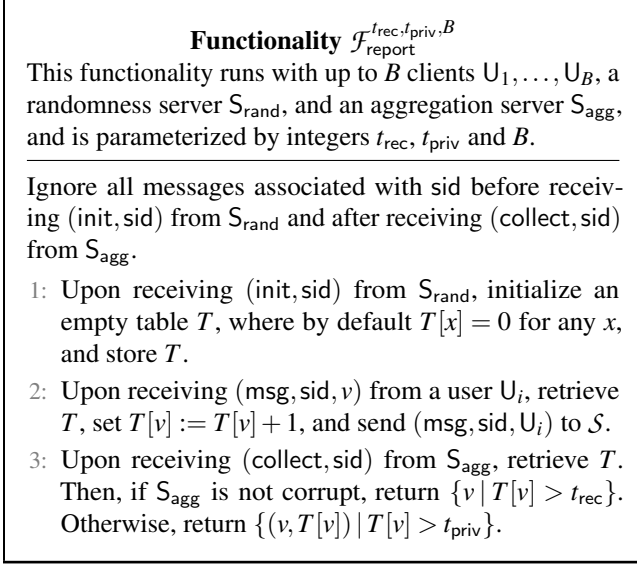


Figure 10: Ideal functionality for MDSS-STAR

We implement all algorithms in C++ using the NTL [40] library. We additionally make a number of optimizations to the $\text{MDSS}_c.\text{Reconstruct}$ algorithm, which we describe below. All code can be seen at <https://github.com/heldridge/secure-aggregation-monorepo/>.

5.1 Optimizing Reconstruction

As noted in [16], metrics of interest often follow a Zipf power-law distribution. As such, [16], as well as both STAR [22] and POPSTAR [31], benchmark their implementations against Zipf-distributed client reports. Assuming such a distribution, we can speed up the aggregation step via *subsampling*, running $\text{MDSS}_c.\text{Reconstruct}$ on only a randomly sampled subset of submitted reports rather than the entire pool. Because the reported values follow a Zipfian distribution, the k -heavy-hitters should be overrepresented in this subset, allowing us to recover them with high probability. We note that this approach makes black-box use of the underlying MDSS reconstruction algorithm, and does not require any changes from its original description in [25].

We can additionally push this technique further via *iterative* aggregation. The runtime of $\text{MDSS}_c.\text{Reconstruct}$ scales with the number of polynomial evaluations used (c), as well as the number of reports in its input (referred to as a *batch size*, or bs). Therefore, we attempt to minimize the runtime by using the smallest values of c and bs required to recover each heavy-hitter. For the heavy-hitter of rank 1, i.e. the most prevalent reported value, we can use low values of c and bs when aggregating. Then, after recovering this value (along with the polynomials associated with it), we can remove all reports for it from the input, making each other value more

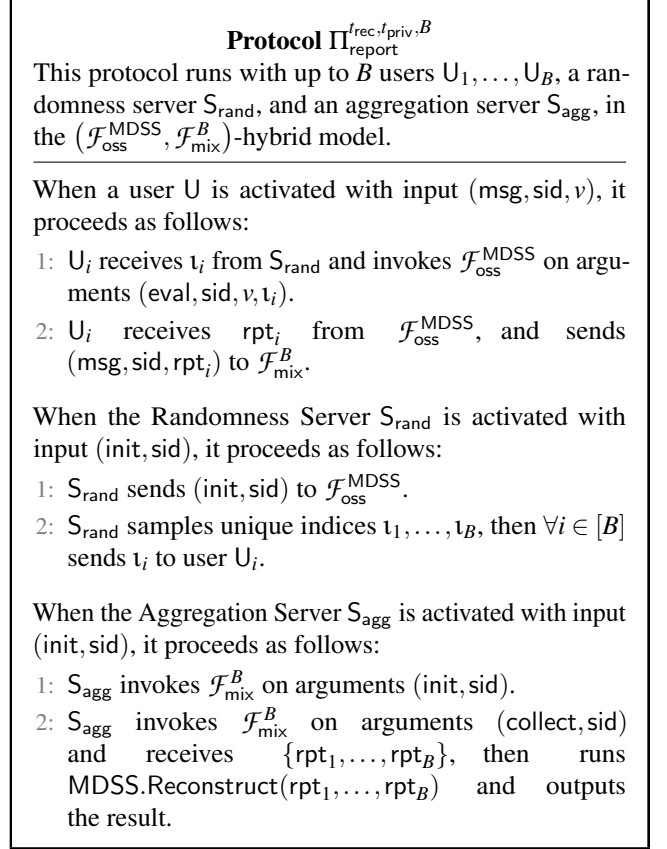


Figure 11: Protocol for MDSS-STAR

prevalent (and thereby more likely to appear in the subsample), and lowering the required c and bs to successfully recover the heavy-hitter of rank 2, and so on. We additionally set a final configuration to recover all values over the threshold, and continue to use it to decode until we recover all heavy-hitters.

The exact numbers used for bs and c will vary between deployments. The approach we use in this work is to use a Monte Carlo simulation to find the values of bs and c that minimize $bs \times c$ while ensuring a $\geq 99\%$ chance of success in each aggregation iteration. Further details on our experimental setup can be found in Section 6.

Finally, the CH-MDSS algorithm described in [25] aggregates by performing lattice reductions on polynomial matrices where each reduction produces a canonical reduced basis known in literature as the *Popov form*. We find that relaxing the canonical condition on the reduced basis to *weak Popov form* results in a measurably faster decoding algorithm for our input distributions while retaining (heuristic) correctness.

Algorithm 1: STEP-DECODE

Input: A set of shares Z , a set of values $\mathcal{V}$, and integers $t_{\text{rec}}, t_{\text{priv}}, \text{bs}_i, c_i$
Output: A new set of shares Z' , a new set of values $\mathcal{V}'$, and a flag *failed*
 $z'_1, \dots, z'_{\text{bs}_i} \leftarrow \text{Sample}(Z, \text{bs}_i)$
 $t'_i := \lceil \frac{1}{c_i+1} (\text{bs}_i + (c_i \cdot (t_{\text{priv}} + 1))) \rceil$
 $\text{out} \leftarrow \text{CH-MDSS.Step}(t'_i, t_{\text{priv}}, \text{bs}_i, \{z'_1, \dots, z'_{\text{bs}_i}\})$
if $\text{out} = \{p_{c_1}, \dots, p_{c_i}\}$ **then**
 Let $\mathcal{P}$ be the set of all points in Z that agree with $\{p_{c_1}, \dots, p_{c_i}\}$
 if $|\mathcal{P}| < t_{\text{rec}}$ **then**
 return $Z, \mathcal{V}, \text{true}$
 end
 else
 Using $\mathcal{P}$, interpolate the polynomials $p_1, \dots, p_c$
 $v_i := p_1(0) \parallel \dots \parallel p_c(0)$
 $\mathcal{V}' := \mathcal{V} \cup \{v_i\}$
 $Z' := Z \setminus \mathcal{P}$
 return $Z', \mathcal{V}', \text{false}$
 end
end
else
 return $Z, \mathcal{V}, \text{true}$
end

Algorithm 2: MDSS_c.Reconstruct

Data: Configuration parameters $(\text{bs}_1, c_1), \dots, (\text{bs}_l, c_l)$, and integers $t_{\text{rec}}, t_{\text{priv}}$
Input: Shares $z_1, \dots, z_w$
Output: A set of values $\mathcal{V}$
Initialize $Z := \{z_1, \dots, z_w\}$
Initialize an empty set $\mathcal{V}$
Initialize *failed* := **false**
for $i \in [l - 1]$ **do**
 $Z, \mathcal{V}, \text{failed} := \text{STEP-DECODE}(Z, \mathcal{V}, t_{\text{priv}}, \text{bs}_i, c_i)$
 if *failed* **then**
 break
 end
end
while not failed do
 $Z, \mathcal{V}, \text{failed} := \text{STEP-DECODE}(Z, \mathcal{V}, t_{\text{priv}}, \text{bs}_l, c_l)$
end
return $\mathcal{V}$

5.1.1 Putting it All Together

We are now ready to describe our complete algorithm. Let CH-MDSS.Step be defined as running Algorithm 1 of [25] with the alteration that LatticeReduce puts the matrix in weak rather than full Popov form. Our iterative aggregation algorithm MDSS_c.Reconstruct can then be seen in Algorithm 2, where Sample(Z, n) indicates sampling n elements from a set Z without replacement. A reproduction of [25] Algorithm 1 with notational alterations can be found in Appendix D.

6 Experiments

All benchmarks were run on an AWS c7i.8xlarge EC2 instance.

6.1 Report Generation

Modulus	Client Runtime	Server Runtime	Client Message	Server Message
2048	1.44s	0.22s	35.73 kb	0.42 kb
3072	2.72s	0.71s	53.52 kb	0.42 kb

Figure 12: Report generation runtimes for $c = 139$ and $t_{\text{priv}} = 349$. *Modulus* indicates the length in bits of the modulus used for the Benaloh encryption scheme. *Client Message* is the size of the message sent from each client to the Randomness Server. *Server Message* is the size of the Randomness Server’s response. All runtimes are single-threaded.

The benchmarks for our report generation protocol can be seen in Figure 12. The benchmarks presented are for simultaneously evaluating $c = 139$ polynomials with degree 350, which corresponds to the parameters for MDSS-STAR that result in leakage comparable to POPSTAR. Our protocol is efficient, with client and server computation on the order of seconds, and bandwidth in the range of 50 kilobytes. The benchmarks shown in Figure 12 are all single threaded. When up to 4 threads are used, client computation drops to 0.37 and 0.67 seconds for the 2048 and 3072-bit moduli, respectively. The main computational bottleneck is the homomorphic evaluation over the ciphertexts.

Our protocol does introduce additional computational overhead as compared to STAR/POPSTAR, both of which have single-threaded client and server runtimes under 3 ms. However, we believe that such overhead is justified in achieving robustness against a colluding aggregation server.

Finally, POPSTAR requires at least 363 kb of communication between the client and Randomness Server, meaning the protocol here represents an improvement in this aspect.

6.2 Aggregation

As in STAR and POPSTAR, we simulate client inputs by drawing from a Zipfian distribution with support $N = 10,000$ and parameter $s = 1.03$. We begin by running the aggregation algorithm over sampled instances with number of reports $B = 100,000$ and field size $|\mathbb{F}| = 100,003$.

For these instances we first set the threshold $t_{\text{rec}} = 1000$ (i.e. 1% of the total number of reports). When POPSTAR aggregates 100,000 client reports at threshold 1000, it leaks (in expectation) 19 exact counts, along with 175 end nodes. We therefore set t_{priv} so that our protocol (in expectation) leaks 19 values, which results in $t_{\text{priv}} = 349$. We then run additional experiments, both with $t_{\text{rec}} = 2000$ and with other values of t_{priv} , demonstrating how the runtime of our aggregation algorithm decreases as we either admit more leakage or reduce the number of heavy hitters it is required to recover.

Our aggregation algorithm MDSS_c.Reconstruct requires global parameters (bs_i, c_i) to use at each iterative aggregation step. To determine these parameters, for a given $(t_{\text{rec}}, t_{\text{priv}})$, we use a Monte Carlo simulation to find the values of (bs_i, c_i) that minimize $bs_i \cdot c_i$ while retaining a $\geq 99\%$ chance of success at each aggregation iteration. We note that tuning parameters in this way does not require exact knowledge of the Zipf s parameter, but only a *lower bound* on it. Larger-than-expected values of s cause heavy-hitters to become more concentrated, and therefore increase the chance of decoding success. The parameters used for the case when $t_{\text{priv}} = 349$ and $t_{\text{rec}} = 1000$ can be seen in Figure 13.

Iteration	bs_i	c_i
1	8000	14
2	12000	32
3	16000	49
4	22000	59
5	28000	69
6	32000	83
7	39000	89
8	41000	107
9	45000	120
10	47000	136
11	53000	125
12	42000	139

Figure 13: Example Aggregation Parameters for $B = 100,000$, $t_{\text{priv}} = 350$, and $t_{\text{rec}} = 1000$.

The results of these benchmarks can be seen in Figure 14. Due to the costly lattice reductions, our aggregation requires significantly more computation as compared to prior work: POPSTAR aggregates 100,000 client reports in a matter of seconds, while our aggregation step is on the order of hours. However, we believe that our benchmarks do show that our aggregation algorithm remains practical, and could be deployed in particularly privacy-sensitive settings.

t_{priv}	t_{rec}	Leaked Values (Expected)	Aggregation Runtime
349	1000	19	9 hours, 34 minutes
251	1000	31	5 hours, 52 minute
201	1000	41	4 hours, 53 minutes
169	1000	51	3 hours, 17 minutes
420	2000	20	1 hour, 38 minutes
289	2000	31	58 minutes
224	2000	41	42 minutes
183	2000	51	33 minutes

Figure 14: Aggregation runtimes for $B = 100,000$. Computations were parallelized over 32 threads.

B	Leaked Values (Expected)	t_{rec}	Aggregation Runtime
100,000	19	1000 10,000	9 hr, 34 min 3 min
200,000	50	2000 10,000	9 hr, 24 min 9 min
400,000	100	4000 10,000	10 hr, 21 min 41 min
500,000	125	5000 10,000	12 hr, 34 min 1 hr, 22 min
600,000	150	6000 10,000	10 hr, 54 min 2 hr, 23 min
800,000	200	8000 10,000	8 hr, 26 min 5 hr, 37 min
1,000,000	250	10,000	13 hr, 3 min

Figure 15: Aggregation runtimes and leaked values across client counts with $t_{\text{rec}} \in \{\frac{B}{100}, 10000\}$. Computations were parallelized over 32 threads.

As the number of client reports increases, one must again either admit an increase either in runtime or leaked values. We demonstrate this with a series of benchmarks for values of B ranging between 100,000 and 1,000,000, the same range considered in STAR and POPSTAR. While STAR considers thresholds equal to 1%, 0.1%, and 0.01% of B , POPSTAR looks at the exact thresholds 100, 1000, and 10,000. We find that our construction is only suitable for thresholds at the high end of these ranges, namely 1% of B and 10,000. In each trial we set t_{priv} to keep the runtime close to 12 hours for the 1% case, admitting increases in the number of leaked values as the client count climbs. We then keep t_{priv} constant as we increase t_{rec} , demonstrating how runtime falls as the decoder is required to recover fewer values. The field size is set to the smallest prime greater than B . The results of these benchmarks can be seen in Figure 15.

The tables shown in Figure 14 and Figure 15 demonstrate how our protocol offers a three-way tradeoff between privacy, correctness, and efficiency. In Figure 14, runtime falls as

either privacy is reduced (by reducing t_{priv}), or fewer values are recovered (increasing t_{rec}). In Figure 15, we keep runtime somewhat constant in the face of larger instances by reducing privacy, and demonstrate efficiency gains by increasing the recovery threshold.

7 Discussion

Our t -heavy-hitter construction introduces a tradeoff between correctness, privacy, and efficiency. As one attempts to increase privacy, one must in turn either accept a longer aggregation step, or the recovery of fewer heavy-hitters. Concretely, this tradeoff appears in the “gap” between t_{rec} and t_{priv} . This leakage paradigm diverges from previous works in which there is a single threshold t , values with counts above t are considered heavy-hitters, values below t are private, and there is some leakage of the value histogram. In order to facilitate comparison, we offer here two possible interpretations of our leakage model.

Perfect Correctness, Some Leakage In the perspective that we have taken so far throughout this paper, we can view the MDSS gap as leakage. In this case, we consider $t_{\text{rec}} = t$, i.e. our recovery threshold as the equivalent to the overall threshold in prior work. Then, as we lose the MDSS privacy guarantees after a value has been reported over t_{priv} times, we say that we possibly leak values after t_{priv} reports.

We believe that this leakage profile is preferable to the profiles of prior work, as it guarantees privacy for values with low numbers of reports. Indeed, values incurring privacy loss only if a sufficient number of users report them has an intuitive appeal as a guarantee for a t -heavy-hitters protocol.

Perfect Security, Partial Correctness Alternatively, one could consider $t_{\text{priv}} = t$. In this case we achieve *perfect* security, with all values with counts under t_{priv} remaining private due to the unlinkability of the underlying MDSS scheme. However, we are not fully correct, as we fail to recover all values reported between t_{priv} and t_{rec} times. Using the parameters from the first row of Figure 14, this would mean we fail to recover 19 of the heavy-hitters.

8 Related Works

Private t -heavy-hitters, and privacy preserving secure aggregation in general, has a long line of work that can be broadly generalized into a few categories. Works such as [6, 9, 10, 15, 30, 32, 33] propose protocols in the *single server* model, where clients communicate with a single aggregation server. These protocols often require multiple rounds of communication, and/or communication between the clients themselves. In a parallel line of work, [1, 3, 5, 16, 20, 23, 34, 35, 39, 42] operate in the *multi-server* model, where clients split their reports

between a collection of servers that perform aggregation via joint computation, and requires some subset of the servers to remain honest in order to preserve privacy. A third line of work [7, 8, 13, 17, 38] protects clients using differential privacy [24]. While differential privacy offers attractive guarantees for clients, these works rely on the service provider to accurately configure and spend their privacy budget, which has a history of failure in practice [41].

References

- [1] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22: 13th International Conference on Security in Communication Networks*, volume 13409 of *Lecture Notes in Computer Science*, pages 516–539, Amalfi, Italy, September 12–14, 2022. Springer, Cham, Switzerland.
- [2] Apple and Google. Exposure notification privacy-preserving analytics (enpa) white paper. 2021.
- [3] Borja Balle, James Bell, Albert Cheu, Adria Gascon, Jonathan Katz, Mariana Raykova, Phillipp Schoppmann, and Thomas Steinke. Hash-prune-invert: Improved differentially private heavy-hitter detection in the two-server model. *Cryptology ePrint Archive*, Report 2024/2024, 2024.
- [4] Borja Balle, James Bell, Albert Cheu, Adria Gascon, Jonathan Katz, Mariana Raykova, Phillipp Schoppmann, and Thomas Steinke. Hash-prune-invert: Improved differentially private heavy-hitter detection in the two-server model. *Cryptology ePrint Archive*, Paper 2024/2024, 2024.
- [5] Laasya Bangalore, Albert Cheu, and Muthuramakrishnan Venkitasubramaniam. Prime: Differentially private distributed mean estimation with malicious security. *Cryptology ePrint Archive*, 2024.
- [6] Laasya Bangalore, Mohammad Hossein Faghihi Sereshgi, Carmit Hazay, and Muthuramakrishnan Venkitasubramaniam. Flag: A framework for lightweight robust secure aggregation. In Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik, editors, *ASIACCS 23: 18th ACM Symposium on Information, Computer and Communications Security*, pages 14–28, Melbourne, VIC, Australia, July 10–14, 2023. ACM Press.
- [7] Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Guha Thakurta. Practical locally private heavy hitters. *Advances in Neural Information Processing Systems*, 30, 2017.

- [8] Raef Bassily and Adam D. Smith. Local, private, efficient protocols for succinct histograms. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 127–135, Portland, OR, USA, June 14–17, 2015. ACM Press.
- [9] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. ACORN: Input validation for secure aggregation. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 4805–4822, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.
- [10] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1253–1269, Virtual Event, USA, November 9–13, 2020. ACM Press.
- [11] Josh Benaloh. Verifiable secret-ballot elections, 1987.
- [12] Josh Benaloh. Dense probabilistic encryption. In *Proceedings of the Selected Areas in Cryptography*, SAC ’94, 1994.
- [13] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th symposium on operating systems principles*, pages 441–459, 2017.
- [14] Jonas Böhler and Florian Kerschbaum. Secure multi-party computation of differentially private heavy hitters. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2361–2377, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.
- [15] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1175–1191, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.
- [16] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.
- [17] Mark Bun, Jelani Nelson, and Uri Stemmer. Heavy hitters and the structure of local privacy. *ACM Transactions on Algorithms (TALG)*, 15(4):1–40, 2019.
- [18] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.
- [19] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85, Amsterdam, The Netherlands, February 21–24, 2007. Springer Berlin Heidelberg, Germany.
- [20] Henry Corrigan-Gibbs and Dan Boneh. Prio: private, robust, and scalable computation of aggregate statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI’17, page 259–282, USA, 2017. USENIX Association.
- [21] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, Cheju Island, South Korea, February 13–15, 2001. Springer Berlin Heidelberg, Germany.
- [22] Alex Davidson, Peter Snyder, E. B. Quirk, Joseph Genereux, Benjamin Livshits, and Hamed Haddadi. STAR: Secret sharing for private threshold aggregation reporting. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 697–710, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.
- [23] Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. Verifiable distributed aggregation functions. Cryptology ePrint Archive, Report 2023/130, 2023.
- [24] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284, New York, NY, USA, March 4–7, 2006. Springer Berlin Heidelberg, Germany.

- [25] Harry Eldridge, Gabrielle Beck, Matthew Green, Nadia Heninger, and Abhishek Jain. Abuse-resistant location tracking: Balancing privacy and safety in the offline finding ecosystem. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.
- [26] Tim Geoghegan, Christopher Patton, Brandon Pitman, Eric Rescorla, and Christopher A. Wood. Distributed Aggregation Protocol for Privacy Preserving Measurement. Internet-Draft draft-ietf-ppm-dap-13, Internet Engineering Task Force, November 2024. Work in Progress.
- [27] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th Annual ACM Symposium on Theory of Computing*, pages 365–377, San Francisco, CA, USA, May 5–7, 1982. ACM Press.
- [28] Pranav Jangir, Nishat Koti, Varsha Bhat Kukkala, Arpita Patra, Bhavish Raj Gopal, and Somya Sangal. Vogue: Faster computation of private heavy hitters. Cryptology ePrint Archive, Report 2022/1561, 2022.
- [29] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall, CRC Press, third edition, 2014.
- [30] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. LERNA: Secure single-server aggregation via key-homomorphic masking. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part I*, volume 14438 of *Lecture Notes in Computer Science*, pages 302–334, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.
- [31] Hanjun Li, Sela Navot, and Stefano Tessaro. POP-STAR: Lightweight threshold reporting with reduced leakage. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.
- [32] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. RoFL: Robustness of secure federated learning. In *2023 IEEE Symposium on Security and Privacy*, pages 453–476, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.
- [33] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *2023 IEEE Symposium on Security and Privacy*, pages 477–496, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.
- [34] Dimitris Mouris, Christopher Patton, Hannah Davis, Pratik Sarkar, and Nektarios Georgios Tsoutsos. Mastic: Private weighted heavy-hitters and attribute-based metrics. *Proceedings on Privacy Enhancing Technologies*, 2025.
- [35] Dimitris Mouris, Pratik Sarkar, and Nektarios Georgios Tsoutsos. PLASMA: Private, lightweight aggregated statistics against malicious adversaries. *Proceedings on Privacy Enhancing Technologies*, 2024(3):4–24, July 2024.
- [36] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT’99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer Berlin Heidelberg, Germany.
- [37] David MW Powers. Applications and explanations of zipf’s law. In *New methods in language processing and computational natural language learning*, 1998.
- [38] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. Heavy hitter estimation over set-valued data with local differential privacy. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 192–203, Vienna, Austria, October 24–28, 2016. ACM Press.
- [39] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. ELSA: Secure aggregation for federated learning with malicious actors. In *2023 IEEE Symposium on Security and Privacy*, pages 1961–1979, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.
- [40] Victor Shoup. Ntl: A library for doing number theory. <https://libntl.org>, 2021. Version 11.5.1.
- [41] Jun Tang, Aleksandra Korolova, Xiaolong Bai, Xueqiang Wang, and Xiaofeng Wang. Privacy loss in apple’s implementation of differential privacy on macos 10.12. *arXiv preprint arXiv:1709.02753*, 2017.
- [42] Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. Federated heavy hitters discovery with differential privacy. In *International Conference on Artificial Intelligence and Statistics*, pages 3837–3847. PMLR, 2020.

A MDSS Proofs

A.1 Unlinkability

We provide here a proof of [Theorem 1](#).

Proof. By the admissibility restrictions of the adversary $\mathcal{A}$, what $\mathcal{A}$ receives when either $b = 0$ or $b = 1$ is the list of indices it chose, along with $\leq t_{\text{priv}}$ points on a collection of $t_{\text{priv}} + 1$ -degree polynomials with their leading coefficient fixed. The points are identical to the uniform distribution on $\mathbb{F}$, and therefore the distribution passed to $\mathcal{A}$ in both games is identical. It follows that the behavior of $\mathcal{A}$ in both games must be identical as well. ■

A.2 Correctness

We provide here a formalization of our heuristic assumption and a proof that our construction satisfies ϵ -unlinkability under it. As in [\[25\]](#), we first formalize the creation of shares. For a field $\mathbb{F}$, we define the algorithm $\text{Eval}^{\mathbb{F}}$ in [Algorithm 3](#).

Algorithm 3: $\text{Eval}^{\mathbb{F}}$

Input: A set of sets of field elements $\{m_1, \dots, m_h\}$ where $\sum_{i=1}^h m_i = B$, a set of polynomial sets $P = \{\{p_{1,1}, \dots, p_{1,c}\}, \dots, \{p_{h,1}, \dots, p_{h,c}\}\}$, and field elements $x_1, \dots, x_B \in \mathbb{F}$.

Output: Evaluations $sh_1, \dots, sh_B \in \mathbb{F}^{c+1}$

$S := \emptyset$

index := 1

for $i \in [h]$ **do**

for $_ \in [m_i]$ **do**

$S := S \cup \{(x_{\text{index}}, p_{i,1}(x_{\text{index}}), \dots, p_{i,c}(x_{\text{index}}))\}$

 index ++

end

end

return S

Heuristic Assumption 1. Let $\mathbb{F}$ be a field. Let λ be a security parameter, and let ℓ, B, h be integers polynomial in λ . Let $m_1, \dots, m_h$ be integers where $B = \sum_{i=1}^h m_i$, let $s_1, \dots, s_h \in \mathbb{F}^c$ be vectors of field elements, and let $x_1, \dots, x_B \in \mathbb{F}$ be unique field elements. Let $\mathcal{P}_{\mathbb{F}, \ell, c, s}$ denote the set of all polynomial sets $P = \{p_1, \dots, p_c\}$ over $\mathbb{F}$ where $\forall i \in [c], \deg(p_i) = \ell$ and the leading coefficient of p_i is 1, and $s = (p_1(0), \dots, p_c(0))$. Finally, let $t_{\text{rec}} = \frac{B+(c \cdot (\ell+1))}{c+1}$. Then:

$$\Pr \left[\text{CH-MDSS}(S) \neq \{s_i \text{ s.t. } m_i \geq t_{\text{rec}}\} \mid \begin{array}{l} \forall i \in [h], P_i \leftarrow \mathcal{P}_{\mathbb{F}, \ell, c, s_i} \\ S := \\ \text{Eval}^{\mathbb{F}}(\{m_1, \dots, m_h\}, \\ \{P_1, \dots, P_h\}, \\ \{x_1, \dots, x_B\}) \\ \leq \text{negl}(\lambda) \end{array} \right]$$

Our heuristic assumption differs from the heuristic in [\[25\]](#) in a few ways. First, it uses a different bound on the number of agreeing points, t_{rec} . Second, the degree of the polynomials is exactly ℓ rather than $\leq \ell$, and the polynomials have their leading coefficient fixed at 1. Third, the probability of failure is negligible in the security parameter rather than the size of the field. The first change comes from an error in [\[25\]](#): the t_{rec} parameter was reported incorrectly as $t_{\text{rec}} = \frac{B+(c \cdot \ell)}{c+1}$ due to an off by one error, and our corrected version represents a strict weakening of the assumption. The second and third changes we believe are minor and justify experimentally. In addition to the experiments described in [Section 6](#), we tested the decoder on 2500 instances and recorded a 100% success rate.

Given this, the proof of [Theorem 2](#) follows directly from the construction in [Figure 2](#) and Heuristic Assumption 1.

B Benaloh Encryption Scheme

The Benaloh Encryption Scheme	
Let GrpGen be a group generator with respect to which the prime residuosity assumption holds (Definition 6) and let dlog be the algorithm from Lemma 7 .	
$\text{Gen}(1^\lambda, v)$	$\text{Enc}(\text{pk}, x)$
1 : $(p, q, g) \leftarrow \text{GrpGen}(1^\lambda, v)$	1 : parse $\text{pk} = (v, g, N)$
2 : $N := pq$	2 : $\gamma \leftarrow \mathbb{Z}_N^*$
3 : $\text{pk} := (v, g, N)$	3 : $\text{ct} := g^x \gamma^v \pmod{N}$
4 : $\text{sk} := (v, g, p, q)$	4 : return ct
5 : return (pk, sk)	
$\text{Dec}(\text{sk}, \text{ct})$	$\text{Eval}(\text{pk}, \text{ct}, v)$
1 : parse $\text{sk} = (v, g, p, q)$	1 : parse $\text{pk} = (v, g, N)$
2 : $N := pq$	2 : parse $\text{ct} = (\text{ct}_1, \dots, \text{ct}_\ell)$
3 : $x := \text{dlog}(p, q, g, \text{ct})$	3 : parse $v = (v_1, \dots, v_\ell)$
4 : return x	4 : $\text{ct} = \prod_{i=1}^{\ell} \text{ct}_i^{v_i} \pmod{N}$
	5 : return ct

Figure 16: The Benaloh encryption scheme [\[11, 12\]](#)

In this section we describe the Benaloh encryption scheme [\[11, 12\]](#), which we use to instantiate the $\Pi_{\text{dope}}^{\mathcal{E}, \ell}$ protocol described in [Section 4.1](#). The security of the Benaloh encryption scheme is based on the prime residuosity assumption. We first introduce some additional notation, then define

the prime residuosity assumption, and conclude with a description of the encryption scheme and its properties.

Notation For a positive integer N , an integer α is said to be an v -th residue modulo N if and only if there exists some integer β such that $\alpha \equiv \beta^v \pmod{N}$. Let $\mathbb{Z}_N^*$ denote the set of integers that are relatively prime to N . Let Res_N^v denote the set of v -th residues modulo N in $\mathbb{Z}_N^*$, and let $\overline{\text{Res}}_N^v = \mathbb{Z}_N^* \setminus \text{Res}_N^v$.

Prime residuosity assumption Let $\text{GrpGen}(1^\lambda, v) \rightarrow (p, q, g)$ be a PPT algorithm such that (1) p and q are prime, (2) v divides $\phi(N)$, (3) v does not divide $\phi(N)/v$, and (4) $g \in \overline{\text{Res}}_N^v$, where $N = pq$ and $\phi(N) = (p-1)(q-1)$ denotes Euler's totient function. We assume there exists a bound $B(\lambda) = 2^{O(\lambda)}$ such that p and q output by $\text{GrpGen}(1^\lambda, v)$ satisfy $p < q < B(\lambda)$. The security of Benaloh's encryption scheme is based on the prime residuosity assumption which informally says that for any prime v , it is computationally hard to distinguish between the uniform distribution over Res_N^v and uniform distribution over $\mathbb{Z}_N^*$, where $N = pq$ is computed as the output of GrpGen .

Definition 6 (Prime Residuosity Assumption). For all non-uniform polynomial time adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all primes v and $\lambda \in \mathbb{N}$, we have

$$\left| \Pr[\mathcal{A}(1^\lambda, N, g, \alpha) = 1] - \Pr[\mathcal{A}(1^\lambda, N, g, \beta) = 1] \right| \leq \text{negl}(\lambda),$$

where $(p, q, g) \leftarrow \text{GrpGen}(1^\lambda, v)$, $N = pq$, $\alpha \leftarrow \mathbb{Z}_N^*$, and $\beta \leftarrow \text{Res}_N^v$.

Next, we recall some useful facts about the structure of the group generated by GrpGen .

Lemma 6 ([11, Theorem 2.22]). For all $\lambda \in \mathbb{N}$, all primes $v \in \mathbb{N}$, every (p, q, g) in the support of $\text{GrpGen}(1^\lambda, v)$ and all $\alpha \in \mathbb{Z}_N^*$, where $N = pq$, there exists a unique $x \in [0, v]$ and unique $\gamma \in \text{Res}_N^v$ such that $\alpha \equiv g^x \gamma \pmod{N}$.

Lemma 7 ([11, Theorem 2.24]). There exists an algorithm dlog such that for all $\lambda \in \mathbb{N}$, all polynomials $\text{poly}(\cdot)$, all primes $v \in \mathbb{N}$, every (p, q, g) in the support of $\text{GrpGen}(1^\lambda, v)$ and all $\alpha \in \mathbb{Z}_N^*$, where $v \leq \text{poly}(\lambda)$ and $N = pq$, $\text{dlog}(p, q, g, \alpha)$ computes $x \in [0, v]$ in polynomial time in λ such that $\alpha \equiv g^x \pmod{N}$ for some $\gamma \in \text{Res}_N^v$.

Encryption scheme We describe the encryption scheme in Figure 16. The following theorem is proved in [11].

Theorem 8. If the prime residuosity assumption holds with respect to GrpGen (Definition 6), then for all prime-valued polynomials $v := v(\lambda)$, the scheme described in Figure 16 is a linearly homomorphic (Definition 4) IND-CPA secure public-key encryption scheme with message space $\mathbb{Z}_v$.

We conclude this section by showing that the encryption scheme has dense ciphertext space.

Theorem 9. For all prime-valued polynomials $v := v(\lambda)$, the scheme described in Figure 16 has dense ciphertext space (Definition 5).

Proof. We first prove the following claim, which states that the uniform distribution on $\mathbb{Z}_N^*$ is identical to the distribution of encryptions of uniformly random messages from $\mathbb{Z}_v$.

Claim. For all prime-valued polynomials $v := v(\lambda)$, all $\lambda \in \mathbb{N}$, and all (p, q, g) in the support of $\text{GrpGen}(1^\lambda, v)$, we have

$$\{(p, q, g, \alpha) : \alpha \leftarrow \mathbb{Z}_N^*\} \stackrel{D}{=} \left\{ (p, q, g, \text{ct}) : \begin{array}{l} x \leftarrow \mathbb{Z}_v \\ \text{ct} \leftarrow \mathcal{E}.\text{Enc}(\text{pk}, x) \end{array} \right\},$$

where $N = pq$, and $\text{pk} = (v, g, N)$.

Proof. Let $\Psi : \mathbb{Z}_v \times \text{Res}_N^v \rightarrow \mathbb{Z}_N^*$ be defined as $\Psi(x, \beta) = g^x \beta \pmod{N}$. Observe that to encrypt a message $x \in \mathbb{Z}_v$, the encryption algorithm samples $\gamma \leftarrow \mathbb{Z}_N^*$ uniformly at random and outputs $\Psi(x, \gamma^v)$. We will show that Ψ is a bijection from $\mathbb{Z}_v \times \text{Res}_N^v$ to $\mathbb{Z}_N^*$. Since γ^v is uniformly random in Res_N^v for uniformly random $\gamma \in \mathbb{Z}_N^*$, the fact that Ψ is a bijection immediately implies that the distribution of encryptions of random messages is equal to the uniform distribution on $\mathbb{Z}_N^*$.

We are left to show that Ψ is indeed a bijection. Observe that from Lemma 6, Ψ is injective. Moreover, since $|\text{Res}_N^v| = \phi(N)/v$, we have $|\mathbb{Z}_v \times \text{Res}_N^v| = \phi(N) = |\mathbb{Z}_N^*|$, which along with the fact that Ψ is injective, implies that Ψ is surjective. Thus, Ψ is a bijection and the claim immediately follows. $\square$

From the above claim and the fact that there exists an efficient algorithm to sample from $\mathbb{Z}_N^*$, defining the ciphertext space as $\mathbb{Z}_N^*$ would seem to imply that the encryption scheme has dense ciphertext space. However, the resulting ciphertext space is defined with respect to the public key, which does not match Definition 5. To conclude the proof, it suffices to define the ciphertext space as $C = \mathbb{Z}_{2^\lambda B^2}$, where $B := B(\lambda)$ is a bound on the primes output by $\text{GrpGen}(1^\lambda)$. Observe that there exists an efficient algorithm to sample from C , and that uniformly random elements from C are statistically close to uniformly random elements from $\mathbb{Z}_N^*$, where $N = pq$ is the product of primes p and q output by $\text{GrpGen}(1^\lambda)$. It then follows from the above claim that the encryption scheme has dense ciphertext space. $\blacksquare$

C Protocol Proofs

C.1 DOPE Protocol

We prove Lemma 3 below.

Proof. Let $\mathcal{A}$ be an adversary that interacts with the parties running $\Pi_{\text{dope}}^{\mathcal{E}, \mathcal{I}_{\text{priv}}}$. We describe a simulator $\mathcal{S}$ below such that

the view of any environment $\mathcal{Z}$ in an interaction with $\mathcal{A}$ and $\Pi_{\text{dope}}^{\mathcal{E}, \ell, \text{priv}}$ is indistinguishable from its view of an interaction with $\mathcal{S}$ in $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$.

Simulator $\mathcal{S}$

$\mathcal{S}$ internally invokes $\mathcal{A}$ and forwards all messages between $\mathcal{Z}$ and $\mathcal{A}$.

1. Upon first activation, it initializes an empty table T and computes $(pk, sk) \leftarrow \mathcal{E}.\text{Gen}(1^\lambda)$. For every invocation of $\overline{\mathcal{F}}_{\text{setup}}$, it returns pk if invoked by a corrupt user and returns (pk, sk) if invoked by a corrupt S_{rand} .
2. Upon receiving (query, sid', x) on behalf of $\overline{\mathcal{F}}_{ro}^C$, it proceeds as follows.
 - (a) If $sid' = sid$ and S_{rand} is corrupt, it checks if $(sid, x) \in T$, and if not, it samples $\beta \leftarrow \mathbb{F}$ uniformly at random, computes $ct \leftarrow \mathcal{E}.\text{Enc}(pk, \beta)$ and sets $T[sid, x] := (\beta, ct)$. It then retrieves $(\beta, ct) := T[sid, x]$ and sends ct to $\mathcal{A}$.
 - (b) If $sid' = sid$ and S_{rand} is honest, it checks if $(sid, x) \in T$, and if not, it samples $\beta \leftarrow \mathbb{F}$ uniformly at random, computes $ct \leftarrow \mathcal{E}.\text{Enc}(pk, 0)$ and sets $T[sid, x] := (\beta, ct)$. It then retrieves $(\beta, ct) := T[sid, x]$ and sends ct to $\mathcal{A}$.
 - (c) If $sid' \neq sid$, it checks if $(sid', x) \in T$, and if not, it samples $ct \leftarrow \mathcal{C}$ uniformly at random and sets $T[sid', x] := ct$. It then sends $T[sid', x]$ to the activating party.
3. Upon receiving (eval, sid, U) from $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$, if S_{rand} is corrupt, sample $\gamma \leftarrow \mathbb{F}$ uniformly at random, compute $ct \leftarrow \mathcal{E}.\text{Enc}(pk, sk)$ and send (ct, γ) to $\mathcal{A}$.
4. When $\mathcal{A}$ invokes a corrupt user U with input (eval, sid, v, x), $\mathcal{S}$ proceeds as follows.
 - (a) For all $i \in [\ell]$, if $(sid, v||i) \notin T$, it samples $\beta \leftarrow \mathbb{F}$ uniformly at random, computes $ct \leftarrow \mathcal{E}.\text{Enc}(pk, \beta)$ and sets $T[sid, v||i] := (\beta, ct)$.
 - (b) For each $i \in [\ell]$, let $(\beta_i, ct_i) := T[sid, v||i]$. It samples $\alpha \leftarrow \mathbb{F}$ uniformly at random and computes

$$\gamma := \alpha + v + \sum_{i=1}^{\ell} \beta_i \cdot x^i, \text{ and}$$

$$ct \leftarrow \mathcal{E}.\text{Enc}(pk, \gamma),$$
 and sends (ct, γ) to $\mathcal{A}$ on behalf of the corrupt user. Moreover, if S_{rand} is corrupt it sends (ct, γ) to $\mathcal{A}$ on behalf of S_{rand} .

Next, we use a hybrid argument to show that the joint dis-

tribution of the output of honest parties and the environment $\mathcal{Z}$ in an execution of $\Pi_{\text{dope}}^{\mathcal{E}, \ell}$ with the adversary $\mathcal{A}$ is indistinguishable from the joint distribution of the output of honest parties and $\mathcal{Z}$ in an execution of $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$ with the simulator $\mathcal{S}$ described above.

- Hyb_0 : This hybrid corresponds to the joint distribution of the output of honest parties and the environment $\mathcal{Z}$ in an execution of $\Pi_{\text{dope}}^{\mathcal{E}, \ell}$ with $\mathcal{A}$.

- Hyb_1 : This hybrid is identical to the previous hybrid, except that $\overline{\mathcal{F}}_{\text{setup}}$ is simulated by $\mathcal{S}$.

Observe that $\mathcal{S}$ computes $(pk, sk) \leftarrow \mathcal{E}.\text{Gen}(1^\lambda)$ and responds to queries as in $\overline{\mathcal{F}}_{\text{setup}}$. It thus follows that $\text{Hyb}_0 \stackrel{d}{=} \text{Hyb}_1$.

- Hyb_2 : This hybrid is identical to the previous hybrid, except that $\overline{\mathcal{F}}_{ro}^C$ is simulated by $\mathcal{S}$ when S_{rand} is corrupt.

The only difference between Hyb_1 and Hyb_2 is that in Hyb_1 , $\overline{\mathcal{F}}_{ro}$ returns a uniformly random element from the ciphertext space $\mathcal{C}$ for each query, while in this hybrid, the $\mathcal{S}$ returns an encryption of a random message when S_{rand} is corrupt. It then follows from the dense ciphertext property of $\mathcal{E}$ that $\text{Hyb}_1 \stackrel{d}{\approx} \text{Hyb}_2$.

- Hyb_3 : This hybrid is identical to the previous hybrid, except that $\overline{\mathcal{F}}_{ro}^C$ is simulated by $\mathcal{S}$.

The only difference between Hyb_2 and Hyb_3 is when S_{rand} is honest. In this case, the output of the query is a uniformly random element from $\mathcal{C}$ in Hyb_2 while in this hybrid it is an encryption of 0. Since $\mathcal{S}$ does not need the secret key sk when S_{rand} is honest, it follows from the IND-CPA security of $\mathcal{E}$ that $\text{Hyb}_2 \stackrel{d}{\approx} \text{Hyb}_3$.

- Hyb_4 : This hybrid is identical to the previous hybrid, except that $\mathcal{S}$ simulates the view of a corrupt parties.

Observe that due to the linear homomorphism of the encryption scheme, in Hyb_3 , the honest user sends an encryption of $\alpha + v + \sum_{i=1}^{\ell} \beta_i \cdot x^i$ to S_{rand} , where $\alpha \leftarrow \mathbb{F}$ is sampled uniformly at random. In this hybrid, $\mathcal{S}$ sends an encryption of a uniformly random field element, which in turn implies that the message sent by honest users to S_{rand} are identically distributed. Next, note that for corrupt users, $\mathcal{S}$ sends an encryption of $\alpha + v + \sum_{i=1}^{\ell} \beta_i \cdot x^i$, where α is sampled using the corrupt user's random tape—which from the linear homomorphism of the encryption scheme is distributed identically to Hyb_3 . It thus follows that $\text{Hyb}_3 \stackrel{d}{=} \text{Hyb}_4$.

- Hyb_5 : This hybrid corresponds to the joint distribution of the output of honest parties and the environment $\mathcal{Z}$ in an execution of $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$ with $\mathcal{S}$.

Algorithm 4: CH-MDSS.Step

Input: Integers t', t_{priv}, n , field element tuples $\{(\alpha_i, \beta_{i,1}, \dots, \beta_{i,c})\}_{i=1}^n$
Output: Polynomials $(p_1 \dots p_c)$ or $\perp$
 $\lambda := t_{\text{priv}} + n - t'$
forall $j \in [c]$ **do**
 $f_j(z) = \text{LagrangeInterpol}(\{(\alpha_1, \beta_{1,j}), \dots, (\alpha_n, \beta_{n,j})\})$
end
 $N(z) = \prod_{i=1}^n (z - \alpha_i)$
construct the matrix $M \in \mathbb{F}[z]^{(c+1) \times (c+1)}$:

$$M = \begin{bmatrix} z^{t_{\text{priv}}} & f_1(z) & f_2(z) & \dots & f_c(z) \\ & N(z) & & & \\ & & N(z) & & \\ & & & \ddots & \\ & & & & N(z) \end{bmatrix}$$

$M_{\text{red}} \leftarrow \text{LatticeReduce}(M)$
if there is one shortest vector $\mathbf{v} = (v_0 \dots v_c)$ that has a length $\leq \lambda$ **and** v_0 divides each of $\{v_1 \cdot z^{t_{\text{priv}}}, \dots, v_c \cdot z^{t_{\text{priv}}}\}$ **then**
 return $(v_1 \cdot z^{t_{\text{priv}}}/v_0, \dots, v_c \cdot z^{t_{\text{priv}}}/v_0)$
end
else
 return $\perp$
end

Observe that from the dense ciphertext property of the encryption scheme, in Hyb₄, the output of honest parties corresponds to the decryption of the ciphertext

$$\text{ct}' = \mathcal{E}.\text{Eval}(\text{pk}, (\text{ct}_0, \dots, \text{ct}_\ell), (1, x, \dots, x^\ell)),$$

where $\text{ct}_0 \leftarrow \mathcal{E}.\text{Enc}(\text{pk}, v)$ and $\text{ct}_i \leftarrow \mathcal{E}.\text{Enc}(\text{pk}, \beta_i)$ for $\beta_i \leftarrow \mathbb{F}$. It then follows immediately from the linear homomorphism property of the encryption scheme that $\text{Hyb}_4 \approx \text{Hyb}_5$.

It follows from our argument above that $\text{Hyb}_0 \approx \text{Hyb}_5$, which in turn implies that $\Pi_{\text{dope}}^{\mathcal{E}, \ell}$ securely realizes $\mathcal{F}_{\text{dope}}^{\mathbb{F}, \ell}$ in the $(\overline{\mathcal{F}}_{\text{setup}}^{\mathcal{E}}, \overline{\mathcal{F}}_{\text{ro}}^{\mathcal{C}})$ -hybrid model. ■

C.2 Report Protocol

We give here a proof of [Theorem 5](#).

Proof. Let $\mathcal{A}$ be an adversary corrupting the Aggregation Server and some number of clients and interacts with the remaining parties running $\Pi_{\text{report}}^{\mathcal{F}_{\text{rec}}, \mathcal{F}_{\text{priv}}, B}$. We describe a simulator

$\mathcal{S}$ below such that the view of any environment $\mathcal{Z}$ in an interaction with $\mathcal{A}$ and $\Pi_{\text{report}}^{\mathcal{F}_{\text{rec}}, \mathcal{F}_{\text{priv}}, B}$ is indistinguishable from its view of an interaction with $\mathcal{S}$ in $\mathcal{F}_{\text{report}}^{\mathcal{F}_{\text{rec}}, \mathcal{F}_{\text{priv}}, B}$.

For an MDSS scheme MDSS, define the subroutine $\text{SimOSS}(T, B, v, \mathbf{t})$:

- 1: If $v \notin T$, compute $\{sh_1, \dots, sh_B\} \leftarrow \text{MDSS.Share}(v, B)$ and set $T[v] := \{sh_1, \dots, sh_B\}$.
- 2: Retrieve $\{sh_1, \dots, sh_B\} \leftarrow T[v]$ and return sh_1 .

Simulator $\mathcal{S}$

$\mathcal{S}$ internally invokes $\mathcal{A}$ and forwards all messages between $\mathcal{Z}$ and $\mathcal{A}$.

1. Upon first activation, it does the following:
 - (a) Initialize an empty table T where by default $T[x] = 0$ for all values x , an empty table T_{oss} , and an empty set S .
 - (b) If S_{rand} is not corrupt, do the following:
 - i. Sample a set of unique indices $I := \mathbf{t}_1, \dots, \mathbf{t}_B$.
 - ii. For each corrupted U_i , send $\mathbf{t}_i$ to $\mathcal{A}$. Let $I_{\mathcal{A}}$ be the set of all indices $\mathbf{t}_i$ sent to $\mathcal{A}$, and compute $\mathbf{t}'_1, \dots, \mathbf{t}'_m := I \setminus I_{\mathcal{A}}$.
2. When $\mathcal{A}$ issues a query (eval, sid, $v, \mathbf{t}$) to $\mathcal{F}_{\text{oss}}^{\text{MDSS}}$, it does the following:
 - (a) Query $\mathcal{F}_{\text{report}}^{\mathcal{F}_{\text{rec}}, \mathcal{F}_{\text{priv}}, B}$ on arguments (msg, sid, v)
 - (b) Set $T[v] := T[v] + 1$.
 - (c) Compute $sh \leftarrow \text{SimOSS}(T_{\text{oss}}, B, v, \mathbf{t})$, then add sh to S and return sh to $\mathcal{A}$.
3. When $\mathcal{A}$ issues a query (collect, sid_{mix}) to $\mathcal{F}_{\text{mix}}^B$, it does the following:
 - (a) Query $\mathcal{F}_{\text{report}}^{\mathcal{F}_{\text{rec}}, \mathcal{F}_{\text{priv}}, B}$ on arguments (collect, sid), receiving $\{(v_1, c_1), \dots, (v_n, c_n)\}$.
 - (b) Initialize the set $V := \{v_1, \dots, v_n\}$.
 - (c) Initialize $k := 0$.
 - (d) For each (v_i, c_i) do the following:
 - i. $\forall j \in [c_i - T[v_i]]$:
 - A. Compute $sh \leftarrow \text{SimOSS}(T_{\text{oss}}, B, v_i, \mathbf{t}'_k)$, then add sh to S .
 - B. Set $k = k + 1$.
 - (e) $\forall j \in [B - \sum_{i=1}^n c_i]$:
 - i. Sample $v_j \leftarrow \text{MDSS}.\mathcal{H} \setminus V$.
 - ii. Add v_j to V .

- iii. Compute $sh \leftarrow \text{SimOSS}(T_{\text{oss}}, B, v_j, \mathbf{t}'_k)$, then add sh to S .
 - iv. Set $k = k + 1$.

(f) Send a random permutation of S to $\mathcal{A}$.

Next, we use a hybrid argument to show that the joint distribution of the output of honest parties and the environment $\mathcal{Z}$ in an execution of $\Pi_{\text{report}}^{t_{\text{rec}}, t_{\text{priv}}, B}$ with the adversary $\mathcal{A}$ is indistinguishable from the joint distribution of the output of honest parties and $\mathcal{Z}$ in an execution of $\mathcal{F}_{\text{report}}^{t_{\text{rec}}, t_{\text{priv}}, B}$ with the simulator $\mathcal{S}$ described above.

- Hyb_0 : This hybrid corresponds to the joint distribution of the output of honest parties and the environment $\mathcal{Z}$ in an execution of $\Pi_{\text{report}}^{t_{\text{rec}}, t_{\text{priv}}, B}$ with $\mathcal{A}$.

- Hyb_1 : This hybrid is identical to the previous hybrid, except that $\mathcal{F}_{\text{oss}}^{\text{MDSS}}$ is simulated by $\mathcal{S}$.

Observe that $\mathcal{S}$ responds to queries as in $\mathcal{F}_{\text{oss}}^{\text{MDSS}}$. It thus follows that $\text{Hyb}_0 \stackrel{c}{\approx} \text{Hyb}_1$.

- Hyb_2 : This hybrid is identical to the previous hybrid, except that $\mathcal{S}$ simulates the actions of S_{rand} when it is not corrupt.

Observe that $\mathcal{S}$ sends to corrupt users indices distributed identically to those in Hyb_1 . It thus follows that $\text{Hyb}_1 \stackrel{c}{\approx} \text{Hyb}_2$.

- Hyb_3 : This hybrid is identical to the previous hybrid, except that $\mathcal{F}_{\text{mix}}^B$ is simulated by $\mathcal{S}$.

The only difference between Hyb_2 and Hyb_3 is the reports computed at step 7.e of $\mathcal{S}$. In each hybrid, these reports consist of MDSS shares of values for which no value has $> t_{\text{priv}}$ shares. Therefore, by the unlinkability property of the MDSS scheme, $\text{Hyb}_2 \stackrel{c}{\approx} \text{Hyb}_3$.

- Hyb_4 : This hybrid corresponds to the joint distribution of the output of honest parties and the environment $\mathcal{Z}$ in an execution of $\mathcal{F}_{\text{report}}^{t_{\text{rec}}, t_{\text{priv}}, B}$ with $\mathcal{S}$.

The output of an honest Aggregation Server in Hyb_3 is $\text{MDSS.Reconstruct}(\text{rpt}_1, \dots, \text{rpt}_B)$. Therefore, by the MD-Correctness of MDSS_c , $\text{Hyb}_3 \stackrel{c}{\approx} \text{Hyb}_4$.

It follows from our argument above that $\text{Hyb}_0 \stackrel{c}{\approx} \text{Hyb}_4$, which in turn implies that $\Pi_{\text{report}}^{t_{\text{rec}}, t_{\text{priv}}, B}$ securely realizes $\mathcal{F}_{\text{report}}^{t_{\text{rec}}, t_{\text{priv}}, B}$ in the $(\mathcal{F}_{\text{oss}}^{\text{MDSS}}, \mathcal{F}_{\text{mix}}^B)$ -hybrid model. ■

D Modified CH-MDSS Algorithm

The full description of the algorithm CH-MDSS.Step described in [Section 5](#) can be seen in Algorithm 5.

Algorithm 5: CH-MDSS.Step

Input: Integers t', t_{priv}, n , field element tuples

$$\{(\alpha_i, \beta_{i,1}, \dots, \beta_{i,c})\}_{i=1}^n$$

Output: Polynomials $(p_1 \dots p_c)$ or $\perp$

$\lambda := t_{\text{priv}} + n - t'$

forall $j \in [c]$ **do**

$f_j(z) = \text{LagrangeInterpol}(\{(\alpha_1, \beta_{1,j}), \dots, (\alpha_n, \beta_{n,j})\})$

end

$$N(z) = \prod_{i=1}^n (z - \alpha_i)$$

construct the matrix $M \in \mathbb{F}[z]^{(c+1) \times (c+1)}$:

$$M = \begin{bmatrix} z_{\text{priv}}^{t'} & f_1(z) & f_2(z) & \dots & f_c(z) \\ & N(z) & & & \\ & & N(z) & & \\ & & & \ddots & \\ & & & & N(z) \end{bmatrix}$$

$M_{\text{red}} \leftarrow \text{LatticeReduce}(M)$

if there is one shortest vector $\mathbf{v} = (v_0 \dots v_c)$ that has a length $\leq \lambda$ **and** v_0 divides each of

$\{v_1 \cdot z_{\text{priv}}^{t'}, \dots, v_c \cdot z_{\text{priv}}^{t'}\}$ **then**

return $(v_1 \cdot z_{\text{priv}}^{t'}/v_0, \dots, v_c \cdot z_{\text{priv}}^{t'}/v_0)$

end

else

return $\perp$

end
