Distributed, Private, Sparse Histograms in the Two-Server Model

James Bell
Google
London, UK
jhbell@google.com

Adrià Gascón
Google
New York, NY, US
adriag@google.com

Badih Ghazi
Google
Mountain View, CA, US
badih.ghazi@gmail.com

Ravi Kumar
Google
Mountain View, CA, US
ravi.k53@gmail.com

Pasin Manurangsi
Google
Bangkok, Thailand
pasin@google.com

Mariana Raykova
Google
New York, NY, US
marianar@google.com

Phillipp Schoppmann
Google
New York, NY, US
schoppmann@google.com

ABSTRACT

We consider the computation of sparse, (ε, δ)-differentially private (DP) histograms in the two-server model of secure multi-party computation (MPC), which has recently gained traction in the context of privacy-preserving measurements of aggregate user data. We introduce protocols that enable two semi-honest non-colluding servers to compute histograms over the data held by multiple users, while only learning a private view of the data. Our solution achieves the same asymptotic ℓ_∞-error of $O\left(\frac{\log(1/\delta)}{\varepsilon}\right)$ as in the central model of DP, but *without* relying on a trusted curator. The server communication and computation costs of our protocol are independent of the number of histogram buckets, and are linear in the number of users, while the client cost is independent of the number of users, ε, and δ. Its linear dependence on the number of users lets our protocol scale well, which we confirm using microbenchmarks: for a billion users, $\varepsilon = 0.5$, and $\delta = 10^{-11}$, the per-user cost of our protocol is only 1.08 ms of server computation and 339 bytes of communication. In contrast, a baseline protocol using garbled circuits only allows up to 10^6 users, where it requires 600 KB communication per user.

ACM Reference Format:
James Bell, Adrià Gascón, Badih Ghazi, Ravi Kumar, Pasin Manurangsi, Mariana Raykova, and Phillipp Schoppmann. 2022. Distributed, Private, Sparse Histograms in the Two-Server Model. In *CCS '22: ACM Conference on Computer and Communications Security, November 14–19, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3548606.3559383

1 INTRODUCTION

Aggregate statistics computed over large user populations are widely used to discover general trends in user behavior and preferences. Applications can be found in many different contexts including product analysis and browser telemetry [10, 15, 26, 27], understanding the spread of viruses [4, 58], and detecting distributed attacks and fraud behavior [13, 63]. Designing techniques for computing such analytics with high accuracy while protecting the privacy of individual users has been an active research topic [6, 10, 11, 13, 15, 23, 26, 27, 33, 40, 43, 63, 67–69, 74, 79].

The notion of differential privacy (DP) [31, 32] formalizes the guarantee that the output of an algorithm does not reveal substantial information about individual user contributions. The techniques for achieving DP inject noise during the computation, which also affects the accuracy of the output. Central DP mechanisms [32] provide the best known trade-off between privacy guarantees and accuracy. However, they rely on the strong assumption of the existence of a trusted curator that has access to the entire dataset. The local DP setting [32, 35, 55] alleviates the privacy implications of the central curator by distributing the privacy mechanism to the clients, which however comes at a high cost in accuracy [8, 21].

Secure multiparty computation (MPC) [45, 46, 60, 77] offers techniques that allow two or more parties to jointly compute a function that depends on their private inputs, while revealing nothing beyond the function output during the computation. A natural idea for achieving strong privacy and high accuracy in a distributed setting is to use MPC to execute central DP mechanisms [31]. However, applying this idea directly to compute aggregate user statistics would require executing a multi-round protocol across the devices of all users whose data is included in the aggregate statistics. Given the high computation and communication overhead of existing large-scale MPC implementations [3, 56, 76], and the unpredictable availability patterns of client devices, this approach becomes challenging with user populations of hundreds of millions or billions.

An intermediate trust model, which avoids a central aggregator and the scalability challenges of fully distributed MPC, is the *outsourced MPC model*. Here, the functionality of the aggregator is split across a small number of non-colluding parties. These receive secret-shared (or encrypted) inputs from the clients, and then compute the desired aggregate statistics using an MPC protocol between them. As long as at least one of the parties remains honest, the clients' inputs remain private and only the desired aggregate is

CCS '22, November 14–19, 2022, Los Angeles, CA, USA

https://doi.org/10.1145/3548606.3559383

revealed. Apart from a lower communication and computation overhead, the outsourced MPC model can handle client drop-outs, since usually only a single message from each client is required. The particular case of two computing servers is called the *two-server model*, which has been applied in many large MPC deployments [4, 27]. While honest-majority protocols with a larger number of parties can result in better efficiency, it remains challenging to ensure that the honest-majority assumption indeed holds. On the other hand, dishonest-majority MPC protocols for more than two parties suffer from performance drawbacks compared to their two-party counterparts (see, e.g., [18, Table 6] for a comparison). Therefore, in this work, we focus on this setting with two non-colluding servers and a large number of clients that each only send a single message.

Sparse histograms. Many popular aggregation functions can be described by histograms over user data. Here, each user has a single value from a domain D, and the goal is to compute the number of users holding each possible input value. In many settings, the domain D of the user contributions is much larger than the actual number of unique values among the inputs, and in some settings it is also larger than the total number of users. Hence, the resulting histograms will often be *sparse*, i.e., most values in the domain will have a count of zero. Examples include the computation of heavy hitters among strings held by the users [16, 57], finding commuter patterns in location data [25], or spatial decompositions [25].

In the case of sparse histograms the question of computational efficiency becomes even more pronounced—ideally, protocols should achieve computation and communication complexities that are *independent* of the domain size $|D|$ and only depend on the number of contributions that need to be processed. The first question to answer in the search for such a protocol is if there is a central DP mechanism that has output length and computation cost that are independent of $|D|$. While mechanisms that add DP noise to every possible entry in the histogram do not satisfy this property, the work of Korolova et al. [57] provides such a solution by guaranteeing that zero counts are always (implicitly) reported as zeros and only a subset of the non-empty histogram locations are reported.

Leveraging existing MPC techniques to realize the central DP mechanism of [57] comes with a set of challenges. Clearly, techniques that require the clients to send inputs proportional to $|D|$ [26] are undesirable. Distributed point functions [17] compress the client computation and communication to $O(\log |D|)$, and can be used as frequency oracles to discover non-zero locations in the sparse histogram [16]. This approach, however, will incur an error due to DP that is also $O(\log |D|)$, which is worse than that of [57]. To the best of our knowledge, there is no efficient DP protocol for computing sparse histograms that achieves an error independent of $|D|$, without relying on a trusted curator.

Our contributions. In this work, we present distributed protocols in the two-server model for computing sparse histograms. Our protocols require one-shot communication of $O(\log(|D|))$ bits from the clients, and the communication between the two servers is linear in the number of contributions from the clients. It provides (ϵ, δ)-DP for the output with ℓ_∞-error of $O\left(\frac{\log(1/\delta)}{\epsilon}\right)$, which matches the best possible bound in the central DP model.

Our protocols guarantee that the output is DP; furthermore, they also guarantee that the view of each server satisfies a computational version of DP called SIM$^+$-CDP [65]. Unlike previous work on distributed DP protocols, however, we explicitly specify the DP leakage that is revealed during the protocol execution. This enables comparisons of different approaches beyond the guarantees of DP, and in particular allows distinguishing pure MPC solutions from protocols revealing additional information.

Our result is summarized in the following informal theorem.

Theorem 1. *Consider n clients each holding a pair $\mathsf{ind}_i \in D, \mathsf{val}_i \in [\Delta]$. There is a one-round protocol relying on two non-colluding servers $\mathsf{P1}, \mathsf{P2}$ for $\mathsf{P1}$ to obtain a histogram of the input data with l_∞-error $O(\Delta \log(1/\delta)/\varepsilon)$. The combination of the output histogram of the protocol and its leakage (see Definition 3 for a formal definition) is (ϵ, δ)-DP. For constant ϵ and δ inverse polynomial in n, the communication and computation are $O(\log |D| \cdot n)$ for the servers, and $O(\log |D|)$ for the clients.*

At the core of our solution is a reduction from the problem of computing DP histograms in a distributed manner over large (exponential-sized) domains to the problem of computing anonymous histograms over small domains proportional to the number of non-zeros in the output histogram. To achieve this, we leverage cryptographic techniques for distributed evaluation of oblivious pseudorandom functions (OPRFs) [54, 64], which enable the two computing parties to transform the indices from the histogram domain to a pseudorandom domain that allows aggregation while hiding the actual values.

We also develop new distributed DP protocols for computing anonymous histograms, where the servers do not have access to the indices of the inputs in the clear. Our first technique relies on duplication and rerandomization of ciphertexts, and our second alternative technique builds on a secure two-server implementation of a heavy-hitters like the one of Boneh et al. [16].

Beyond asymptotic analysis of our protocols, we present an experimental evaluation of the communication and computation costs, and compare our protocols to a baseline that uses garbled circuits [77]. Our results show that our protocols scale well with increasing numbers of parties, due to their linear complexity in the number of inputs. For a billion users and domain elements that fit in a single ciphertext, we can compute a DP histogram using just 1.08 ms of total server computation, and 339 bytes of communication between the servers per user. At the same time, each user only needs to perform 0.46ms of computation and communicate 192 bytes in a single message.

Related work. Böhler and Kerschbaum [14] present a two-party protocol for computing approximate heavy hitters with DP. Like our baseline (Section 3.1), their protocol uses generic MPC. In fact, their Algorithm 1 with $t = n$ is functionally equivalent to Figure 14, with smaller values of t trading off accuracy against performance. In contrast, our main protocol outperforms the garbled circuit baseline by orders of magnitude, by allowing the two servers to learn additional, private information about the inputs.

The *clones* technique we use in Section 4.2 is similar to the *fake users* technique used in [24]. A crucial difference is that even for real users, they have to randomize their inputs using a RAPPOR-like procedure (cf. [34]), i.e., flipping the bit in each bucket (see [24, Algorithm 2]). This cannot be easily done for anonymous histograms because the "buckets" here are the multiplicities that cannot be

determined from each (encrypted) input. Therefore, their protocol cannot be applied to our setting.

2 BACKGROUND & MODEL

2.1 Privacy

We use $\text{supp}(\mathcal{U})$ to denote the support of a distribution $\mathcal{U}$. We also write $p_{\mathcal{U}}(x)$ to denote the probability mass of $\mathcal{U}$ at x. For $k \in \mathbb{N}$, we write $\mathcal{U}^{\star k}$ to denote the distribution of the sum of k independent samples from $\mathcal{U}$, i.e., the k-wise convolution of $\mathcal{U}$. For convenience, we write $a + \mathcal{U}$ for some $a \in \mathbb{R}$ to denote the distribution of $a + X$ where $X \sim \mathcal{U}$. We also sometimes write a random variable in place of its distribution and vice versa.

The *ε-hockey stick divergence* between distributions $\mathcal{U}, \mathcal{U}'$ is

$$d_\varepsilon(\mathcal{U} \| \mathcal{U}') := \sum_{x \in \text{supp}(\mathcal{U})} [p_{\mathcal{U}}(x) - e^\varepsilon \cdot p_{\mathcal{U}'}(x)]_+,$$

where $[y]_+ := \max\{y, 0\}$.

We say that two distributions $\mathcal{U}, \mathcal{U}'$ are (ε, δ)-*indistinguishable*, denoted by $\mathcal{U} \equiv_{\varepsilon,\delta} \mathcal{U}'$, iff $d_\varepsilon(\mathcal{U} \| \mathcal{U}'), d_\varepsilon(\mathcal{U}' \| \mathcal{U}) \leq \delta$. We consider two datasets X, X' to be *neighboring* if X' results from changing a single user's contribution in X.

Differential privacy. A function f is said to be (ε, δ)-*differentially private* (or (ε, δ)-*DP*) [31] if, for every pair of neighboring datasets X, X' it holds that, $f(X) \equiv_{\varepsilon,\delta} f(X')$.

The above neighboring notion is referred to in the literature as *substitution* DP. We will as part of the proof make use of the notion of *add/remove* DP. This is defined by saying X' neighbors X if one is reached from the other by removing a single user. We will use the fact that add/remove DP implies substitution DP. [1] However we do not provide an add/remove DP guarantee for the whole protocol as the view of a server in our protocol includes the number of users.

We use the following probability distribution families. The *Poisson* distribution, denoted $\text{Poi}(\eta)$, is the discrete non-negative distribution with mass function $\exp(-\eta)\eta^x/x!$. The *negative binomial* distribution, denoted $\text{NBin}(r, p)$, is the discrete non-negative distribution with mass function given by $\binom{x+r-1}{x}(1-p)^r p^x$. The *discrete Laplace* distribution, denoted $\text{DLap}(\lambda)$, is the discrete distribution with mass function $\propto \exp(-|x|/\lambda)$. We will use the *(discrete) Laplace Mechanism*, i.e., the fact that adding a noise sample from $\text{DLap}(\lambda)$, with $\lambda = \Delta/\epsilon$, to the result of a sensitivity-Δ (discrete) query provides $(\epsilon, 0)$-DP. The *truncated discrete Laplace* distribution, denoted $\text{TDLap}(\lambda, t)$, is the discrete distribution on $\{-t, ..., t\}$ with mass function $\propto \exp(-|x|/\lambda)$. We will use the fact that adding a noise sample from $\text{TDLap}(\lambda, t)$, with $\lambda = \Delta/\epsilon$, to the result of a sensitivity Δ query provides $(\epsilon, 2e^{-(t-\Delta)\epsilon/\Delta})$-DP. This follows from the following tail bound, which we use throughout the paper: for $X \sim \text{DLap}(\lambda)$, it holds that $Pr[|X| \geq s\lambda] \leq 2e^{-s}$. Thus, setting $t = \lceil \Delta + \Delta/\epsilon \log(2/\delta) \rceil$ provides (ϵ, δ)-DP. We use this mechanism in situations where we require bounded noise samples. The *truncated shifted discrete Laplace* distribution, denoted $\text{TSDLap}(\lambda, t)$, is the discrete distribution on $\{0, ..., 2t\}$ with mass function $\propto \exp(-|x - t|/\lambda)$. An analogous result holds in this case: adding a noise sample from $\text{TSDLap}(\lambda, t = \lceil \Delta + \Delta/\epsilon \log(2/\delta) \rceil)$ to the result of a sensitivity Δ query provides (ϵ, δ)-DP. We use this mechanism in situations where we require positive noise samples.

[1] If f is (ε, δ)-add/remove DP, then f is $(2\varepsilon, (1 + \exp(\varepsilon))\delta)$-substitution DP.

2.2 Security

Homomorphic encryption. Homomorphic encryption (HE) is a primitive that allows computation on encrypted data. In our construction we only use additive HE schemes with function secrecy, denoted by AHE. Our main construction relies on ElGamal encryption in its additively-homomorphic variant (Figure 13 in the appendix.)

Garbled circuits. Garbled circuits [77] are a generic approach for secure two-party computation that enables the secure evaluation of any function that can be represented by a Boolean circuit. This a one-round protocol where one of the parties, the garbler, prepares an encoding of the evaluated circuit referred to as a *garbled circuit (GC)* and sends it to the other party, the evaluator, which can only evaluate the GC on a set of inputs for which it has the corresponding garbled encodings. The garbler provides the encodings of its own input and the parties run a protocol to enable the evaluator to obtain the encodings for its input.

Oblivious pseudorandom function (OPRF). A pseudorandom function (PRF) [44] is a keyed function F_K such that the output $\mathsf{F}_\mathsf{K}(x)$ is indistinguishable from random even when the input x is known, as long as the key K is secret. An oblivious PRF [54, 64] is a PRF that has a mechanism for evaluating it such that the party holding the key K does not learn the input x, and the party providing the input x learns $\mathsf{F}_\mathsf{K}(x)$.

In our protocols we use the PRF $\mathsf{F}_\mathsf{K}(x) = \mathsf{H}(x)^\mathsf{K}$ introduced by Jarecki et. al [54], who showed that this function is pseudorandom when H is modeled as a random oracle.

2.3 Setting & Threat Model

The goal of our paper is to compute a DP histogram over inputs held by many clients, without trusting any single party. We achieve this by distributing trust across two servers, and having them compute the histogram using an interactive secure computation protocol. The servers are assumed to be *semi-honest*, i.e., they follow the steps of the protocol and in addition, are *non-colluding* and do not share or receive any information with each other.

We require the outputs of our protocols to guarantee (ε, δ)-DP. However, since the original definition of DP assumes a central, trusted curator, it does not immediately generalize to multiple parties. Beimel et al. [8] extend the notion of DP to the multi-party setting, by requiring that the views of each subset of parties corrupted by an adversary be DP; their work focuses on the information-theoretic setting without computational assumptions. Mironov et al. [65] introduce *computational DP (CDP)*, which allows for a computationally bounded adversary, and which has been used in recent works [41, 73]. Their strongest privacy notion, SIM$^+$-CDP, requires that the protocol in question securely implements (in the ideal/real simulation paradigm of MPC [45, 59]) a functionality that in turn provides DP. What this means is that the distributed execution of the MPC protocol does not reveal to any of the parties anything more than the output of the computation, which also provides DP properties. As they show, this is a stronger guarantee than only requiring that the view of each party during the execution is DP.

In the MPC literature, multiple works [50, 62, 70] explore the notion of *DP leakage*. This relaxes the regular MPC guarantee where no party can learn anything other than the output, by allowing the participants to learn additional information, but imposing the requirement that this additional information provided is DP. Formally, this is modeled by capturing the additional information revealed during the protocol execution as a leakage term, which is provided to the simulator used in the security proof. This allows comparing different protocols for the same functionality in terms of their leakage, which can vastly differ. In particular, it allows a more fine-grained control over the information leaked, beyond DP.

We follow the same paradigm for our security definition and require protocols to explicitly define their leakage $\mathcal{L}$ that gets revealed in the ideal-world functionality together with the output. A protocol implementing functionality $\mathcal{F}$ is *secure with leakage* $\mathcal{L}$, if it computes $\mathcal{F}$ and the view can be simulated from $(\mathcal{F}, \mathcal{L})$. We require that $\mathcal{F}$ and $\mathcal{L}$ be jointly defined so as to define their joint distribution in a function $\hat{\mathcal{F}}$.

Definition 2 (View). Let Π be a two-party protocol with inputs from $X_1 \times X_2$. Then $\mathsf{View}_b^{\Pi}(x_1, x_2)$ denotes the view of party b during the execution of Π with inputs $x_1 \in X_1$ from P1 and $x_2 \in X_2$ from P2. The view includes all messages received, as well as all random numbers sampled during the execution (see Goldreich [45, Section 7.2]).

Definition 3 (Functionality with leakage). Let $\hat{\mathcal{F}} = (\hat{\mathcal{F}}_1, \hat{\mathcal{F}}_2) = \big((\mathcal{F}_1, \mathcal{L}_1), (\mathcal{F}_2, \mathcal{L}_2)\big)$ be a two-party functionality from $X_1 \times X_2$. Let $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ and $\mathcal{L} = (\mathcal{L}_1, \mathcal{L}_2)$. We say that a two-party protocol Π *securely implements* $\mathcal{F}$ *with leakage* $\mathcal{L}$, if for each $b \in \{1, 2\}$ there exists a probabilistic polynomial-time algorithm Sim_b such that for all $x_1 \in X_1, x_2 \in X_2$, the output of $\big(\mathsf{Sim}_b(x_b, \hat{\mathcal{F}}_b(x_1, x_2)), \mathcal{F}(x_1, x_2)\big)$ is computationally indistinguishable from $\big(\mathsf{View}_b^{\Pi}(x_1, x_2), \Pi(x_1, x_2)\big)$. We call $\hat{\mathcal{F}}$ the *functionality with leakage*.

Note that this definition does not require the leakage to be explicitly computed by Π, which would be required if we asked for a secure computation of $\hat{\mathcal{F}}$. This also means however that we will not rule out the possibility that learning $\mathcal{L}_1$ and $\mathcal{L}_2$ together might leak too much about the output. For this reason we require that the party not colluding with the adversary does nothing to reveal their leakage to the other party. This includes through any further actions taken. We do however allow, as in classical MPC, each party to share their output with the other party, or use it in subsequent computations.

Malicious clients. While the focus of this work is constructing a distributed aggregation protocol that protects the privacy of the contributing clients, another concern for practical deployments might be malicious clients who provide incorrect inputs that skew the output and render it useless, or collude with one of the two servers to reveal the values of honest clients. The main approach that has been adopted in such a setting is aiming to limit the clients' contributions to some allowable range by adding zero-knowledge proofs [47] from the clients that allow the aggregators to verify the clients' inputs are valid without learning any further information. Techniques such as Bulletproofs [20] enable the client to generate a proof for the range of its input, which can be verified by any other party. The approach of the Prio work [26] enables range proofs that leverage two non-colluding verifiers to achieve better efficiency. Introducing clients' range proofs and integrating with our constructions in this paper to protect against malicious clients is an interesting topic for future work.

3 TARGET FUNCTIONALITY & BASELINES

In this paper we aim to implement a distributed version of the mechanism of Korolova et al. [57], also introduced by Bun, Nissim, and Stemmer [19] in a different context, and sometimes referred as a *stability-based histogram*. Given a dataset $\mathcal{I} = (\mathsf{ind}_i)_{i \in [n]}$ of indices from a large domain D, the mechanism (i) builds a histogram $\mathcal{H}$ of $\mathcal{I}$, (ii) adds $\mathsf{DLap}(2/\varepsilon)$ noise to each of the non-zero entries of $\mathcal{H}$, (iii) removes the entries whose value is below a threshold $\tau = 2\log(2/\delta)/\varepsilon$, and (iv) releases the resulting histogram. The threshold is chosen so that the probability of releasing an index with true count 1 is bounded by δ. The variant where each client might contribute a larger value $\mathsf{val}_i \in [1, \ldots, \Delta]$ to ind_i, and thus the input is a set $\mathcal{I} = (\mathsf{ind}_i, \mathsf{val}_i)_{i \in [n]}$ can be easily handled by adding $\mathsf{DLap}(2\Delta/\varepsilon)$ noise and setting $\tau = \Delta + 2\Delta\log(2/\delta)/\varepsilon$.

3.1 Generic MPC Solution

One direct solution is to apply generic two-party computation (2PC) between the two servers for the central DP mechanism described above. Clients secret-share their input across the two servers, and then the servers engage in a generic 2PC, e.g., using garbled circuits, to implement our target functionality described above. Recall that the garbled circuits protocol requires us to express the computed function as a Boolean circuit. Therefore, using a naive encoding with too many input-dependent operations blows up the circuit size, and thus the computational costs (which are linear in the number of AND gates in the circuit). Figure 14 in the appendix presents a data-oblivious algorithm for our target functionality that results in a circuit of size $O(\log|D| \cdot n \log n)$ by relying on well-known sorting/permutation networks of size $O(n \log n)$ [1][2]. The solution is inspired by the sort-compare-shuffle approach to garbled circuits based private set intersection [53]. The properties of the resulting protocol, which we use as a baseline in our experimental evaluation, are captured in the following theorem. As mentioned above, a distinctive aspect of our solution is that the server costs are $O(n \cdot \log(|D|))$.

Theorem 4. *Consider n clients each holding a pair $\mathsf{ind}_i \in D, \mathsf{val}_i \in [\Delta]$. There is a one-round secure protocol relying on two non-colluding servers* P1, P2 *for* P1 *to obtain a DP histogram of the input data with l_∞-error $O(\Delta\log(1/\delta)/\varepsilon)$. The communication and computation are $O(\log|D| \cdot n \log n)$ for the servers, and $O(\log|D|)$ for the clients.*

Note that the communication and computation cost in Theorem 4 is larger by a $\log n$ factor than the one in our Theorem 1.

3.2 Shuffle DP

Another possible baseline is to use the protocols from the shuffle DP literature. Recall that the *shuffle model* of DP [11, 23, 33] is an intermediate model between the local and central models of DP, where the client sends messages to a trusted *shuffler* that randomly

[2] In practice, sorting networks of size $O(n \log^2 n)$ are used due to their better concrete efficiency [7].

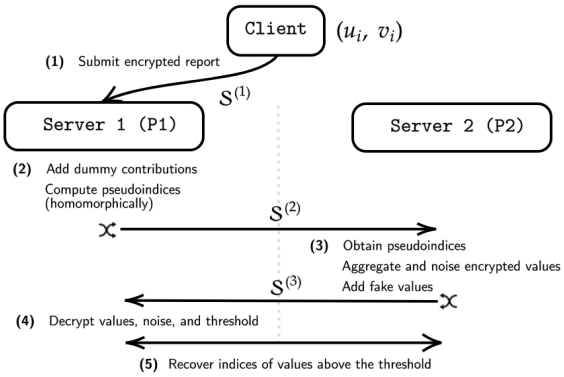

Figure 1: High-level flow of our main protocol.

permutes the messages of all users together before sending them to the analyzer. The requirement is that the view of the analyzer (or equivalently the multiset of messages) needs to be DP. It is possible to instantiate the Shuffle DP model in a two-server setting by implementing secure shuffling (e.g., via onion shuffling).

Histogram queries are well-studied in the shuffle model [5, 23, 24, 36, 39, 40, 42, 43]. Unfortunately, while it is known that an error of $O_\varepsilon(\log(1/\delta))$ is achievable [5, 42], known protocols suffer from communication complexity that grows with $\Omega_\varepsilon\left(\frac{|D|}{n}\sqrt{\log(1/\delta)}\right)$ where $|D|$ denotes the domain size. This is prohibitively large in our setting of interest where $|D| \gg n$; therefore, we cannot use this as a baseline in our experiments.

4 TECHNICAL OVERVIEW

Recall that the input to our problem is a set $\{(\mathsf{ind}_i, \mathsf{val}_i)\}_{i\in[n]}$ of client-held (index, value) pairs.

As mentioned above, our protocol leaks a DP view of the input data to each of the non-colluding servers. Intuitively, our protocol reveals, besides the output, a DP anonymized histogram of $\{\mathsf{ind}_i\}_{i\in[n]}$ to one of the servers, and a DP anonymized histogram of $\{\mathsf{val}_i\}_{i\in[n]}$ to the other one. Recall that an anonymized histogram corresponds to the number of values occurring i times, for every $i > 0$. The privacy of individuals' input (as well as that of small groups) are thus protected in the precise sense of DP, while protecting the input as a whole (in the sense of standard simulation-based MPC) is sacrificed in favor of efficiency, as discussed above.

4.1 Main Protocol Description

Outline of the protocol steps. A high-level description of our main protocol is shown in Figure 1, and involves four steps of interaction between the servers P1 and P2, as outlined below.

Step (1): Clients submit an encrypted report to P1, constituting a set $\mathcal{S}^{(1)}$ of ciphertexts, encrypted under keys held by P2 – the values are encrypted directly while the indices are hashed and then encrypted. Due to the properties of the (ElGamal) encryption, P1 can manipulate the encrypted reports in $\mathcal{S}^{(1)}$ to *homomorphically*, i.e., without prior decryption, (i) randomize the hashed index $\mathsf{H}(\mathsf{ind}_i)$ into a pseudoindex $\mathsf{H}(\mathsf{ind}_i)^K$, generated under a key K held by P1, and (ii) duplicate and rerandomize encryptions.

Step (2): Using these two operations as well as simulating additional dummy contributions, P1 constructs a set $\mathcal{S}^{(2)}$ of encryptions of (pseudoindex, value) pairs that contains the original set of client contributions. The second component is encrypted under an AHE scheme for which P1 has the key and then additionally with a layer of semantically secure encryption for which P2 has the key, which protects the values from P1. P1 sends $\mathcal{S}^{(2)}$ to P2 in a random order. Dummy contributions in $\mathcal{S}^{(2)}$ are given a value of 0 so that they do not affect the final histogram estimate.

Step (3): P2 decrypts the ciphertexts in $\mathcal{S}^{(2)}$, and groups them by their first component. Note that this only reveals the multiplicity of each index, as the indices are pseudorandom (they are encoded as $\mathsf{H}(\mathsf{ind}_i)^K$) and the values are encrypted. P2 then adds up values homomorphically, and returns the resulting set of values to P1, along with a random number of dummy encryptions of values in $[\Delta]$ (plus Laplace noise) in a random order; let $\mathcal{S}^{(3)}$ be this set. The purpose of the dummy values is to ensure that P1 can decrypt the values homomorphically aggregated by P1 and threshold them (with threshold τ) in the clear, while preserving DP.

Step (4): Pseudoindices are inverted and P1 learns the histogram.

The above description is a slight simplification as we cannot "invert" the pseudoindices. Instead, each client also sends an encryption of its index encrypted under the keys of both P1, P2 (denoted by b_i in Figure 5); these encrypted indices are passed around together with the aforementioned pseudoindices and values, and only fully decrypted for the indices that pass the threshold.

Dummy contributions and DP. Note that there are two steps where dummy contributions are injected: step (2) and step (3). In both cases the distributions for the dummy contributions are carefully chosen to ensure that the amount the other party can learn about the input, observing the traffic in the respective steps, is bounded in the sense of DP. This results in a trade-off between computation/communication costs and privacy.

Concretely, in step (3) P2 learns an *anonymized histogram* (aka *histogram of a histogram*) of the set $\{\mathsf{ind}_i\}_{i\in[N]}$ of indices in the input, which is defined as the histogram $\mathcal{H}$ whose ith entry $\mathcal{H}_i$ contains the number of indices with multiplicity i in the input. But, this is leaky since $\mathcal{S}^{(2)}$ reveals the multiplicity of each of the indices in the input. Unfortunately, this makes our protocol not DP (e.g., if the adversary knows all-but-one of the indices, then it can infer from $\mathcal{H}$ with certainty if the remaining index coincides with its known indices.)

As mentioned above, we overcome this issue by having P1 insert dummy contributions in $\mathcal{S}^{(2)}$, in addition to the ones corresponding to the input. As we will explain below, a careful selection of the distribution of the dummy contributions ensures that $\mathcal{S}^{(2)}$ now only leaks a DP anonymized histogram. Note that the situation in step (3) is analogous, as in that case P2 inserts dummy contributions to ensure DP of P1's view of the protocol. A core challenge of this approach is in balancing the trade-off between privacy and communication: dummy contributions help provide meaningful DP protection but can blow-up communication. A main component of our solution is a mechanism for doing this efficiently, which we overview next. We build up to our solution by starting with a simpler, less efficient approach and progressing to more sophisticated, efficient ones.

4.2 Anonymous Histograms via Duplication

In this section we present two different protocols to achieve DP under complementary assumptions on the input distribution. Our *hybrid* protocol will correspond to running these two protocols sequentially. (In Appendix E, we present and evaluate an entirely different protocol that is based on private heavy-hitters; this has communication advantage both asymptotically and numerically only in settings with a small number of heavy hitters.)

More specifically, for a threshold value T, the first protocol (per-multiplicity noising) provides DP *only* to user contributions whose multiplicity is at most T, while the second protocol, duplication-based noising, protects inputs with multiplicity at least T.

Per-multiplicity noising: An efficient protocol for small multiplicities. A standard approach to producing a DP histogram is to add appropriately scaled (discrete) Laplace noise to each of its entries. To implement this idea in our setting, P1 would have to add $O_{\varepsilon,\delta}(D)$ dummy contributions ($O_{\varepsilon,\delta}(1)$ many for each possible index). A slight optimization follows from the fact that, since P2 observes an anonymized histogram, it is enough to noise a histogram of multiplicities $\mathcal{H}$, where $\mathcal{H}_i$ counts the number of *pseudoindices* with multiplicity i in $\mathcal{S}^{(2)}$. In our setting, P1 can implement this mechanism by adding contributions with dummy indices (from a domain $\mathcal{I}$ disjoint with the original domain); adding i contributions with a single dummy index is equivalent to adding a noise of value one to the i-anonymized histogram entry $\mathcal{H}_i$. Since $\mathcal{H}$ can have as many as n non-zero entries and the noise has to be added to each entry, P1 needs to add $\sum_{i\in[n]} O_{\varepsilon,\delta}(i) = O_{\varepsilon,\delta}(n^2)$ different such contributions to ensure that $\mathcal{S}^{(2)}$ is DP. However, if we could assume that no pseudoindex has multiplicity above a threshold T, i.e., that $\forall i > T : \mathcal{H}_i = 0$, then noising up to multiplicity T suffices, and the overhead is $O_{\varepsilon,\delta}(T^2)$; this is clearly undesirable for large T.

Duplication: An efficient protocol for large multiplicities. Note that P1 is not limited to simulating dummy contributions: since ElGamal encryption allows for rerandomization, P1 can obliviously produce an encryption of $(\mathsf{ind}_i, 0)$ given an encryption of $(\mathsf{ind}_i, \mathsf{val}_i)$, learning neither ind_i nor val_i. We will leverage this "duplication" capability to construct a protocol.

The following observation is crucial. Consider an input dataset $\mathcal{D}$ and another dataset $\mathcal{D}'$ identical to $\mathcal{D}$ except without client 1's data, and the corresponding anonymized histograms $\mathcal{H}, \mathcal{H}'$ for the respective set $\mathcal{I} = \{\mathsf{ind}_i\}_{i\in[n]}$ and $\mathcal{I}' = \{\mathsf{ind}_i\}_{i\in[2\ldots n]}$ of indices. Note that these datasets are neighboring in the add/remove sense (Section 2.1). Now, let x be the multiplicity of ind_1 in $\mathcal{I}$, and note that $\mathcal{H}, \mathcal{H}'$ *differ only in two adjacent entries* $x, x-1$, as removing ind_1 from $\mathcal{I}$ reduces the number of indices with multiplicity x by one, while increasing the number of indices with multiplicity $x-1$ by one. More precisely, it holds that $\mathcal{H}_x = \mathcal{H}'_x + 1, \mathcal{H}_{x-1} = \mathcal{H}'_{x-1} - 1$, and $\forall y \notin \{x, x-1\}, \mathcal{H}_y = \mathcal{H}'_y$. (Note that this is less general than a histogram where changes with respect to a neighboring dataset can happen in arbitrary buckets, although the ℓ_1 sensitivity is bounded by 2 in both cases.)

Consider how $\mathcal{H}$ changes when we duplicate each index in $\mathcal{I}$ a random number of times sampled from a distribution $\mathcal{U}$. This is described in the following algorithm $\mathsf{Dup}(\mathcal{H})$, which returns the modified histogram $\mathcal{H}^d$ given $\mathcal{H}$:

$\mathsf{Dup}(\mathcal{H})$:
  $\mathcal{H}^d = \emptyset$ ▷ Empty histogram
  For $y \in \mathsf{Dom}(\mathcal{H})$
    Repeat $\mathcal{H}_y$ **times**
      Sample $a \sim \mathcal{U}^{\star y}$
      $\mathcal{H}^d_{y+a} \leftarrow \mathcal{H}^d_{y+a} + 1$
  Return $\mathcal{H}^d$

Note that the algorithm iterates over each entry y of the original histogram $\mathcal{H}$ "shifting" each of the contributions to entry y by $a \sim \mathcal{U}^{\star y}$ entries. Thus, a corresponds to the total number of additional copies of an index with multiplicity y, where each of its y instances is duplicated $X \sim \mathcal{U}$ times.

To satisfy DP, $\mathcal{D}$ should be chosen so that $\mathsf{Dup}(\mathcal{H}) \equiv_{\varepsilon,\delta} \mathsf{Dup}(\mathcal{H}')$. Since $\mathcal{H}$ and $\mathcal{H}$ differ by one in entries $x-1$ and x, and are equal elsewhere, this boils down to the property that $a_{x-1} \equiv_{\varepsilon,\delta} 1 + a_x$, where $a_{x-1} \sim \mathcal{U}^{\star(x-1)}, a_x \sim \mathcal{U}^{\star x}$. This is almost the same as the condition for the mechanism that adds $\mathcal{U}^{\star x}$ noise to be DP (which just replaces $a_{x-1} \sim \mathcal{U}^{\star(x-1)}$ with $a_{x-1} \sim \mathcal{U}^{\star x}$). Indeed, we show that several well-known distributions $\mathcal{U}$ such as Negative Binomial—which have already been used for DP—satisfy our more stringent condition, assuming that $x > T$. Note that the latter assumption is necessary: if $x = 1$, then the condition obviously fails as $a_{x-1} = 0$ always whereas $1 + a_x \geq 1$. To achieve DP, we are required to have $a_{x-1} > 0$ with at least $O_{\varepsilon,\delta}(1)$ probability. Hence, the expected number of duplicates required per item is $O_{\varepsilon,\delta}(1/T)$, yielding a total of $O_{\varepsilon,\delta}(n/T)$ duplicates.

A hybrid protocol: Best of both worlds. We sequentially combine the duplication and per-multiplicity noising protocols to obtain a hybrid protocol. To do that, we first add per-multiplicity noise up to a predefined threshold T, and then apply the duplication protocol to the resulting set. This leaves us with the task of choosing T. Since our goal is to minimize the total number of dummy contributions inserted by P1, this boils down to optimizing $O_{\varepsilon,\delta}(T^2 + n/T)$ (which in practice we perform numerically), corresponding to the overhead of this hybrid protocol.

As we demonstrate in our experiments later, the hybrid protocol—instantiated with $\mathsf{TSDLap}(\cdot)$ and $\mathsf{NBin}(\cdot)$ distributions, respectively—results in a practical protocol, but we can do better. Next, we introduce an optimization that leads to our final protocol.

An improved protocol. Notice that once the threshold T is fixed, the hybrid protocol reduces the DP proof to two cases. Let x be, as above, the number of occurrences of the index of the user being protected. If $x \leq T$, then adding Laplace noise provides DP and, if $x > T$, then the duplication provides DP. Let us now consider the amount of noise that the hybrid protocol adds for a multiplicity $x = T - 1$, i.e., when x is large, but not large enough to be protected by duplication. In the hybrid protocol, inputs with multiplicity x are protected exclusively by the per-multiplicity noise, even if they get duplicated *almost* as many times as necessary to achieve DP via duplication. This is unsatisfying as duplication in this case results in "wasted" communication overhead without improving privacy. To tackle this we will introduce a carefully calibrated Poisson noise to supplement duplication.

At a high-level, we introduce an intermediate regime (T, T'). For multiplicities up to T, we will use per-bucket noise; for multiplicities

larger than T', we will use (appropriately calibrated) duplication. We next describe how to protect inputs with multiplicity in (T, T').

Let x be a multiplicity in (T, T'). After duplication, the new multiplicity $x + \mathcal{U}^{\star x}$ is "spread out" in the interval $[x, \infty)$. In particular, this means that adding $O_{\varepsilon,\delta}(j)$ noise to each multiplicity $j \in [x, T']$ (as the per-multiplicity noising would do) is an overkill. Instead, that additional $O_{\varepsilon,\delta}(j)$ noise can be spread out analogously to how $x + \mathcal{U}^{\star x}$ is spread out. We achieve that by adding noise to each multiplicity j by a $\mathrm{Poi}(\eta_j)$ amount, where η_j's are carefully chosen parameters. Asymptotically this seems to improve the dependence of the required noise on δ and makes a practical improvement as shown in experiments.

The analysis for this approach is largely inspired by the work on shuffle DP of Feldman et al. [36]. In particular, we view the supplement Poisson noise as creating (a randomized number of) "clones" of x or $x-1$. Using properties of Poisson distributions, the number of these clones also follows the Poisson distribution; we show that DP is achieved as long as the expected number of clones is sufficiently large. This condition is then used to select our choice of η_j's both theoretically and numerically in experiments.

Relationship to prior work on DP anonymized histograms. Before we continue, let us mention that several works have considered the problem of releasing DP anonymized histograms in the central model [2, 12, 51, 52, 61, 72]. In fact, the optimal central DP algorithms [61, 72] also add separate noise for $x > T$ and $x \le T$, similar to our hybrid protocol. We stress however that our setting is much more challenging as we cannot add noise directly to $\mathcal{H}$; we may only create dummies with new indices or duplicate the existing ones, without knowing the (encrypted) true indices of the input. This is why we need to use novel noising schemes for our purposes. Another related work is Ghazi et al. [41]. Although the work is *not* for computing DP histograms, it also suffers from leakage in a form of the anonymized histogram, and uses a per-multiplicity noising to make the anonymized histogram DP. However, as explained in Section 4.2, this technique alone is insufficient for our setting and we have to develop a couple additional techniques (duplication and cloning) to make the protocol practical.

5 OUR PROTOCOLS

In this section we describe our main protocol for DP sparse histograms in detail, following the high-level approach outlined in Section 4. We split the target functionality (Figure 4) into two parts: We start in Section 5.1 by describing a thresholding functionality (Figure 2) and protocol (Figure 3) that allows the two servers to reveal the DP values among a set of encrypted values that pass a certain threshold τ. In Section 5.2, we then use that functionality inside our larger protocol (Figure 5) for computing a private histogram.

In both of the following subsections, we use the same structure: First, we describe the target functionality, followed by our protocol. Then, we define the leakage functionality of our protocol (see Definition 3), and prove that our protocol securely implements the target functionality with the given leakage. Finally, we prove that the output of the combined functionality (target functionality + leakage) provides DP.

Public parameters:
- Noise parameters λ, t and threshold τ.
- AHE scheme with public key PK_1.

Inputs:
P1: SK_1, the secret key corresponding to PK_1.
P2: Ciphtertexts $(w_i)_{i\in[n]}$, where each w_i has the form $\mathsf{Enc}(\mathsf{PK}_1, \mathsf{val}_i)$.

Funtionality:
(1) For $i = 1, \dots, n$:
  (a) $\xi_i^{(1)}, \xi_i^{(2)} \leftarrow_R \mathsf{TDLap}(\lambda, t), \xi_i \leftarrow \xi_i^{(1)} + \xi_i^{(2)}$
  (b) $\overline{\mathsf{val}}_i \leftarrow \begin{cases} \mathsf{val}_i + \xi_i & \text{if } \mathsf{val}_i + \xi_i \ge \tau, \\ 0 & \text{otherwise} \end{cases}$
(2) Return $(\overline{\mathsf{val}}_i)_{i\in[n]}$ to P1.

Figure 2: The target functionality $\mathcal{F}_{\mathsf{threshold}}$.

In Appendix E, we also describe another approach to private histograms based on a private heavy-hitters protocol, and compare it against our main protocol from this section.

5.1 Thresholding Protocol

Public parameters:
- Noise parameters λ, t.
- Threshold $\tau > 2t$.
- AHE scheme with public key PK_1.

Inputs:
P1: SK_1, the secret key corresponding to PK_1.
P2: Ciphertexts $(w_i)_{i\in[n]}$, where each w_i has the form $\mathsf{Enc}(\mathsf{PK}_1, \mathsf{val}_i)$.

Protocol:
(1) P2:
  (a) For each $i \in [n]$, add noise to the encrypted values using the homomorphic encryption properties:
  $$\mathcal{S}^{(1)} \leftarrow \left(\mathsf{Enc}(\mathsf{PK}_1, \mathsf{val}_i + \xi_i)\right)_{i\in[n]}$$
  where $\xi_i \leftarrow \mathsf{TDLap}(\lambda, t)$.
  (b) Send $\mathcal{S}^{(1)}$ to P1.
(2) P1:
  (a) For each record $w_i' \in \mathcal{S}^{(1)}$ received from P2:
    (i) Decrypt $\mathsf{val}_i' \leftarrow \mathsf{Dec}(\mathsf{SK}_1, w_i')$.
    (ii) Sample $\xi_i' \leftarrow \mathsf{TDLap}(\lambda, t)$, and compute $\mathsf{val}_i'' \leftarrow \mathsf{val}_i' + \xi_i'$.
    (iii) If $\mathsf{val}_i'' < \tau$, $\mathsf{val}_i'' \leftarrow 0$.
  (b) Set $\mathcal{S}^{(2)} \leftarrow \left(\mathsf{val}_i''\right)_{i\in[n]}$.
(3) P1 outputs $\mathcal{S}^{(2)}$.

Figure 3: Protocol $\Pi_{\mathsf{threshold}}$ that implements $\mathcal{F}_{\mathsf{threshold}}$ with leakage $\mathcal{L}_{\mathsf{threshold}}$.

In this section, as a warmup, we describe the thresholding protocol that underlies Steps (4) and (5) in Figure 1. Its ideal functionality is given in Figure 2. The inputs of P2 are homomorphically encrypted ciphertexts $(w_i)_{i\in[n]}$, and P1 holds the corresponding secret key. The functionality first, in Step (1a), samples noise from

a *truncated* centered discrete Laplace distribution that is added to each decrypted value. It then, in Step (1b), sets all values that are below the threshold τ to zero. Finally, in Step (2), the thresholded values are returned to both parties.

Our protocol for implementing the thresholding functionality is given in Figure 3. There are two sources of leakage in that protocol. First, each party keeps their own share of the noise value ξ_i that is added to each entry $i \in [n]$. This means that the parties can locally compute a version of the output that is *less noisy* than the ideal functionality output. Therefore, we have to include each party's respective noise share in the leakage. The second source of leakage comes from the fact that P1 learns all values with only P2's noise added *before* thresholding.

In the following formal description, we omit the parties' inputs for readability, e.g., we write $\mathcal{L}^{\mathsf{P1}}_{\mathsf{threshold}}$ to denote $\mathcal{L}^{\mathsf{P1}}_{\mathsf{threshold}}(x_0, x_1)$.

Definition 5 (Leakage of $\Pi_{\mathsf{threshold}}$). Let $\xi_i^{(1)}, \xi_i^{(2)}$ be the noise samples generated in Steps (2ii) and (1a) of Figure 2 respectively. Then we define leakages for $\Pi_{\mathsf{threshold}}$:

$$\mathcal{L}^{\mathsf{P1}}_{\mathsf{threshold}} = \left\{ \left(\xi_i^{(1)}\right)_{i\in[n]}, \left(\mathsf{val}_i + \xi_i^{(2)}\right)_{i\in[n]} \right\},$$
$$\mathcal{L}^{\mathsf{P2}}_{\mathsf{threshold}} = \left\{ \perp \right\}.$$

The *functionality with leakage* is defined to be the joint distribution $\left(\mathcal{F}_{\mathsf{threshold}}, \mathcal{L}^{\mathsf{Pi}}_{\mathsf{threshold}}\right)$, denoted $\hat{\mathcal{F}}_{\mathsf{threshold}}$.

Theorem 6. *The protocol $\Pi_{\mathsf{threshold}}$ in Figure 3, setting $\lambda_1 = \lambda$, securely implements $\mathcal{F}_{\mathsf{threshold}}$ from Figure 2 with the leakage $\mathcal{L}_{\mathsf{threshold}} = \left(\mathcal{L}^{\mathsf{P1}}_{\mathsf{threshold}}, \mathcal{L}^{\mathsf{P2}}_{\mathsf{threshold}}\right)$ in Definition 5.*

See Appendix B.1 for the proof.

Theorem 7. *Let $\lambda = 2\Delta/\varepsilon$ and $t = \Delta + \lambda \log(2/\delta)$. Then, for $i \in \{1, 2\}$, $\hat{\mathcal{F}}_{\mathsf{threshold}} = \left(\mathcal{F}_{\mathsf{threshold}}, \mathcal{L}^{\mathsf{Pi}}_{\mathsf{threshold}}\right)$ is an (ϵ, δ)-DP function on a database $(\mathsf{val}_j \in [\Delta])_{j\in[n]}$.*

Proof. The statement follows from the truncated Laplace mechanism as, once we condition on knowing the $\xi_i^{(1)}$'s (resp. $\perp$),P1 (resp. P2) observe a Laplace-noised histogram with parameter λ. Note that the thresholding by τ is then postprocessing. □

5.2 Private Sparse Histograms

We now describe our main protocol for private sparse histograms. We give a formal description of the target functionality in Figure 4. It closely follows the high-level description in Section 3, with the main difference being that we explicitly split up the noise terms ξ_i into two components, one of which is leaked to each helper server through the thresholding protocol from the previous section.

Our main protocol is in Figure 5. It follows the outline in Figure 1. In Step (1), each client i starts by preparing three ciphertexts from its $(\mathsf{ind}_i, \mathsf{val}_i)$ pair: One containing an encryption of the hash of ind_i, which is going to be used to obtain an OPRF value for ind_i. The second encrypts ind_i (without the hash). This is used to recover the cleartext bucket IDs that pass the threshold after aggregating. Finally, the clients generate homomorphic encryptions of their values, using P1's AHE public key, and then again encrypting the resulting ciphertext under P2's public key using standard encryption. This

Public parameters:
- DP parameters ε, δ, sensitivity Δ.
- Noise parameters $\lambda = 2\Delta/\varepsilon$ and $t = \Delta + \lambda \log(2/\delta)$.
- Threshold $\tau = \Delta + 2t + 1$.

Inputs:
Clients: Index-value pairs $(\mathsf{ind}_i, \mathsf{val}_i)_{i\in[n]}$

Funtionality:
(1) Let $(\mathsf{ind}'_j)_{j\in[n']}$ denote the unique indices in the input. For each $j \in [n']$:
  (a) Sample $\xi_j^{(1)}, \xi_j^{(2)} \leftarrow_R \mathsf{TDLap}(\lambda, t)$.
  (b) Compute $\xi_j \leftarrow \xi_j^{(1)} + \xi_j^{(2)}$ and

$$\mathsf{val}'_i = \begin{cases} \perp & \text{if } s_i + \xi_i < \tau \\ s_i + \xi_i & \text{otherwise,} \end{cases}$$

  where $s_i = \sum_{\{j \mid \mathsf{ind}_j = \mathsf{ind}'_i\}} \mathsf{val}_j$.
(2) Output $\{(\mathsf{ind}'_j, \mathsf{val}'_j) \mid j \in [n'], \mathsf{val}'_j \neq \perp\}$ to P1.

Figure 4: The target private histograms functionality $\mathcal{F}_{\mathsf{hist}}$.

allows P2 to homomorphically add up client contributions belonging to the same bucket, while hiding the values from P1 until they are aggregated via the outer encryption layer.

In Step (2), P1 exponentiates each first component with its secret PRF key K, to hide the cleartext indices from P2. It then proceeds to add dummy values as discussed in Section 4, using encryptions of zero as the third component to ensure the dummies do not add to the aggregated values. We describe the details of the dummy sampling algorithm in Figures 6–9. The resulting set of ciphertexts is shuffled and then sent to P2.

P2 can now in Step (3) decrypt two of the three components. After decryption in Step (3a),

- h'_i is equal to $\mathsf{H}(u'_i)^{\mathsf{K}}$,
- w'_i is similarly equivalent to $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{PK}_{\mathsf{AHE}}, v'_i)$,

for some index-value pair (u'_i, v'_i), either contributed by a client, or added by P1 as a dummy.

After decrypting, P2 homomorphically aggregates all third components of triples that share the same first component, choosing one of the second components arbitrarily. Now observe that at this point, the number of aggregate buckets is not differentially private from P1's view, since P1 knows exactly how many dummy buckets were added in Step (2a). To account for this, P2 has to add additional dummy buckets, which is done in the call to SampleBuckets in Step (3d). After that, the parties invoke $\Pi_{\mathsf{threshold}}$ on the aggregated values, obtaining the cleartext values above the threshold τ. We have chosen to set τ to be more than $\Delta + 2t_1$ to guarantee that the dummies added by P2 in Step (2a) are always below τ even after adding two TDLap samples. Note that this can potentially be optimized if we allow dummies to be above the threshold with probability $2^{-\sigma}$ for a statistical security parameter σ. We leave this optimization for future work.

Finally, in Steps (6)-(7), P2 and P1 jointly decrypt the second components corresponding to values above the threshold, which allows P1 to obtain the cleartext indices for those values. Note that

Public parameters:

- Group $\mathbb{G}$ of prime order q with generator g.
- Histogram index domain $\mathcal{U} = \{0, ..., 2^d - 1\}$, value domain $\mathcal{V} = \{0, \ldots, \Delta\}$. Dummy index domain $\mathcal{I}$ with $\mathcal{U} \cap \mathcal{I} = \emptyset$.
- Random oracle $\mathsf{H} : \mathcal{U} \cup \mathcal{I} \to \mathbb{G}$.
- ElGamal public encryption keys $\mathsf{PK} = \mathsf{PK}_1 \cdot \mathsf{PK}_2, \mathsf{PK}' \in \mathbb{G}$. Public encryption key $\mathsf{PK}_{\mathsf{AHE}}$ for additive homomorphic encryption. Public encryption key for semantically secure encryption PK''.
- DP parameters $\varepsilon = \varepsilon_{\text{leakage}} + \varepsilon_{\text{counts}}, \delta = \delta_{\text{leakage}} + \delta_{\text{counts}}$, sensitivity Δ.
- Free Noise Parameters T, T' chosen by grid search in Section 6.2.
- Determined Noise Parameters $\lambda_1 = 2\Delta/\varepsilon_{\text{counts}}$, $t_1 \geq \Delta + \lambda_1 \log(2/\delta_{\text{counts}})$, $\lambda_2 = 1/\varepsilon_{\text{leakage}}$, $t_2 \geq \lambda_2 \log(1/\delta_{\text{leakage}})$, threshold $\tau = \Delta + 2t_1 + 1$.

Inputs:

Client_i**:** an index-value pair $(u_i, v_i) \in \mathcal{U} \times \mathcal{V}$.

P1**:** ElGamal secret key $\mathsf{SK}_1 \in \mathbb{Z}_q$ is the secret key for PK_1, additive HE secret key $\mathsf{SK}_{\mathsf{AHE}}$ corresponding to $\mathsf{PK}_{\mathsf{AHE}}$, a secret PRF key $\mathsf{K} \leftarrow \mathbb{Z}_q$

P2**:** ElGamal secret keys $\mathsf{SK}_2, \mathsf{SK}' \in \mathbb{Z}_q$, where SK_2 is the secret key for PK_2, and SK' is the secret for PK', and decryption key SK'' corresponging to PK'' for semantically secure encryption.

Protocol:

(1) Each Client_i computes $h_i \leftarrow \mathsf{H}(u_i)$ and $w_i \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{PK}_{\mathsf{AHE}}, v_i)$ and encrypts.

$$(a_i, b_i, c_i) \leftarrow \left(\mathsf{Enc}_{\mathsf{ElGamal}}(\mathsf{PK}', h_i), \mathsf{Enc}_{\mathsf{ElGamal}}(\mathsf{PK}, u_i), \mathsf{Enc}(\mathsf{PK}'', w_i)\right).$$

(2) P1 receives the ciphertexts from all clients $\mathcal{S}^{(1)} = \{(a_i, b_i, c_i)\}_{i \in [n]}$.

(a) $\mathcal{S}^{(1)} \leftarrow \mathcal{S}^{(1)} \cup \mathsf{SampleDummies}(T, T', \varepsilon_{\text{leakage}}, \delta_{\text{leakage}}, \mathcal{S}^{(1)})$.

(b) Choose random $\mathsf{K} \leftarrow_R \mathbb{Z}_q$ and set $\mathcal{S}^{(2)} \leftarrow \emptyset$.

(c) For every tuple $(a_i, b_i, c_i) \in \mathcal{S}^{(1)}$:

(i) Unpack $(\mathsf{ct}_1, \mathsf{ct}_2) \leftarrow a_i$.

(ii) $a_i' \leftarrow (\mathsf{ct}_1^{\mathsf{K}}, \mathsf{ct}_2^{\mathsf{K}})$.

(iii) $\mathcal{S}^{(2)} \leftarrow \mathcal{S}^{(2)} \cup \{(a_i', b_i, c_i)\}$.

(d) Shuffle $\mathcal{S}^{(2)}$ and send the result to P2.

(3) P2 received $\mathcal{S}^{(2)} = \{(a_i', b_i', c_i')\}_{i \in [n']}$ from P1 and computes:

(a) Decrypts all three components of each ciphertext as follows

$$h_i' \leftarrow \mathsf{Dec}_{\mathsf{ElGamal}}(\mathsf{SK}', a_i'), w_i' \leftarrow \mathsf{Dec}(\mathsf{SK}'', c_i')$$

and sets $\mathcal{S}^{(3)} = \left\{\left(h_i', b_i', w_i'\right)\right\}_{i \in [n']}$.

(b) Partitions the tuples in $S^{(3)}$ based on the first component by defining $\mathcal{H}_i = \{j \mid h_j' = h_i'\}$.

(c) For each unique value h_i' in the first components of the tuples in $\mathcal{S}^{(3)}$, homomorphically adds up all third components and chooses one of the second components at random. This results in a set (ordered by h_i'): $\mathcal{S}^{(4)} = \left((h_i'', d_i', w_i'')\right)_{i \in [n'']}$ containing $n'' \leq n'$ tuples of the form

$$\left(\mathsf{H}(u_i'')^{\mathsf{K}}, \mathsf{Enc}_{\mathsf{ElGamal}}(\mathsf{PK}, u_i''), \mathsf{Enc}_{\mathsf{HE}}\left(\mathsf{PK}_{\mathsf{AHE}}, \sum_{j \in \mathcal{H}_i} v_j''\right)\right).$$

(d) $\mathcal{S}^{(4)} \leftarrow \mathcal{S}^{(4)} \cup \mathsf{SampleBuckets}(\mathsf{PK}_{\mathsf{AHE}}, \lambda_2, t_2)$.

(e) Shuffle $\mathcal{S}^{(4)}$.

(4) Let $(d_i')_{i \in [n'']}$ and $(w_i'')_{i \in [n'']}$ be the ordered sets (in the same order) of the second and third components of $\mathcal{S}^{(4)}$. P1 and P2 invoke $\Pi_{\mathsf{threshold}}$ from Figure 3 where P1 has input $\mathsf{SK}_{\mathsf{AHE}}$ and P2 has inputs $(w_i'')_{i \in [n'']}$, setting τ, λ_1, and t_1 as above. Let $(\mathsf{val}_i)_{i \in [n'']}$ be the output that P1 receives. Moreover, P2 sends $\left(\mathsf{Randomize}_{\mathsf{ElGamal}}(d_i')\right)_{i \in [n'']}$ them to P1.

(5) Let $V = \left\{(d_i', \mathsf{val}_i) \mid i \in [n''], \mathsf{val}_i \neq 0\right\}$ be the index - value pairs (with encrypted index) obtained by P1 in the previous step, excluding pairs with value 0. P1 sends to P2 the following set, randomly shuffled.

$$D = \{\mathsf{Randomize}_{\mathsf{ElGamal}}(d) \mid (d, v) \in V\}.$$

(6) P2 computes $\{\mathsf{PartialDec}_{\mathsf{ElGamal}}(\mathsf{SK}_2, d) \mid d \in D\}$ and sends it to P1 in the same order.

(7) P1 reverts the shuffle on D from Step 5 and outputs $\{(\mathsf{Dec}_{\mathsf{ElGamal}}(\mathsf{SK}_1, d), v) \mid (d, v) \in V\}$.

Figure 5: Our full protocol Π_{hist} for computing private histograms.

Parameters: Thresholds T and T', privacy parameters $\varepsilon_{\text{leakage}}$ and δ_{leakage}, dummy index domain $\mathcal{I}$, a set of messages $\mathcal{S} = \{(a_i, b_i, c_i)\}_{i\in[n]}$.
Algorithm:
(1) Find $\lambda_3, t_3, r, p, T''$ and $\{\eta_j\}_{T\le j\le T''}$ to satisfy Theorem 15 with $T, T', \varepsilon = \varepsilon_{\text{leakage}}/2$, $\delta = \delta_{\text{leakage}}/2(1+\exp(\varepsilon))$ and $\hat{\delta} = \delta_{\text{leakage}}/2$.
(2) $\mathcal{S} \leftarrow \mathcal{S} \cup \mathsf{SampleFrequencyDummies}(\lambda_3, t_3, T, \mathcal{I})$.
(3) $\mathcal{S} \leftarrow \mathcal{S} \cup \mathsf{SampleDuplicateDummies}(\mathcal{S}, r, p)$.
(4) $\mathcal{S} \leftarrow \mathcal{S} \cup \mathsf{SampleBlanketDummies}(\{\eta_j\}_j, T, T'')$.
(5) Return $\mathcal{S}$.

Figure 6: Algorithm SampleDummies**.**

Parameters: Threshold T, noise parameters λ_3 and t_3, dummy index domain $\mathcal{I}$
Algorithm:
(1) $\mathcal{R} \leftarrow \emptyset$
(2) For every $i = 1, \ldots, T$:
  (a) Randomly draw N_i from $\mathsf{TSDLap}(\lambda_3, t_3)$.
  (b) For $j = 1, \ldots, N_i$:
    (i) Randomly select $x' \leftarrow_R \mathcal{I}$.
    (ii) Perform Step 1 of Figure 5 i times, simulating a client with input $(x', 0)$. Add the resulting ciphertexts to $\mathcal{R}$.
(3) Return $\mathcal{R}$.

Figure 7: Algorithm SampleFrequencyDummies**.**

Parameters: AHE Public key PK, noise parameters r and p, dummy index domain $\mathcal{I}$, a set of messages $\mathcal{S} = \{(a_i, b_i, c_i)\}_{i\in[n]}$, where all b_i are re-randomizeable ElGamal ciphertexts.
Algorithm:
(1) $\mathcal{R} \leftarrow \emptyset$
(2) For every $i \in [n]$:
  (a) Randomly draw N_i from $\mathsf{NBin}(r, p)$.
  (b) For $j = 1, \ldots, N_i$ add $(a_i, b'_j, \mathsf{Enc}_{\mathsf{AHE}}(\mathsf{PK}, 0))$ to $\mathcal{R}$, where $b'_j = \mathsf{Randomize}_{\mathsf{ElGamal}}(b_j)$.
(3) Return $\mathcal{R}$.

Figure 8: Algorithm SampleDuplicateDummies**.**

Parameters: Public key PK, noise intensities η_j, dummy index domain $\mathcal{I}$, thresholds T, T''.
Algorithm:
(1) $\mathcal{R} \leftarrow \emptyset$
(2) For every $T \le j \le T''$:
  (a) Repeat $\mathsf{Poi}(\eta_j)$ times.
    (i) Randomly select $x' \leftarrow_R \mathcal{I}$.
    (ii) Perform step 1 of fig. 5 j times, simulating a client with input $(x', 0)$. Add the resulting ciphertexts to $\mathcal{R}$.
(3) Return $\mathcal{R}$.

Figure 9: Algorithm SampleBlanketDummies**.**

Public parameters:
- Boundary T.
- Noise distribution parameters $\lambda_1, \lambda_2, \lambda_3, t_1, t_2, t_3, r, p$ and η_i for $i \ge T$

Inputs:
Clients: Index-value pairs $\mathcal{I} = (\mathsf{ind}_i, \mathsf{val}_i)_{i\in[n]}$.
Functionality:
(1) Let $\mathcal{H}^0$ be an anonymized histogram of the first components of the inputs. That is, $\mathcal{H}^0_i$ denotes the number of distinct buckets in the input that appear exactly i times.
(2) Initialize $\mathcal{N}^1, \mathcal{N}^3, \mathcal{N}^4$ to $\emptyset$.
(3) Initialize $\mathcal{H}^1 \leftarrow \mathcal{H}^0$. For every $i = 1, \ldots, T$:
  (a) Draw $N_i \leftarrow_R \mathsf{TSDLap}(\lambda_3, t_3)$.
  (b) $\mathcal{N}^1 \leftarrow \mathcal{N}^1 \cup \{(i, N_i)\}$
  (c) $\mathcal{H}^1_i \leftarrow \mathcal{H}^1_i + N_i$
(4) Initialize $\mathcal{H}^2$ to an empty histogram.
  For every i s.t. $\mathcal{H}^1_i \neq 0$, repeat $\mathcal{H}^1_i$ times:
  (a) Draw $a \leftarrow_R \mathsf{NBin}(i \cdot r, p)$.
  (b) $\mathcal{H}^2_{i+a} \leftarrow \mathcal{H}^2_{i+a} + 1$
(5) Initialize $\mathcal{H}^3 \leftarrow \mathcal{H}^2$. For every $T \le i \le T''$:
  (a) Draw $N'_i \leftarrow_R \mathsf{Poi}(\eta_i)$
  (b) $\mathcal{N}^3 \leftarrow \mathcal{N}^3 \cup \{(i, N'_i)\}$
  (c) $\mathcal{H}^3_i \leftarrow \mathcal{H}^3_i + N'_i$
(6) Let $\mathcal{I}' = \left((\mathsf{ind}'_i, \mathsf{val}'_i)\right)_{i\in[n']}$ be the input $\mathcal{I}$ grouped by first component, summing up the second components, shuffled. For each $j \in [\Delta]$:
  (a) Draw M_j from $\mathsf{TSDLap}(\lambda_2, t_2)$
  (b) $\mathcal{N}^4 \leftarrow \mathcal{N}^4 \cup \{(j, M_j)\}$
  (c) $\mathcal{I}' \leftarrow \mathcal{I}' \,||\, \left((\perp, j)\right)^{M_j}$
(7) Let $\xi^{(1)}, \xi^{(2)} \leftarrow_R \mathsf{TDLap}(\lambda_1, t_1)^{|\mathcal{I}'|}$, and let $\xi = \xi^{(1)} + \xi^{(2)}$.
(8) Let $\mathcal{I}'' = \left((\mathsf{ind}_j, \mathsf{val}_j + \xi_j) \mid j \in [|\mathcal{I}'|], \mathcal{I}'_j = (\mathsf{ind}_j, \mathsf{val}_j)\right)$.
(9) Define $\mathcal{F}^{\mathsf{P1}}_{\mathsf{hist}} \leftarrow \{(\mathsf{ind}, \mathsf{val}) \mid (\mathsf{ind}, \mathsf{val}) \in \mathcal{I}'', \mathsf{val} \ge \tau\}$.
(10) Define $\mathcal{F}^{\mathsf{P2}}_{\mathsf{hist}} \leftarrow \emptyset$.
(11) Define $\mathcal{V}^{\perp} \leftarrow \{\mathsf{val} \mid (\mathsf{ind}, \mathsf{val}) \in \mathcal{I}'', \mathsf{val} < \tau\}$.
(12) $\mathcal{L}^{\mathsf{P1}}_{\mathsf{hist}} \leftarrow \left(\mathcal{N}^1, \mathcal{N}^3, \xi^{(1)}, \mathcal{V}^{\perp}\right)$, $\mathcal{L}^{\mathsf{P2}}_{\mathsf{hist}} \leftarrow \left(\mathcal{H}^3, \mathcal{N}^4, \xi^{(2)}\right)$.
(13) The *functionality with leakage* $\hat{\mathcal{F}}_{\mathsf{hist}}$ is defined to be the joint distribution $(\hat{\mathcal{F}}^{\mathsf{P1}}_{\mathsf{hist}}, \hat{\mathcal{F}}^{\mathsf{P2}}_{\mathsf{hist}})$ with $\hat{\mathcal{F}}^{\mathsf{Pi}}_{\mathsf{hist}} = (\mathcal{F}^{\mathsf{Pi}}_{\mathsf{hist}}, \mathcal{L}^{\mathsf{Pi}}_{\mathsf{hist}})$.

Figure 10: Functionality $\hat{\mathcal{F}}_{\mathsf{hist}} = (\mathcal{F}_{\mathsf{hist}}, \mathcal{L}_{\mathsf{hist}})$. We show that Π_{hist} securely implements $\mathcal{F}_{\mathsf{hist}}$ with leakage $\mathcal{L}_{\mathsf{hist}}$.

we need to do this while preventing P2 from linking the decrypted indices to the pseudorandom buckets for which it did aggregation.

Next, in Figure 10, we define the *functionality with leakage*, $\hat{\mathcal{F}}_{\mathsf{hist}}$ for our protocol. While $\mathcal{F}_{\mathsf{hist}}$ is already described in Figure 4, we include it in Figure 10 to make the correlation between $\mathcal{F}_{\mathsf{hist}}$ and $\mathcal{L}_{\mathsf{hist}}$ explicit. The definition of $\hat{\mathcal{F}}_{\mathsf{hist}}$ follows Π_{hist} (Figure 5). The components of the leakage correspond to the noisy (anonymized) histograms revealed to both parties throughout the protocol, where one party learns the noise samples, and the other learns the noisy histogram. Steps (3)–(5) in $\hat{\mathcal{F}}_{\mathsf{hist}}$ correspond to the call to SampleDummies in Step (2a) of Π_{hist}, whereas Step (6) in $\hat{\mathcal{F}}_{\mathsf{hist}}$ corresponds to the call to SampleBuckets in Step (3d) of Π_{hist}. Note that $\mathcal{H}^4$ is not revealed to P1 directly, but only after adding noise to each entry, as

Parameters: AHE Public key PK, noise parameters λ_2, t.
Algorithm:
(1) $\mathcal{R} \leftarrow \emptyset$
(2) For $j \in [\Delta]$:
  (a) Sample $M_j \leftarrow \mathsf{TSDLap}(\lambda_2, t)$.
  (b) For $k \in [M_j]$, generate dummy record $(\perp, \perp, \mathsf{Enc}(\mathsf{PK}, j))$, and concatenate these dummies with $\mathcal{R}$.
(3) Return $\mathcal{R}$.

Figure 11: Algorithm SampleBuckets**.**

part of $\mathcal{L}^{\mathsf{P1}}_{\mathsf{threshold}}$. Steps (7)–(9) correspond to the call to $\Pi_{\mathsf{threshold}}$ in Step (4) of Π_{hist}. Note that all dummy buckets added by P2 in Step (6) will be below $\tau = \Delta + 2t_1 + 1$, and so no dummy buckets will appear in $\mathcal{F}^{\mathsf{P1}}_{\mathsf{hist}}$. In Step (12) we define the leakage $\mathcal{L}_{\mathsf{hist}}$ to both parties. Note that we make the output of $\Pi_{\mathsf{threshold}}$ to P2 part of the leakage here, since P2 does not have any output in $\mathcal{F}_{\mathsf{hist}}$.[3]

The following theorem establishes the security property of our protocol, i.e., that it implements the intended functionality with the intended leakage to each of the parties. Afterwards, we show that the joint distribution of output and leakage is DP.

Theorem 8. *Protocol Π_{hist} (Figure 5) securely implements $\mathcal{F}_{\mathsf{hist}}$ (Figures 4, 10) with leakage $\mathcal{L}_{\mathsf{hist}}$ (Figure 10). Note that ε and δ in Figure 4 correspond to ε_{counts} and δ_{counts} in Figures 4 and 10.*

Clients colluding with servers. Note that the above security arguments still hold when either P1 or P2 collude with any number of semi-honest clients, as clients do not receive any messages, and for any input the distribution of the outputs will be the same in the ideal and real model. However, as mentioned in Section 2.3, our protocol does not protect against malicious clients. In particular, observe that our protocol requires the values $\mathsf{H}(\mathsf{ind}_i)^{\mathsf{K}}$ to be indistinguishable from random. Now consider an adversary controlling P2 and client j. If the client is allowed to deviate from the protocol, instead of submitting an encryption of $\mathsf{H}(\mathsf{ind}_j)$ as its first component, it could submit $\mathsf{H}(\mathsf{ind}'_j)^{\frac{1}{2}}$ for a target ind'_j. After P2 obtains $(a'_i)_{i\in[n']} = \left(\mathsf{H}(\mathsf{ind}_i)^{\mathsf{K}}\right)_{i\in[n']}$, it can now, for all $i \in [n']$, square a'_i and check if $a'^2_i = a'_{i'}$ for any $i' \neq i$. If that is the case, then the adversary knows that $a'_{i'} = \mathsf{H}(\mathsf{ind}_j)^{\mathsf{K}}$ and that at least one other client submitted ind_j.

The above attack could be averted by having the client prove that it indeed evaluated H correctly, using a ZKP-friendly hash function as for example proposed by Grassi et al. [49]. However, this still allows malicious clients to poison the result by providing arbitrary large val_j. We leave a complete extension of our protocol to malicious clients for future work.

5.2.1 The Privacy Guarantees. We now state that the outputs to both parties combined with either of the leakages is DP. See Appendices B.3 and B.4 for the proofs.

Lemma 9. $(\mathcal{F}^{\mathsf{P1}}_{\mathsf{hist}}, \mathcal{L}^{\mathsf{P1}}_{\mathsf{hist}}, \mathcal{F}^{\mathsf{P2}}_{\mathsf{hist}})$ *is (ε, δ)-DP.*

Lemma 10. $(\mathcal{F}^{\mathsf{P2}}_{\mathsf{hist}}, \mathcal{L}^{\mathsf{P2}}_{\mathsf{hist}}, \mathcal{F}^{\mathsf{P1}}_{\mathsf{hist}})$ *is (ε, δ)-DP.*

[3] We could choose to reveal the output histogram to both parties in $\mathcal{F}_{\mathsf{hist}}$. Since this would require an additional communication round, we instead only have P1 learn the output and regard the noisy values P2 learns in the thresholding protocol as leakage.

5.2.2 Privacy Analysis of $\mathcal{H}^3$. To analyze the privacy of $\mathcal{H}^3$, we work mainly with the add/remove notion of DP; at the end, we will transfer our result to substitution DP using the relationship between the two described in Section 2.1.

Let $\mathcal{D}$ be a database and $\mathcal{D}'$ be the same but with the first entry removed. Let $\mathcal{H}^3$ be computed from $\mathcal{D}$ and $\mathcal{H}^{3\prime}$ from $\mathcal{D}'$ by the process in Figure 10. Recall that $\mathcal{H}^3$ corresponds to Algorithm 6, run by P1, and corresponding to the call to SampleDummies in Step (2a) of Π_{hist} (Figure 5). Let m_1 be the multiplicity of ind_1 in $\mathcal{D}$ and $m'_1 = m_1 - 1$ be the multiplicity in $\mathcal{D}'$.

Hybrid protocol: per-multiplicity and duplication. We will start by analyzing the hybrid method without Poisson supplement noise described in the overview, i.e., the case $T = T'$ of our final improved method described in Section 4. As outlined in Section 4, our analysis consists of two cases, based on whether the count is below T or above T'. In the former, the privacy guarantee follows that of the truncated Laplace mechanism (e.g., [30]), which gives:

Lemma 11 (Privacy for Low Counts). *$\mathcal{H}^3$ and $\mathcal{H}^{3\prime}$ are (ε, δ)-indistinguishable if $m_1 \leq T$, $\lambda \geq 2/\varepsilon$ and $t \geq 1 + \lambda \log(2/\delta)$.*

For the latter case, as outlined in Section 4, we can show that $1 + \mathcal{U}^{\star(T'+1)} \equiv_{\varepsilon,\delta} \mathcal{U}^{\star T'}$ where $\mathcal{U} = \mathsf{NBin}(r, p)$. Using the tail bound of negative binomial noise, we can select concrete parameters as follows.

Lemma 12 (Privacy for High Counts). *Let $\varepsilon, \delta \in (0, 1)$. $\mathcal{H}^3$ and $\mathcal{H}^{3\prime}$ are (ε, δ)-indistinguishable if $m_1 > T'$ where $T' \geq 3\left(1 + \log(2/\delta)\right) \cdot \left(\frac{4}{\varepsilon}\left(1 + \log(1/\varepsilon)\right) + \frac{100}{\varepsilon^2}\right)$, $r = \frac{3(1+\log(2/\delta))}{T'}$ and $p = e^{-0.2\varepsilon}$.*

These lemmas are sufficient to prove privacy for the hybrid method without Poisson supplement noise (i.e., if we take $T = T'$ and $\eta_j = 0$ for all j). In this case, we get the following theorem.

Theorem 13. *Let $\varepsilon, \delta \in (0, 1)$. If $T = T' \geq 3\left(1 + \log(2/\delta)\right) \cdot \left(\frac{4}{\varepsilon}\left(1 + \log(1/\varepsilon)\right) + \frac{100}{\varepsilon^2}\right)$, $\lambda \geq 2/\varepsilon$, $t \geq 1 + \lambda \log(2/\delta)$, $r = \frac{3(1+\log(2/\delta))}{T'}$ and $p = e^{-0.2\varepsilon}$, then the algorithm given in Figure 6 is (ε, δ)-add/remove DP. It is then immediate that it is $(2\varepsilon, (1 + \exp(\varepsilon))\delta)$-DP.*

Furthermore, for $\log(1/\delta)/\varepsilon = o(n^{1/3})$, if we take $T(= T') = \Theta(n^{1/3})$, then the expected number of dummy messages generated and sent in step (2) is $\Theta(n^{2/3} \log(1/\delta)/\varepsilon)$

To cover the case $T < m_1 < T'$, we must make use of the η_j. Let $N_i \sim \mathsf{NBin}(ri, p)$ and let $\tau_{i,j} = \mathbb{P}(N_i = j - i)$ (this is the probability that a message with multiplicity i is duplicated to have multiplicity j). To consider the change made by increasing i by one, we take the smallest q that allows the following breakdown, with $\alpha_{i,j}, \beta_{i,j}$ and $\gamma_{i,j}$ distributions:

$$\tau_{i,j} = q_i \alpha_{i,j} + (1 - q_i)\gamma_{i,j}, \tag{1}$$

and

$$\tau_{i+1,j} = q_i \beta_{i,j} + (1 - q_i)\gamma_{i,j}. \tag{2}$$

Further, define μ_i to be the smallest μ such that if $A, B, C \sim \mathsf{Poi}(\mu)$ are independent then

$$\mathbb{P}\left(\frac{q_i A + (1 - q_i)C + 1}{q_i B + (1 - q_i)C} > \exp\varepsilon\right) \leq \delta. \tag{3}$$

The main privacy guarantee of this approach is stated below. As mentioned earlier, this proof employs techniques from [36] by

viewing each supplement Poisson noise as a "clone". The full proof is deferred to Appendix B.8.

Lemma 14. *If for all j*

$$\eta_j \geq \mu_{m'_1}(\alpha_{m'_1,j} + \beta_{m'_1,j} + \gamma_{m'_1,j}), \tag{4}$$

then $\mathcal{H}^3$ and $\mathcal{H}^{3\prime}$ are (ε, δ)-indistinguishable

The final privacy guarantee is stated in the following theorem, which is a straightforward combination of Lemmas 11, 12, and 14. It is shown in our experiments that this protocol in practice achieves an improvement over the $T = T'$ case in communication. We also provide a heuristic argument that the asymptotics are improved by at least a factor of $\log(1/\delta)^{1/3}$ in Appendix C.

Theorem 15. *Let $\varepsilon, \delta \in (0,1)$. For given T, T', let $\lambda \geq 2/\varepsilon$, $t \geq 1 + \lambda \log(2/\delta)$, $r = \frac{3(1+\log(2/\delta))}{T'}$, $p = e^{-0.2\varepsilon}$. Further, let*

$$\eta_j = \max_{T<i<T'} \mu_i(\alpha_{i,j} + \beta_{i,j} + \gamma_{i,j}), \tag{5}$$

for all j, and choose T'' so that $\sum_{j>T''} \eta_j \leq \hat{\delta}$.

Then so long as $T' \geq 3\,(1 + \log(2/\delta)) \cdot \left(\frac{4}{\varepsilon}\,(1 + \log(1/\varepsilon)) + \frac{100}{\varepsilon^2}\right)$, then $\mathcal{H}^3$ is (ε, δ)-add/remove DP. It is then immediate that it is $(2\varepsilon, (1 + \exp(\varepsilon))\delta + \hat{\delta})$-DP.

6 EXPERIMENTAL EVALUATION

To evaluate our protocol, we optimize the DP parameters to minimize communication. We then perform microbenchmarks to measure the computation time needed for each encryption scheme used. The code needed to reproduce our results is available at https://github.com/google-research/sparse_dp_histograms.

6.1 Encryption Schemes

We compare two instantiations of AHE: the *Paillier* cryptosystem [29, 66], and the *exponential ElGamal* cryptosystem [28, 38]. The former is a well-known choice for AHE when ciphertext size is a concern as in our case. The latter offers even faster encryption and homomorphic operations, but the cost of decryption scales with the encrypted value. This happens since x is encrypted as $\mathsf{Enc}_{\mathsf{ElGamal}}(g^x)$, where g is a generator of the underlying group. As a result, two ciphertexts can be homomorphically added by multiplying them, but now standard ElGamal decryption only yields g^x, and so solving for x requires a discrete logarithm. Note however that we have an upper bound on x, since (up to the small additive noise term) the value of every bucket will be less than n. To decrypt all buckets, we can simply pre-compute $(g^i)_{i\in[n]}$, and use it to look up the decryption value. Therefore, amortized across all user contributions, in the worst case, decryption requires a single exponentiation in addition to standard ElGamal decryption. Note that using Shanks's algorithm [71] and its optimizations for elliptic curve groups [22, 37], this cost can be further reduced for large n.

6.2 Computation and Communication Costs

In this section we report concrete computation and communication costs for our protocol, as well as for the baseline solution based on garbled circuits.

Microbenchmarks. In Table 1, we present CPU time microbenchmarks for the cryptographic operations used in our protocol, as well as the corresponding ciphertext sizes. All experiments were run on a Intel Xeon Platinum 8373C CPU @ 2.60GHz, single core. We use an open-source implementation of Paillier and ElGamal for our microbenchmarks [48]. To compare against the baseline protocol using garbled circuits, we implement Figure 14 using the EMP framework [75].

	ElGamal	Paillier	Exp. ElGamal
Ciphertext size	64 bytes	256 bytes	64 bytes
Encrypt	102µs ± 1%	921µs ± 2%	152µs ± 1%
Randomize (offline)	101µs ± 1%	—	—
Randomize (online)	512ns ± 1%	—	—
Hom. Add	—	2.30µs ± 8%	537ns ± 2%
Exp	101µs ± 0%	—	—
Decrypt	50.7µs ± 1%	369µs ± 1%	101µs ± 1%

Table 1: CPU microbenchmarks for the cryptographic operations we need. For exponential ElGamal decryption, we report the worst-case, amortized per-user cost.

Noise Distributions and Parameters. Recall that our approach is parameterized by noise distributions for per-entry noising and duplication. For the former we use $\mathsf{TSDLap}(\cdot)$ and for the latter we use $\mathsf{NBin}(\cdot)$. Moreover, our protocol takes a threshold T, along with the parameters of the above distributions to provide privacy (according to Theorem 15).

For every choice of n (number of clients) and the privacy parameters ϵ, δ, we do a grid search over the values of $T, t, r, p, \{\eta_i\}_{i\in[n]}$ that results in a secure instance of the protocol[4], and pick the configuration that minimizes overall server communication, i.e., minimizes the number of dummy contributions inserted in step (2) in Figure 5. In our experiments, we distribute the privacy budget (ϵ, δ) using basic composition as $\varepsilon_{\text{counts}} = \varepsilon_{\text{leakage}} = \varepsilon/2$ (and analogously for δ) for each step as described in Figure 5. This privacy budget split is fixed for all our experiments. Throughout this section, we assume $\Delta = 1$.

Communication costs. Recall that in our protocol the clients do a one-shot participation to initiate the protocol, and do not need to stay online during the protocol execution. The communication cost of our protocol for the client is 192 bytes (so long as domain elements fit in a single ciphertext, independent of ε and n). For the remainder, we study the server communication costs. In Figure 12 (left) and Table 2 we report an upper bound on expected total communication costs for any execution of our protocols. In other words, the reported communication costs are for a worst-case input distribution that maximizes communication, and could be lower in practice. For example, when grouping by pseudoindices in step (4) of Figure 5, we assume that the number of different indices in the input is n, which results in the maximum possible communication for that step.

Figure 12 (left) shows that our protocols clearly outperform the baseline protocol based on garbled circuits. In fact, communication costs for the baseline are prohibitive, requiring over $400KB$ of server communication per client for $n = 10^5$ clients, with per client cost that increases with the number of clients (the setting with $n = 10^6$

[4]We use a "numerical" version of Lemma 12, stated as Lemma 18 in the appendix, together with the setting of η_j's as in Theorem 15.

ε	n	P1 offline	P1 online	P1 total	P1 comm.	P2 offline	P2 online	P2 total	P2 comm.	total time	total comm.
0.5	10^5	3.67	0.93	4.60	1539	0.28	0.82	1.10	141	5.70	1680
	10^6	1.15	0.36	1.51	482	0.26	0.26	0.51	130	2.02	612
	10^7	0.70	0.26	0.96	294	0.25	0.16	0.41	128	1.37	422
	10^8	0.56	0.23	0.78	234	0.25	0.13	0.38	128	1.16	362
	10^9	0.50	0.21	0.72	211	0.25	0.11	0.37	128	1.08	339
1.0	10^5	2.11	0.58	2.68	883	0.26	0.47	0.73	133	3.42	1016
	10^6	0.91	0.31	1.22	383	0.26	0.20	0.46	129	1.68	512
	10^7	0.63	0.24	0.87	264	0.25	0.14	0.40	128	1.27	392
	10^8	0.53	0.22	0.75	223	0.25	0.12	0.37	128	1.12	351
	10^9	0.49	0.21	0.70	206	0.25	0.11	0.36	128	1.07	334
2.0	10^5	1.49	0.44	1.92	624	0.26	0.33	0.59	130	2.51	754
	10^6	0.79	0.28	1.07	330	0.25	0.18	0.43	129	1.50	459
	10^7	0.59	0.23	0.82	246	0.25	0.13	0.39	129	1.21	375
	10^8	0.51	0.22	0.73	216	0.25	0.12	0.37	129	1.10	344
	10^9	0.48	0.21	0.69	203	0.25	0.11	0.36	129	1.06	332

Table 2: Per-client computation (in ms) and communication cost (in bytes) of our protocol for different values of ε, n, and $\delta = 10^{-11}$. We use exponential ElGamal as additively homomorphic encryption scheme. See Table 3 in the appendix for results using Paillier. The client cost, which is independent of ε and n, is 0.46ms CPU time and 192 bytes of communication.

requires $600KB$ of server communication per client). In contrast (see Table 2), for $\epsilon = 1$, our protocol requires roughly 1KB (for $n = 10^5$) and 0.5KB (for $n = 10^6$) of server communication per client. This is more than a 400× and a 1200× improvement, respectively, for these settings. For larger n the per client cost of our solution keeps decreasing, resulting in less than 0.4KB per client for $n \geq 10^7$. Moreover, Figure 12 (left) shows the improvements offered by our most optimal version. In particular, for $\varepsilon = 1, n = 10^6$, the optimized version incurs less than 1KB of total per-client communication, while the basic version requires over 13KB.

Figure 12 (right) shows how the total communication scales as n grows, for different values of ϵ. Note that the per-client cost approaches a constant as n grows. This is a nice feature of an $O(n)$ communication protocol, as opposed to the garbled circuits baseline, which has communication complexity $\Omega(n \log n)$ (ignoring the logarithmic dependence on $|D|$). More concretely, in our protocol, the overhead on top of sending the encrypted client data, i.e., the dummy generation in Step (2) of Figure 5, is $o(n)$. Thus the per-client communication across the servers approaches the (constant) client communication cost as n increases. The same observation applies to computation (Figure 12, middle). For large enough n the costs that the servers incur approaches the aggregated cost incurred by all clients. In particular, for a billion clients, with $\epsilon = 1$, our protocol requires only 1.07 ms of computation time 334 bytes of server communication, per client.

7 CONCLUSIONS AND OPEN QUESTIONS

In this paper we showed how to construct distributed two-server protocols to compute sparse histograms with DP. Similar to central DP mechanisms, our protocol achieves communication and computation efficiency that are independent of the domain size and only proportional to the number of client contributions and the histogram sparsity. Our protocols outperform the baseline that uses only garbled circuits, by revealing a DP view of the data to the two servers. It remains an open question if this is optimal, and if not, what the lower bound is in terms of communication and computation overhead for any given leakage and privacy parameters.

While our protocol protects against arbitrary collusions of clients with one of the two servers, it does not protect against malicious clients. Possible approaches to protect against these in future work include using ZKP-friendly hash functions, switching to a maliciously secure shared-key OPRF, and range proofs for client values.

We also pose a full formal treatment of MPC protocols with DP leakage as an open question. In particular, while our approach allows composing protocols in an MPC sense, we still proved DP of our end-to-end protocol by itself, without any composition into sub-protocols. A composition theorem for MPC protocols with DP leakage could simplify the development of future protocols with similar guarantees as ours.

ACKNOWLEDGMENTS

We thank Mingxun Zhou for informing us of an error in an earlier version of this paper.

REFERENCES

[1] Miklós Ajtai, János Komlós, and Endre Szemerédi. 1983. An $O(n \log n)$ Sorting Network. In *STOC*. 1–9.
[2] Francesco Aldà and Hans Ulrich Simon. 2018. A lower bound on the release of differentially private integer partitions. *IPL* 129 (2018), 1–4.
[3] Abdelrahaman Aly, Emmanuela Orsini, Dragos Rotaru, Nigel P. Smart, and Tim Wood. 2019. Zaphod: Efficiently Combining LSSS and Garbled Circuits in SCALE. In *WAHC*.
[4] Apple and Google. 2021. Exposure Notifications Private Analytics. https://github.com/google/exposure-notifications-android/blob/master/doc/ENPA.pdf.
[5] Victor Balcer and Albert Cheu. 2020. Separating Local & Shuffled Differential Privacy via Histograms. In *ITC*. 1:1–1:14.
[6] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. 2019. The Privacy Blanket of the Shuffle Model. In *CRYPTO*. 638–667.
[7] Kenneth E. Batcher. 1968. Sorting Networks and Their Applications. In *AFIPS Spring Joint Computing Conference*. 307–314.
[8] Amos Beimel, Kobbi Nissim, and Eran Omri. 2008. Distributed private data analysis: Simultaneously solving how and what. In *CRYPTO*. 451–468.
[9] James Bell, Adria Gascon, Badih Ghazi, Ravi Kumar, Pasin Manurangsi, Mariana Raykova, and Phillipp Schoppmann. 2022. Distributed, private, sparse histograms in the two-server model. In *CCS*.
[10] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In *CCS*.
[11] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard

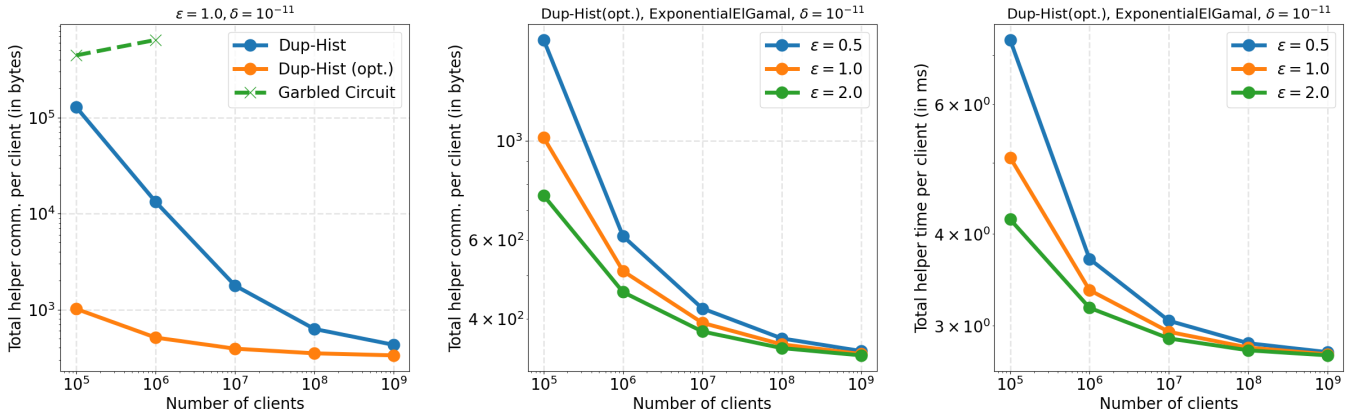

Figure 12: Per-client cost of our protocols. (Left) Total worst-case communication for the garbled circuits baseline, and the two variants of our protocol: the one with Negative Binomial noise for duplication (Dup-Hist), and the optimized variant with additional Poisson noise (Dup-Hist, opt.), for $\epsilon = 1/2$ and $\delta = 10^{-11}$. (Middle) Total worst-case server communication for our optimized protocol and several values of ϵ. (Right) Total server computation (offline+online), for several values of ϵ.

Seefeld. 2017. Prochlo: Strong Privacy for Analytics in the Crowd. In *SOSP*. 441–459.

[12] Jeremiah Blocki, Anupam Datta, and Joseph Bonneau. 2016. Differentially Private Password Frequency Lists. In *NDSS*.

[13] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. 2016. Privacy-preserving tax fraud detection in the cloud with realistic data volumes. Cybernetica Research, Report, https://cyber.ee/research/reports/T-4-24-Privacy-preserving-tax-fraud-detection-in-the-cloud-with-realistic-data-volumes.pdf.

[14] Jonas Böhler and Florian Kerschbaum. 2021. Secure Multi-party Computation of Differentially Private Heavy Hitters. In *CCS*. 2361–2377.

[15] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *CCS*.

[16] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2021. Lightweight Techniques for Private Heavy Hitters. In *SP*. 762–776.

[17] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function Secret Sharing: Improvements and Extensions. In *CCS*. 1292–1303.

[18] Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. 2022. MOTION–A Framework for Mixed-Protocol Multi-Party Computation. *ACM Trans. Priv. Sec.* 25, 2 (2022), 1–35.

[19] Mark Bun, Kobbi Nissim, and Uri Stemmer. 2019. Simultaneous Private Learning of Multiple Concepts. *JMLR* 20 (2019), 94:1–94:34.

[20] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *SP*.

[21] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2012. Optimal Lower Bound for Differentially Private Multi-party Aggregation. In *ESA*. 277–288.

[22] Panagiotis Chatzigiannis, Konstantinos Chalkias, and Valeria Nikolaenko. 2021. Homomorphic decryption in blockchains via compressed discrete-log lookup tables. In *DPM/CBT@ESORICS*. 328–339.

[23] Albert Cheu, Adam D. Smith, Jonathan R. Ullman, David Zeber, and Maxim Zhilyaev. 2019. Distributed Differential Privacy via Shuffling. In *EUROCRYPT*. 375–403.

[24] Albert Cheu and Maxim Zhilyaev. 2021. Differentially Private Histograms in the Shuffle Model from Fake Users. *CoRR* abs/2104.02739 (2021).

[25] Graham Cormode, Cecilia Procopiuc, Divesh Srivastava, and Thanh T. L. Tran. 2012. Differentially Private Summaries for Sparse Data. In *ICDT*.

[26] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *NSDI*.

[27] Henry Corrigan-Gibbs, Dan Boneh, Gary Chen, Steven Englehardt, Robert Helmer, Chris Hutten-Czapski, Anthony Miyaguchi, Eric Rescorla, and Peter Saint-Andre. 2020. Privacy-preserving Firefox telemetry with Prio. https://rwc.iacr.org/2020/slides/Gibbs.pdf.

[28] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. 1997. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *EUROCRYPT*, Vol. 1233. 103–118.

[29] Ivan Damgård and Mads Jurik. 2001. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *PKC*, Vol. 1992. 119–136.

[30] Damien Desfontaines, James Voss, Bryant Gipson, and Chinmoy Mandayam. 2022. Differentially private partition selection. *PoPETS* 2022, 1 (2022), 339–352.

[31] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*. 486–503.

[32] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*.

[33] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. 2019. Amplification by Shuffling: From Local to Central Differential Privacy via Anonymity. In *SODA*. 2468–2479.

[34] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *CCS*.

[35] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. 2003. Limiting privacy breaches in privacy preserving data mining. In *PODS*. 211–222.

[36] Vitaly Feldman, Audra McMillan, and Kunal Talwar. 2021. Hiding Among the Clones: A Simple and Nearly Optimal Analysis of Privacy Amplification by Shuffling. In *FOCS*. 954–964.

[37] Steven D. Galbraith, Ping Wang, and Fangguo Zhang. 2017. Computing elliptic curve discrete logarithms with improved baby-step giant-step algorithm. *Adv. Math. Commun.* 11, 3 (2017), 453–469.

[38] Taher El Gamal. 1984. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO*, Vol. 196. 10–18.

[39] Badih Ghazi, Noah Golowich, Ravi Kumar, Pasin Manurangsi, Rasmus Pagh, and Ameya Velingker. 2020. Pure Differentially Private Summation from Anonymous Messages. In *ITC*. 15:1–15:23.

[40] Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. 2021. On the Power of Multiple Anonymous Messages: Frequency Estimation and Selection in the Shuffle Model of Differential Privacy. In *EUROCRYPT*. 463–488.

[41] Badih Ghazi, Ben Kreuter, Ravi Kumar, Pasin Manurangsi, Jiayu Peng, Evgeny Skvortsov, Yao Wang, and Craig Wright. 2022. Multiparty Reach and Frequency Histogram: Private, Secure, and Practical. *PoPETS* 2022, 1 (2022), 373–395.

[42] Badih Ghazi, Ravi Kumar, and Pasin Manurangsi. 2021. User-Level Differentially Private Learning via Correlated Sampling. In *NeurIPS*.

[43] Badih Ghazi, Ravi Kumar, Pasin Manurangsi, and Rasmus Pagh. 2020. Private Counting from Anonymous Messages: Near-Optimal Accuracy with Vanishing Communication Overhead. In *ICML*. 3505–3514.

[44] Oded Goldreich. 2006. *Foundations of Cryptography: Volume 1.* Cambridge University Press, USA.

[45] Oded Goldreich. 2009. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press.

[46] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to play any mental game. In *STOC*. 218–229.

[47] S Goldwasser, S Micali, and C Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems. In *STOC*.

[48] Google. 2019. Private Join and Compute. https://github.com/google/private-join-and-compute/.

[49] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *USENIX*. 519–535.

[50] Adam Groce, Peter Rindal, and Mike Rosulek. 2019. Cheaper Private Set Intersection via Differentially Private Leakage. *PoPETS* 2019, 3 (2019), 6–25.
[51] Michael Hay, Chao Li, Gerome Miklau, and David D. Jensen. 2009. Accurate Estimation of the Degree Distribution of Private Networks. In *ICDM*. 169–178.
[52] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the Accuracy of Differentially Private Histograms Through Consistency. *VLDB* 3, 1 (2010), 1021–1032.
[53] Yan Huang, David Evans, and Jonathan Katz. 2012. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?. In *NDSS*.
[54] Stanislaw Jarecki and Xiaomin Liu. 2010. Fast Secure Computation of Set Intersection. In *SCN*.
[55] Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2008. What Can We Learn Privately?. In *FOCS*.
[56] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *CCS*.
[57] Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, and Alexandros Ntoulas. 2009. Releasing Search Queries and Clicks Privately. In *WWW*.
[58] Vasileios Lampos, Andrew C Miller, Steve Crossan, and Christian Stefansen. 2015. Advances in nowcasting influenza-like illness rates using search query logs. *Scientific Reports* 12760 (2015). Issue 5.
[59] Yehuda Lindell. 2017. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*. Springer International Publishing, 277–346.
[60] Yehuda Lindell. 2020. Secure Multiparty Computation. *CACM* 64, 1 (2020), 86–96.
[61] Pasin Manurangsi. 2022. Tight Bounds for Differentially Private Anonymized Histograms. In *SOSA*. 203–213.
[62] Sahar Mazloom and S. Dov Gordon. 2018. Secure Computation with Differentially Private Access Patterns. In *CCS*. 490–507.
[63] Frank McSherry and Ratul Mahajan. 2010. Differentially-Private Network Trace Analysis. *SIGCOMM Comput. Commun. Rev.* 40, 4 (aug 2010), 123–134.
[64] Catherine A. Meadows. 1986. A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party. In *SP*. 134–137.
[65] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. 2009. Computational Differential Privacy. In *CRYPTO*.
[66] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, Vol. 1592. 223–238.
[67] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2013. Understanding Hierarchical Methods for Differentially Private Histograms. *VLDB* 6, 14 (2013).
[68] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. 2019. Honeycrisp: Large-Scale Differentially Private Aggregation without a Trusted Core. In *SOSP*.
[69] Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C. Pierce. 2020. Orchard: Differentially Private Analytics at Scale. In *OSDI*.
[70] Phillipp Schoppmann, Lennart Vogelsang, Adrià Gascón, and Borja Balle. 2020. Secure and Scalable Document Similarity on Distributed Databases: Differential Privacy to the Rescue. *PoPETS* 2020, 2 (2020), 209–229.
[71] Daniel Shanks. 1971. Class number, a theory of factorization, and genera. In *Proc. of Symp. Math. Soc., 1971*, Vol. 20. 41–440.
[72] Ananda Theertha Suresh. 2019. Differentially Private Anonymized Histograms. In *NeurIPS*. 7969–7979.
[73] Sameer Wagh, Xi He, Ashwin Machanavajjhala, and Prateek Mittal. 2021. DP-cryptography: marrying differential privacy and cryptography in emerging applications. *CACM* 64, 2 (2021), 84–93.
[74] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. 2017. Locally Differentially Private Protocols for Frequency Estimation. In *USENIX*.
[75] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit.
[76] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-Scale Secure Multiparty Computation. In *CCS*.
[77] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *FOCS*. 162–167.
[78] Jun Zhang, Xiaokui Xiao, and Xing Xie. 2016. PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions. In *SIGMOD*. 155–170.
[79] Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. 2020. Federated Heavy Hitters Discovery with Differential Privacy. In *AISTATS*. 3837–3847.

A THE ELGAMAL CRYPTOSYSTEM

In Figure 13, we recall the operations for the ElGamal cryptosystem, including rerandomization of ciphertexts and partial decryption with a shared key.

Public parameters: group $\mathbb{G}$ of prime order q with generator g.

$\mathsf{Gen}_{\mathsf{ElGamal}}(1^\kappa)$:
Sample secret key $\mathsf{SK} \leftarrow_R \mathbb{Z}_q$. Set public key $\mathsf{PK} = g^{\mathsf{SK}} \in \mathbb{G}$. Output $(\mathsf{SK}, \mathsf{PK})$.

$\mathsf{Enc}_{\mathsf{ElGamal}}(\mathsf{PK}, m)$:
Choose $x \leftarrow_R \mathbb{G}$ and output ciphertext $(\mathsf{ct}_1, \mathsf{ct}_2) \leftarrow (g^x, \mathsf{PK}^x \cdot m)$.

$\mathsf{Dec}_{\mathsf{ElGamal}}(\mathsf{SK}, (\mathsf{ct}_1, \mathsf{ct}_2))$:
Output $m = \mathsf{ct}_2 / \mathsf{ct}_1^{\mathsf{SK}}$.

$\mathsf{PartialDec}_{\mathsf{ElGamal}}(\mathsf{SK}, (\mathsf{ct}_1, \mathsf{ct}_2))$:
Let $m = \mathsf{ct}_2 / \mathsf{ct}_1^{\mathsf{SK}}$. Output (m, ct_2).

$\mathsf{Randomize}_{\mathsf{ElGamal}}(\mathsf{ct}_1, \mathsf{ct}_2)$:
Output $(\mathsf{ct}_1', \mathsf{ct}_2') \leftarrow (g^{x'} \cdot \mathsf{ct}_1, \mathsf{PK}^{x'} \cdot \mathsf{ct}_2)$ where $x' \leftarrow_R \mathbb{Z}_q$.

Figure 13: The ElGamal cryptosystem.

B MISSING PROOFS FROM SECTION 5

Lemma 16 (Post-Processing of DP). *Let $\mathcal{U}_1, \mathcal{U}_2$ be distributions. Let f be a possibly randomized function that takes in values in $\mathrm{supp}(\mathcal{U}_1) \cup \mathrm{supp}(\mathcal{U}_2)$. Then, $d_\varepsilon(f(\mathcal{U}_1) \| f(\mathcal{U}_2)) \le d_\varepsilon(\mathcal{U}_1 \| \mathcal{U}_2)$.*

Lemma 17 (DP Basic Composition). *Let $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4$ be distributions. Then, we have $d_\varepsilon(\mathcal{U}_1 \otimes \mathcal{U}_2 \| \mathcal{U}_3 \otimes \mathcal{U}_4) \le d_{\varepsilon/2}(\mathcal{U}_1 \| \mathcal{U}_3) + d_{\varepsilon/2}(\mathcal{U}_2 \| \mathcal{U}_4)$.*

B.1 Proof of Theorem 6

Proof. We construct simulators $\mathsf{Sim}_i^{\mathsf{threshold}}, i \in \{1, 2\}$ that, given the inputs, leakage, and outputs to party i generate a view that is indistinguishable from the view of party i in protocol $\Pi_{\mathsf{threshold}}$.

For P1, $\mathsf{Sim}_1^{\mathsf{threshold}}$ performs the following:

(1) For each $i \in [n]$, computes $\mathsf{Enc}\big(\mathsf{PK}_1, \mathsf{val}_i + \xi_i^{(2)}\big)$, where the encrypted value is taken from the second component of $\mathcal{L}_{\mathsf{threshold}}^{\mathsf{P1}}$.
(2) Sets $\mathcal{S}^{(2)}$ to the set of ciphertexts generated in the previous step.
(3) Outputs $\xi_i^{(1)}$ as the value sampled from $\mathsf{DLap}(\lambda_1)$ in Step (2a) of Figure 3, and $\mathcal{S}^{(2)}$ as the message received from P2 in Step (1b).

This simulated view together with the ideal output is indistinguishable from the real view together with the real output as the AHE scheme provides function secrecy, and the employed noise distributions $\mathsf{TDLap}(\lambda, t)$ in Figure 2 and Figure 3 are the same.

P2 has no output in the protocol.

□

B.2 Proof of Theorem 8

Proof. First, observe that the output of Π_{hist} follows the same distribution as $\mathcal{F}_{\mathsf{hist}}$ from Figure 10 (or, equivalently, Figure 4). This follows from the fact that the dummies added by P1 all have values of 0, the correctness of the AHE scheme, and the fact that the noise samples ξ follow the same distribution in Figure 10 and Figure 3. We describe a simulator $\mathsf{Sim}_1^{\mathsf{hist}}$, which takes as inputs $n, \mathcal{L}_{\mathsf{hist}}^{\mathsf{P1}} = \big(\mathcal{N}^1, \mathcal{N}^3, \xi^{(1)}, \mathcal{V}^\perp\big)$, and $(\mathsf{ind}_i', \mathsf{val}_i')_{i \in [n']}$ from the output of $\mathcal{F}_{\mathsf{hist}}$, and simulates the view $\mathsf{View}_{\mathsf{Sim}_1^{\mathsf{hist}}}$ in the protocol as follows:

(1) $\mathsf{Sim}_1^{\mathsf{hist}}$ generates n random client contributions of the form $\big(\mathsf{Enc}_{\mathsf{ElGamal}}(\mathsf{PK}', \mathsf{H}(u_i)), \mathsf{Enc}_{\mathsf{ElGamal}}(\mathsf{PK}, u_i), \mathsf{Enc}(\mathsf{PK}'', w_i)\big)$, where $u_i \leftarrow_R \mathcal{U}$, $v_i \leftarrow_R \mathcal{V}$, $w_i \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{PK}_{\mathsf{AHE}}, v_i)$.
(2) To produce the noise samples generated in Steps (2) and (4) of Figure 6, $\mathsf{Sim}_1^{\mathsf{hist}}$ uses $\mathcal{N}^1, \mathcal{N}^3$ from $\mathcal{L}_{\mathsf{hist}}^{\mathsf{P1}}$. For Step (3), $\mathsf{Sim}_1^{\mathsf{hist}}$ samples $[n]$ times independently from $\mathsf{NBin}(r, p)$.
(3) $\mathsf{Sim}_1^{\mathsf{hist}}$ recomputes $\mathcal{L}_{\mathsf{threshold}}^{\mathsf{P1}}$ from $\mathcal{V}^{\perp}$ and the second components of $\mathcal{F}_{\mathsf{hist}}^{\mathsf{P1}}$, subtracting $\xi^{(1)}$.
(4) $\mathsf{Sim}_1^{\mathsf{hist}}$ then invokes the simulator $\mathsf{Sim}_1^{\mathsf{threshold}}$ with inputs $\mathsf{SK}_{\mathsf{AHE}}$, $\mathcal{F}_{\mathsf{threshold}}^{\mathsf{P1}}$ and $\mathcal{L}_{\mathsf{threshold}}^{\mathsf{P1}}$ to generate the part of the view coming from the thresholding protocol $\Pi_{\mathsf{threshold}}$.
(5) Finally, to simulate the message received in Step (5), P1 computes $\big(\mathsf{Enc}_{\mathsf{ElGamal}}(\mathsf{PK}_1, \mathsf{ind}_i'')\big)_{i\in[n'']}$.

The indistinguishability of the simulated view $\mathsf{View}_{\mathsf{Sim}_1^{\mathsf{hist}}}$ that consists of the messages generated above and the real execution view $\mathsf{View}_{\Pi_{\mathsf{hist}}^{\mathsf{P1}}}$ of P1 follows from a standard hybrid argument. We can replace the random ciphertexts in Step (1) above by the real client encryptions due to the semantic security of the encryption schemes used. For Step (2), observe that the noise samples in $\mathcal{L}_{\mathsf{hist}}^{\mathsf{P1}}$ follow the same distribution as in SampleDummies. For Step (3), we rely on the fact that the view produced by $\mathsf{Sim}_1^{\mathsf{threshold}}$ is indistinguishable of the real view $\mathsf{View}_{\Pi_{\mathsf{threshold}}^{\mathsf{P1}}}$ of the thresholding subprotocol (Theorem 6). Finally, the encryptions in Step (4) encrypt the same values as in the real protocol, since all indices drawn from $\mathcal{I}$ are guaranteed to be below the threshold, as $\tau > 2t$ and all $\xi_i, \xi_i' \le t$ in Figure 3. Indistinguishability of the distributions follows from the fact that ciphertexts are rerandomized by P2 in the real protocol.

Next we describe a simulator $\mathsf{Sim}_2^{\mathsf{hist}}$ for the view $\mathsf{View}_{\Pi_{\mathsf{hist}}^{\mathsf{P2}}}$ of P2 . It takes as inputs $\mathcal{L}_{\mathsf{hist}}^{\mathsf{P2}} = \big(\mathcal{H}^3, \mathcal{N}^4, \xi^{(2)}\big)$, and simulates the $\mathsf{View}_{\mathsf{Sim}_2^{\mathsf{hist}}}$ as follows:

(1) $\mathsf{Sim}_2^{\mathsf{hist}}$ samples random $\mathsf{K} \leftarrow_R \mathbb{Z}_q$.
(2) Let $i \in \mathcal{H}^3$ denote that $\mathcal{H}_i^3 > 0$. For each $i \in \mathcal{H}^3$, $\mathsf{Sim}_2^{\mathsf{hist}}$ generates $\mathcal{H}_i^3$ values $u_0^i, \ldots, u_{\mathcal{H}_i^3}^i \leftarrow_R \mathcal{U}$, and then for each $j \in [\mathcal{H}_i^3]$ computes $\big(\mathsf{Enc}_{\mathsf{ElGamal}}(\mathsf{PK}', (\mathsf{H}(u_j^i))^{\mathsf{K}}), \mathsf{Enc}_{\mathsf{ElGamal}}(\mathsf{PK}, u_j^i), \mathsf{Enc}(\mathsf{PK}'', w_j^i)\big)$ i times, where $v_j^i \leftarrow_R \mathcal{V}$, $w_i^i \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{PK}_{\mathsf{AHE}}, v_j^i)$, using fresh randomness for each encryption.
(3) To simulate the noise samples generated in SampleBuckets, $\mathsf{Sim}_2^{\mathsf{hist}}$ uses $\mathcal{N}^4$ directly.
(4) Let $n'' = \sum_{i\in\mathcal{H}^3} \mathcal{H}_i^3 + \sum_{(j,M_j)\in\mathcal{N}^4} M_j$. For each $i \in [n'']$, $\mathsf{Sim}_2^{\mathsf{hist}}$ generates values $v_i' \leftarrow_R \mathcal{V}$, and computes $w_i' \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{PK}_{\mathsf{AHE}}, v_i')$. Then, $\mathsf{Sim}_2^{\mathsf{hist}}$ invokes $\mathsf{Sim}_2^{\mathsf{threshold}}$ with inputs $(w_i')_{i\in[n'']}$, $\mathcal{F}_{\mathsf{threshold}}^{\mathsf{P2}} = \perp$, and $\mathcal{L}_{\mathsf{threshold}}^{\mathsf{P2}} = \xi^{(2)}$ to simulate the view from the thresholding subprotocol.

The indistinguishability of the simulated view from the real view follows again from a hybrid argument. For the message generated in Step (2), observe that $(\mathsf{H}(\mathsf{ind}_i))^{\mathsf{K}}$ is indistinguishable from random when H is modeled as a random oracle, and hence the first component is indistinguishable from the real view. For the second and third component, indistinguishability follows from the semantic security of the encryption schemes used. For Step (3), the distribution of $\mathcal{N}^4$ is the same as the one in SampleBuckets. Finally, for Step (4), observe that n'' follows the same distribution as in Step (3c) of Π_{hist}. Thus, we call the $\mathsf{Sim}_2^{\mathsf{threshold}}$ with an input of the correct size, and the resulting view is indistinguishable by Theorem 6. □

B.3 Proof of Lemma 9

Proof. Firstly, $\mathcal{F}_{\mathsf{hist}}^{\mathsf{P2}}$ is a constant and $\mathcal{N}^1$ and $\mathcal{N}^3$ are independent of the input. They are also independent of $\mathcal{F}_{\mathsf{hist}}^{\mathsf{P1}}$ and the rest of $\mathcal{L}_{\mathsf{hist}}^{\mathsf{P1}}$ so can just be ignored.

P1 can subtract $\xi_j^{(1)}$ from each val_j. This is a reversible operation after which $\xi_j^{(1)}$ is independent of the inputs and the values, so $\xi_j^{(1)}$ can then be ignored. To summarize, the only leakage left is $\widetilde{\mathcal{V}}^{\perp} := \Big\{\mathsf{val}_j + \xi_j^{(2)} \mid j \in [|\mathcal{I}'|], \mathcal{I}_j' = (\mathsf{ind}_j, \mathsf{val}_j), \mathsf{val}_j + \xi_j^{(2)} < \tau\Big\}$.

Now consider two neighboring databases. If the user who has different inputs (assumed wlog to be the first user) has in both cases an index that is also held by someone else, then their change in the values of those indices is $(\varepsilon_{\mathsf{counts}}, \delta_{\mathsf{counts}})$-DP by the Laplace mechanism (using the randomness of $\xi^{(2)}$).

If in both cases they have a unique index, then the corresponding distributions of $\widetilde{\mathcal{V}}^{\perp}$ are exactly the same. Furthermore, neither indices will be in $\mathcal{F}_{\mathsf{hist}}^{\mathsf{P1}}$. Therefore, we achieve $(0, 0)$-DP in this case.

Finally, if the first user has a unique index in only one of the databases (wlog D'), then we get DP by the following composition. Let this has value $\mathsf{val} \in [\Delta]$; note that $M_{\mathsf{val}} \sim \mathsf{TSDLap}(\lambda_2, t_2)$ additional indices with the same value were added in Step (6a). This provides $(\varepsilon_{\mathsf{leakage}}, \delta_{\mathsf{leakage}})$-DP for whether the index is present. The remaining difference is amongst the common index, whose value changes by at most Δ. Thus, the noise $\xi^{(2)}$ provides $(\varepsilon_{\mathsf{counts}}/2, \delta_{\mathsf{counts}}/2)$-DP for this value due to the Laplace mechanism. The result follows immediately by composition. □

B.4 Proof of Lemma 10

Proof. Firstly, $\mathcal{F}_{\mathsf{hist}}^{\mathsf{P2}}$ is constant and $\mathcal{N}^4$ is independent of the input and other components. It therefore suffices to consider $(\mathcal{H}^3, \xi^{(2)}, \mathcal{F}_{\mathsf{hist}}^{\mathsf{P1}})$. We have that $(\xi^{(2)}, \mathcal{F}_{\mathsf{hist}}^{\mathsf{P1}})$ is $(\varepsilon_{\mathsf{counts}}, \delta_{\mathsf{counts}})$-DP by exactly the same argument in the proof of Theorem 7. Meanwhile by Theorem 15 (which we prove next in Section 5.2.2) we know $\mathcal{H}^3$ is $(\varepsilon_{\mathsf{leakage}}, \delta_{\mathsf{leakage}})$-DP and it is also independent of $(\xi^{(2)}, \mathcal{F}_{\mathsf{hist}}^{\mathsf{P1}})$, thus by basic composition of DP we are done. □

B.5 Proof of Lemma 11

Proof. Notice that $\mathcal{H}^3$ (resp. $\mathcal{H}'^3$) is a post-processing of $\mathcal{H}^1$ (resp. $\mathcal{H}'^1$). Therefore, we may apply Lemma 16 to conclude that

$$d_\varepsilon(\mathcal{H}^3 \| \mathcal{H}'^3) \le d_\varepsilon(\mathcal{H}^1 \| \mathcal{H}'^1).$$

Recall that $\mathcal{N}^1 = (N_i)_i$ denote the noise added to the histogram in the first step. We can further rearrange the RHS above as

$$\begin{aligned} &d_\varepsilon(\mathcal{H}^1 \| \mathcal{H}'^1) \\ &= d_\varepsilon(\mathcal{H}^0 + \mathcal{N} \| \mathcal{H}'^0 + \mathcal{N}) \\ &= d_\varepsilon(\mathcal{H}^0 - \mathcal{H}^0 + \mathcal{N}^1 \| \mathcal{N}^1) \end{aligned}$$

$$
\begin{aligned}
&= d_\varepsilon(\mathbf{1}_{m_1} - \mathbf{1}_{m_1-1} + \mathcal{N}^1 \| \mathcal{N}^1) \\
&\overset{\text{Lemma 17}}{=} d_{\varepsilon/2}(1 + N_{m_1} \| N_{m_1}) + d_{\varepsilon/2}(-1 + N_{m_1-1} \| N_{m_1-1}) \\
&\le \delta,
\end{aligned}
$$

where the last inequality follows from the standard analysis of the truncated discrete Laplace mechanism (see e.g., [30]) and our setting of parameters.

The argument above shows that $d_\varepsilon(\mathcal{H}^3 \| \mathcal{H}'^3) \le \varepsilon$. An analogous argument also implies $d_\varepsilon(\mathcal{H}'^3 \| \mathcal{H}^3) \le \varepsilon$. As a result, $\mathcal{H}^3$ and $\mathcal{H}'^3$ are (ϵ, δ)-indistinguishable. □

B.6 Proof of Lemma 12

Lemma 18 (Generic Privacy for High Counts). *Let $\mathcal{U} := \mathsf{NBin}(r, p)$. The $\mathcal{H}^3$ and $\mathcal{H}'^3$ are (ϵ, δ)-indistinguishable if $m_1 > T'$ and the following condition holds:*

$$
d_\varepsilon(\mathcal{U}^{\star(T'+1)} + 1 \| \mathcal{U}^{\star T'}) \le \delta, \tag{6}
$$

and

$$
d_\varepsilon(\mathcal{U}^{\star T'} \| \mathcal{U}^{\star(T'+1)} + 1) \le \delta. \tag{7}
$$

Proof. Suppose that the two conditions hold. Consider $d_\varepsilon(\mathcal{H}^3 \| \mathcal{H}'^3)$. Since $\mathcal{H}^3$ (resp. $\mathcal{H}'^3$) is a post-processing of $\mathcal{H}^2$ (resp. $\mathcal{H}'^2$), we may apply Lemma 16 to derive

$$
d_\varepsilon(\mathcal{H}^3 \| \mathcal{H}'^3) \le d_\varepsilon(\mathcal{H}^2 \| \mathcal{H}'^2).
$$

Let $\mathcal{D}_{\mathsf{common}}$ denote the database resulting from removing *all occurences* of ind_1. Then, let $\mathcal{H}_{\mathsf{common}}$ denote the (randomized) output if we were to run the algorithm on ind_1. Notice that we may generate $\mathcal{H}^2$ by letting it be $\mathcal{H}_{\mathsf{common}} + \mathbf{1}_X$ where $X \sim \mathsf{NBin}(rm_1, p)$; similarly, $\mathcal{H}'^2$ has the same distribution as $\mathcal{H}_{\mathsf{common}} + \mathbf{1}_{X'}$ where $X' \sim \mathsf{NBin}(r(m_1 - 1), p)$. Therefore, we have

$$
\begin{aligned}
d_\varepsilon(\mathcal{H}^2 \| \mathcal{H}'^2) &= d_\varepsilon(\mathcal{H}_{\mathsf{common}} + \mathbf{1}_X \| \mathcal{H}_{\mathsf{common}} + \mathbf{1}_{X'}) \\
&\le d_\varepsilon(\mathbf{1}_X \| \mathbf{1}_{X'}) \\
&= d_\varepsilon(X \| X'),
\end{aligned}
$$

where the first inequality again follows from Lemma 16.

Observe again that if we let $Y \sim \mathsf{NBin}(r(T'+1), p)$, $Y' \sim \mathsf{NBin}(rT', p)$ and $Z \sim \mathsf{NBin}(m - T' - 1, p)$, then we may write $X = Y + Z$ and $X = Y' + Z$. Therefore, we may again apply Lemma 16 to arrive at

$$
d_\varepsilon(X \| X') \le d_\varepsilon(Y \| Y') = d_\varepsilon(\mathcal{U}^{\star(T'+1)} + 1 \| \mathcal{U}^{\star T'}) \le \delta,
$$

where the second inequality follows from our first assumption.

Combining the previous three inequalities yields $d_\varepsilon(\mathcal{H}^3 \| \mathcal{H}'^3) \le \delta$. Via a similar derivation, we can also conclude that $d_\varepsilon(\mathcal{H}^3 \| \mathcal{H}'^3) \le d_\varepsilon(\mathcal{U}^{\star T'} \| \mathcal{U}^{\star(T'+1)} + 1) \le \delta$. Therefore, we can conclude that $\mathcal{H}^3$ and $\mathcal{H}'^3$ are (ε, δ)-indistinguishable as desired. □

B.6.1 Useful Lemmas. The following is a well-known but very useful fact about hockey stick divergence.

Lemma 19. *Let $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3$ be three distributions. Then,*

$$
d_\varepsilon(\mathcal{U}_1 \| \mathcal{U}_2) \le d_{\varepsilon/2}(\mathcal{U}_1 \| \mathcal{U}_3) + e^{\varepsilon/2} \cdot d_{\varepsilon/2}(\mathcal{U}_3 \| \mathcal{U}_2)
$$

Proof of Lemma 19. We have

$$
\begin{aligned}
d_\varepsilon(\mathcal{U}_1 \| \mathcal{U}_2) &= \sum_{x \in \mathrm{supp}(\mathcal{U}_1)} [p_{\mathcal{U}_1}(x) - e^\varepsilon \cdot p_{\mathcal{U}_2}(x)]_+ \\
&= \sum_{x \in \mathrm{supp}(\mathcal{U}_1)} \big[p_{\mathcal{U}_1}(x) - e^{\varepsilon/2} \cdot p_{\mathcal{U}_3}(x) \\
&\qquad + e^{\varepsilon/2} \cdot p_{\mathcal{U}_3}(x) - e^\varepsilon \cdot p_{\mathcal{U}_2}(x)\big]_+ \\
&\le \sum_{x \in \mathrm{supp}(\mathcal{U}_1)} [p_{\mathcal{U}_1}(x) - e^{\varepsilon/2} \cdot p_{\mathcal{U}_3}(x)]_+ \\
&\quad \sum_{x \in \mathrm{supp}(\mathcal{U}_1)} [e^{\varepsilon/2} \cdot p_{\mathcal{U}_3}(x) - e^\varepsilon \cdot p_{\mathcal{U}_2}(x)]_+ \\
&\le d_{\varepsilon/2}(\mathcal{U}_1 \| \mathcal{U}_3) + e^{\varepsilon/2} \cdot d_{\varepsilon/2}(\mathcal{U}_3 \| \mathcal{U}_2).
\end{aligned}
$$

□

To instantitate specific parameters for our algorithms, we will also use the following result, which is a special case of $\Delta = 1$ of [43, Theorem 13].

Theorem 20. *Let $\varepsilon, \delta \in (0, 1)$, $p = e^{-0.2\varepsilon}$, $r \ge 3\,(1 + \log(1/\delta))$ and $\mathcal{U} = \mathsf{NBin}(r, p)$. Then, we have*

$$
d_\varepsilon(\mathcal{U} + 1 \| \mathcal{U}), d_\varepsilon(\mathcal{U} \| \mathcal{U} + 1) \le \delta.
$$

B.6.2 Proof of Lemma 12.

Proof. For brevity, we write $R = rT'$ and $\tilde{R} = r(T' + 1)$. We will show the following two inequalities:

$$
d_{\varepsilon/2}(\mathsf{NBin}(\tilde{R}, p) \| \mathsf{NBin}(R, p)) \le \delta/4, \tag{8}
$$

and

$$
d_{\varepsilon/2}(\mathsf{NBin}(R, p) \| \mathsf{NBin}(\tilde{R}, p)) = 0. \tag{9}
$$

Before we prove (8) and (9), let us note how they imply the (ε, δ)-indistinguishability. Using (8), we may bound $d_\varepsilon(\mathcal{U}^{\star(T'+1)} + 1 \| \mathcal{U}^{\star T'})$ as follows:

$$
\begin{aligned}
&d_\varepsilon(\mathcal{U}^{\star(T'+1)} + 1 \| \mathcal{U}^{\star T'}) \\
&= d_\varepsilon(\mathsf{NBin}(\tilde{R}, p) + 1 \| \mathsf{NBin}(R, p)) \\
&\overset{\text{Lemma 19}}{\le} d_{\varepsilon/2}(\mathsf{NBin}(\tilde{R}, p) + 1 \| \mathsf{NBin}(\tilde{R}, p)) \\
&\qquad + e^{\varepsilon/2} \cdot d_{\varepsilon/2}(\mathsf{NBin}(\tilde{R}, p) \| \mathsf{NBin}(R, p)) \\
&\overset{\text{Theorem 20}}{\le} \delta/2 + e^{\varepsilon/2} \cdot d_{\varepsilon/2}(\mathsf{NBin}(\tilde{R}, p) \| \mathsf{NBin}(R, p)) \\
&\overset{(8)}{\le} \delta.
\end{aligned}
$$

Similarly, we can also use (9) to derive $d_\varepsilon(\mathcal{U}^{\star T'} \| \mathcal{U}^{\star(T'+1)} + 1) \le \delta$. Therefore, we may apply Lemma 18 to conclude that $\mathcal{H}^3$ and $\mathcal{H}'^3$ are (ε, δ)-indistinguishabile.

We will next prove (8) and (9), starting with the former. For any $x \in \mathbb{N} \cup \{0\}$, we have

$$
\begin{aligned}
\frac{p_{\mathsf{NBin}(R,p)}(x)}{p_{\mathsf{NBin}(\tilde{R},p)}(x)} &= \frac{\binom{x+R-1}{x}(1-p)^R p^x}{\binom{x+\tilde{R}-1}{x}(1-p)^{\tilde{R}} p^x} \\
&\le \frac{1}{(1-p)^r} \\
&\le \frac{1}{(0.2\varepsilon)^r} \\
&\le e^{\varepsilon/2},
\end{aligned}
$$

where the last inequality follows from our choice of T', which implies that $r \le \frac{4}{\varepsilon}\,(1 + \log(1/\varepsilon))$. Therefore, we simply have $d_{\varepsilon/2}(\mathsf{NBin}(R, p) \| \mathsf{NBin}(\tilde{R}, p)) = 0$, completing the proof of (9).

Finally, we will prove (8). In this case, we have

$$\begin{aligned}\frac{p_{\mathsf{NBin}(\tilde{R},p)}(x)}{p_{\mathsf{NBin}(R,p)}(x)} &= \frac{\binom{x+\tilde{R}-1}{x}(1-p)^{\tilde{R}}p^x}{\binom{x+R-1}{x}(1-p)^R p^x}\\ &= \frac{\binom{x+\tilde{R}-1}{x}}{\binom{x+R-1}{x}}\\ &\le \left(\tilde{R}/R\right)^x\\ &= \left(1+1/T'\right)^x\\ &\le e^{x/T'},\end{aligned}$$

which is less than $e^{\varepsilon/2}$ for all $x \le T'\varepsilon/2$. Therefore, we have

$$d_{\varepsilon/2}(\mathsf{NBin}(\tilde{R},p)\|\mathsf{NBin}(R,p)) \le \Pr_{X\sim\mathsf{NBin}(\tilde{R},p)}[X > T'\varepsilon/2]. \quad (10)$$

It is well known that $\mathbb{E}[e^{tX}] = \left(\frac{1-p}{1-pe^t}\right)^{\tilde{R}}$ for all $t < -\ln p$. Setting $t = 0.1\varepsilon$, we have

$$\begin{aligned}\Pr[X > T'\varepsilon/2] &\le \mathbb{E}[e^{tX}]/e^{tT'\varepsilon/2}\\ &= \left(\frac{1-p}{1-pe^t}\cdot e^{-tT'\varepsilon/(2\tilde{R})}\right)^{\tilde{R}}\\ &= \left((1+e^{0.1\varepsilon})\cdot e^{-0.05T'\varepsilon^2/\tilde{R}}\right)^{\tilde{R}},\\ &\le \left(3\cdot e^{-0.05T'\varepsilon^2/\tilde{R}}\right)^{\tilde{R}}\end{aligned}$$

where the second equality follows from our choice of t.

Since $T' \ge 1$, we have $\tilde{R} \le 2R = 2rT'$. Plugging this into the above, we have

$$\Pr[X > T'\varepsilon/2] \le \left(3\cdot e^{-0.025\varepsilon^2/r}\right)^{\tilde{R}}.$$

Recall from our setting of parameters that $1/r \ge 100/\varepsilon^2$ and therefore, $e^{-0.025\varepsilon^2/r} \le 1/10$. Thus, we have

$$\Pr[X > T'\varepsilon/2] \le \exp(-\tilde{R}) \le \exp(-R) \le \delta/4.$$

The above inequality together with (10) yields (8), which concludes our proof. □

B.7 Proof of Theorem 13

Proof. The privacy statement is a straightforward combination of Lemmas 11 and 12.

To analyze the number of dummies added we consider each step of Figure 6 in turn.

In the first step the number of dummies added is $\sum_i iN_i$ (note that adding one to entry i of the anonymized histogram requires i dummy messages). This has expected value $T(T+1)t/2$.

In the second step the number of dummies added is given by the number of messages so far multiplied by the number of times each expects to be duplicated, i.e., $(n + T(T+1)t/2)rp/(1-p)$.

In the third step no dummies are added as the η_j are all zero.

If we let $R = \log(1/\delta)/\varepsilon$ then $t = \Theta(R)$ and $rp/(1-p) = \Theta(R/T)$ so the total number of dummies added is $\Theta(T^2R + nR/T + TR^2)$. The first two terms together are minimised by taking $T = \Theta(n^{1/3})$ and given the assumption that $R = o(n^{1/3})$ the final term is negligible. Thus the optimal number of generated dummies is $\Theta(n^{2/3}R)$. □

B.8 Proof of Lemma 14

Proof. Denote $\mu_{m'_1}$ and $q_{m'_1}$ by μ and q respectively. Let $A, B, C \sim \mathsf{Poi}(\mu_{m'_1})$ and $\Gamma \sim \mathsf{Ber}(q_{m'_1})$. We will provide a function f such that both $\mathcal{H}^3 \stackrel{d}{=} f(A, B+\Gamma, C+1-\Gamma)$ and $\mathcal{H}'^3 \stackrel{d}{=} f(A+\Gamma, B, C+1-\Gamma)$. By Lemma 16 we will then have

$$d_\varepsilon(\mathcal{H}^3\|\mathcal{H}'^3) \le d_\varepsilon((A, B+\Gamma, C+1-\Gamma)\|(A+\Gamma, B, C+1-\Gamma)). \quad (11)$$

To bound this divergence note that

$$\mathbb{P}((A+\Gamma, B, C+1-\Gamma) = (a,b,c)) = \frac{\exp(-3\mu)\mu^{a+b+c-1}}{a!b!c!}(qa+(1-q)c), \quad (12)$$

and similarly

$$\mathbb{P}((A, B+\Gamma, C+1-\Gamma) = (a,b,c)) = \frac{\exp(-3\mu)\mu^{a+b+c-1}}{a!b!c!}(qb+(1-q)c), \quad (13)$$

giving a likelihood ratio of

$$\frac{qa+(1-q)c}{qb+(1-q)c}. \quad (14)$$

In either case this is bounded by

$$\frac{qA+(1-q)C+1}{qB+(1-q)C}, \quad (15)$$

which by the definition of μ is $> \exp\epsilon$ with probability $\le \delta$. Thus we have (ϵ, δ)-indistinguishability.

It remains to exhibit such a function f. To describe f, let us define $\mathcal{D}_{\mathsf{common}}$ as the histogram resulting from removing all occurences of ind_1 from $\mathcal{D}$ and let $\mathcal{H}^0_{\mathsf{common}}$ denote its anonymized histogram[5]. To compute $f(a,b,c)$ start from $\mathcal{H}^0_{\mathsf{common}}$. Then apply all but the SampleBlanketDummies step of Figure 6. Note that so far the computation is independent of (a,b,c).

Next we have the main step, in this step we sample from the distribution $\alpha_{m'_1}$, a times, and for each result j increment the count of multiplicity j by one. Then repeat this process with distribution $\beta_{m'_1}$, b times and $\gamma_{m'_1}$, c times.

Finally we add Poisson noise with mean $\eta_j - \mu(\alpha_{m'_1,j} + \beta_{m'_1,j} + \gamma_{m'_1,j})$ to multiplicity j. The value of $f(a,b,c)$ is given by the resulting histogram.

If (a,b,c) were given by (A,B,C) then the main step is equivalent to adding Poisson noise with mean $\mu(\alpha_{m'_1,j} + \beta_{m'_1,j} + \gamma_{m'_1,j})$ to multiplicity j. Combining this with the noise added in the final step has the exact same total effect as step SampleBlanketDummies in Figure 6. In summary, $f(a,b,c) \stackrel{d}{=} \mathcal{H}^3_{\mathsf{common}}$, where $\mathcal{H}^3_{\mathsf{common}}$ is the result of the entire procedure in Figure 10 but starting with $\mathcal{D}_{\mathsf{common}}$.

If the input is instead given by $(a,b,c) = (A+\Gamma, B, C+1-\Gamma)$ then the output is the same except with one extra value added to a multiplicity chosen according to the distribution $q\alpha_{m'_1} + (1-q)\gamma_{m'_1}$ i.e. the distribution of the multiplicity the extra value of multiplicity m'_1 after duplication. The output then exactly matches the distribution of $\mathcal{H}'^3$, as required. The argument is identical for the $\mathcal{H}^3$ case. □

[5] In other words, $\mathcal{H}^0_{\mathsf{common}}$ is the result of reducing the m_1 entry of $\mathcal{H}^0$ by one or, equivalently, the result of reducing the $m_1 - 1$ entry of $\mathcal{H}'^0$ by one.

C HEURISTIC ANALYSIS OF ASYMPTOTIC COMMUNICATION REQUIREMENTS

To analyze the number of dummies sent in step (2) we consider each step of Figure 6 separately. As in the proof of Theorem 13 we let R denote $\log(1/\delta)/\varepsilon$ and find similarly that in the first two steps we expect to add $\Theta(R(T^2 + (n + T^2R)/T'))$ dummies. The optimal number of dummies here can be at most $n^{2/3}R$ as that bound was achieved with the extra restrictions of Theorem 13. Therefore it must be optimal to take $T \leq O(n^{1/3})$ and combining this with the assumption that $R = o(n^{1/3})$ the T^2R/T' term must then be negligible. Thus we can simplify this expression to $\Theta(RT^2 + Rn/T')$.

In the third step we expect to add $\sum_j j\eta_j$ dummies. To evaluate this we must determine reasonable bounds on the values of η_j.

Let us start by considering the value or μ_i as a function of q_i. Suppose for a given μ we could prove that A, B, C would with probability $1 - \delta$ all are between l and u. Then inequality (3) would be satisfied so long as $(u - l)/l \leq (\exp(\varepsilon) - 1)/q_i$ which implies it suffices that $l \geq u/(1 + \varepsilon/q_i)$ or that $l \geq q_i u/\varepsilon$. We thus wish to show this happens for not too large a value of μ.

If we set $u = 2.5\mu$ then using an upper tail bound for the Poisson distribution $\mathbb{P}(A > u) \leq \exp(-(u-\mu)^2/2(\mu + (u-\mu)/3)$ which is in turn at most $\exp(-3\mu/4)$. We can then take $l = 2.5q_i\mu/\varepsilon$, a lower tail bound for the Poisson distribution now tells us that $\mathbb{P}(A < l) \leq \exp(-(\mu - l)^2/2\mu$ which here is at most $\exp(-\mu(1 - 2.5q_i/\varepsilon)^2/2)$. If $q_i < \varepsilon/5$ this is at most $\exp(-\mu/8)$. Thus so long as $q_i < \varepsilon/5$ we have $\mu_i \leq 8\log(6/\delta)$.

Let $h_i = \max_j \tau_{i,j}$ and $h'_i = \max_j |\tau_{i,j} - \tau_{i+1,j}|$ The expression $\alpha_{i,j} + \beta_{i,j} + \gamma_{i,j} \leq h_i/(1 - q_i) + h'_i/q_i$. Thus so long as $q_i < \varepsilon/5$

$$\mu_i(\alpha_{i,j} + \beta_{i,j} + \gamma_{i,j}) = O(\log(1/\delta)(h_i + h'_i/q_i)) \tag{16}$$

We now consider the values of the remaining uncertain quantities heuristically. Taking the negative binomial distribution to be spread out smoothly as much as it's variance suggests would give $h_i = O(\varepsilon T'/i\sqrt{\log(1/\delta)})$, $q_i \leq h_i$ and $h'_i = O(q_i\varepsilon T'/i\sqrt{\log(1/\delta)})$. It then follows that the bound $q_1 < \varepsilon/5$ will hold for all i that are at least some $\Theta(T'/\sqrt{\log(1/\delta)})$ amount. To avoid having to worry about this we can take T to be that amount. Then for all $i > T$ we can approximate η_i with the maximum of $\mu_i(\alpha_{i,j} + \beta_{i,j} + \gamma_{i,j})$ which is given by $O(\varepsilon\sqrt{\log(1/\delta)})$.

We also neglect the cost of the η_j for $j > T'$ on the basis that after negligibly beyond T' these will go to zero very quickly.

The number of dummies for the first two steps is $\Theta(RT^2 + Rn/T')$, which if we take $T' = (n\log(1/\delta))^{1/3}$ is $\Theta(Rn^{2/3}\log(1/\delta)^{-1/3})$. Meanwhile, the cost for the third step is $O(T'\varepsilon\sqrt{\log(1/\delta)}) \leq \varepsilon n^{1/3}\log(1/\delta)$, which is a lower order term.

This suggests an improvement of at least a factor of $\log(1/\delta)^{-1/3}$ over the $T = T'$ case.

D DATA-OBLIVIOUS SPARSE HISTOGRAM

In this section we present a data-oblivious algorithm for sparse histograms. As described in Section 3, we can directly implement this functionality using generic MPC (e.g., garbled circuits), to obtain a baseline protocol.

Input: Index-value pairs $(\mathsf{ind}_i, \mathsf{val}_i)_{i\in[n]}$, with $\mathsf{val}_i \in [\Delta]$.
Algorithm:

(1) Obliviously sort $(\mathsf{ind}_i, \mathsf{val}_i)_{i\in[n]}$ by index.
(2) Linearly scan the sorted pairs from the previous step: For each unique value ind_i keep a single pair $(\mathsf{ind}_i, \mathsf{val}'_i)$, with val'_i as defined below, while replacing all other pairs with $(\perp, \perp)$.

$$\mathsf{val}'_i = \begin{cases} \perp & \text{if } s_i + \xi_i < \tau \\ s_i + \xi_i & \text{otherwise,} \end{cases}$$

where $s_i = \sum_{\{j \mid \mathsf{ind}_j = \mathsf{ind}_i\}} \mathsf{val}_j$), $\xi_i \leftarrow \mathsf{DLap}(2\Delta/\epsilon)$, and $\tau = \Delta + 2\Delta\log(2/\delta)/\epsilon$.
(3) Obliviously shuffle the pairs from the previous step and output the resulting array.

Figure 14: Data-oblivious implementation of the sparse histogram mechanism by Korolova et al. [57]

E A PROTOCOL BASED ON PRIVATE HEAVY-HITTERS

In our previous protocols, we used the duplication method to provide DP for the counts at histogram entries that occur at least T times in the clients' input. This technique was leveraging only the encrypted histogram entries IDs to provide DP without identifying what is the set of entries that have counts larger than T. On the other hand, if the servers are able to identify those entries, then they can directly add the appropriate DP noise with sensitivity one, which would protect the contributions of a single client. (This would indeed be more in line with the aforementioned central DP algorithm [61].)

Identifying all items that occur with frequency greater than a fixed threshold is the functionality of finding *heavy hitters*, which has been widely studied in the DP literature. It is thus natural to consider a solution consisting on (i) running a private HH protocol to identify frequent indices (with multiplicities above a threshold T) and (ii) noising the identified indices. These two steps would replace the duplication-based step mentioned above. We consider this type of approach which, while it reduces the communication cost coming purely from duplications of client contributions, it introduces communication from the secure computation evaluating the distributed PHH protocol. It also consumes from the DP budget for the whole execution to identify the heavy hitters. As a result the approach leveraging PHH as a first step has communication advantage both asymptotically and numerically only in settings with very small constant number of heavy hitters. We now present the details.

Assume that we had a protocol Π that the servers can run for P1 to obtain an ε_Π-DP estimate of the set of all T-heavy hitters, i.e., all indices with multiplicity greater than T in the input. Π should satisfy the correctness condition that if an index is a heavy hitter then the protocol will output it, except with probability bounded by δ_Π. Then, an alternative protocol relying on Π proceeds as follows. First, the servers run Π and P1 obtains indices $i_1, ..., i_h \in D$, an estimate of the list of indices whose multiplicity is greater than T.

Since P1 knows the indices i_j in the clear, it can simulate clients an appropriate random number of times analogously to how multiplicities below T are noised with Laplace noise in step (2) of Figure 5. This approach trades the dummy elements due to duplication in step (2) of our protocol by $O_{\varepsilon,\delta_1}(h)$ dummy elements, at the additional cost of running Π and its associated privacy budget $\varepsilon\Pi, \delta_{Pi}$. This approach is thus beneficial for skewed input distributions, where h is expected to be small.

Let us remark the need for the correctness requirement on Π mentioned above: consider two neighboring databases differing in the first input $(\mathsf{ind}_1, \mathsf{val}_1)$ and such that ind_1 has multiplicity $x > T$ in the input. If $\mathsf{ind}_1 \notin \{i_1, ..., i_m\}$, i.e., ind_1 is missed by Π, then x will not be noised sufficiently, which would break privacy. However, this happens with probability bounded by δ_Π due to the correctness requirement on Π.

E.1 Instantiating Π

In our experimental evaluation, presented next, we instantiated Π by a secure 2-party implementation of PrivTree [78]. This approach is analogous to the secure heavy hitters protocol of [16], which relies on a hierarchical decomposition to find heavy hitters, with the exception that a bias term is subtracted from the counts at every level of the hierarchy. This achieves that the total privacy cost is independent of the domain size $|D|$. In contrast, the approach of Boneh et al. [16] treats each level of the hierarchical histogram as an independent frequency oracle query, and applies composition across all $\log(|D|)$ levels. After experimenting with concrete parameters, the first approach was superior in our setting.

Enforcing the correctness requirement mentioned above imposes a constraint on the value of T that the mechanism can tolerate, as setting T too small might result in false negatives with probability larger than δ. Accounting for that fact yields the constraint $T > O\left(\frac{\log(1/\delta_\Pi)+\log(|D|)}{\varepsilon\Pi}\right)$. Since the per-entry noise mechanism to handle multiplicities below T has to operate on a value of T of at least that order, it adds a minimum $\Omega_{\varepsilon,\delta}(\log(|D|)^2)$ dummy contributions with this approach. Hence, setting T to its minimum acceptable value, the total number of dummy indices added in step (2) of our protocol (Figure 5) is $O_{\varepsilon,\delta}(\log(|D|)^2 + h)$. This shows the fact mentioned above that the communication overhead depends on h, which is a property of the general approach based on heavy hitters discovery. The dependence on the domain size, on the other hand, is a consequence of an instantiation using PrivTree.

E.1.1 Experimental Evaluation. We compare the per-client server communication of our best duplication-based protocol with a heavy-hitters based solution built upon a secure evaluation of PrivTree. We split the privacy budget for the duplication-based protocol as described in Section 6. For the heavy-hitters based solution we use a splitting so that the accuracy in the reported counts of the two approaches is the same. In that setting, we compare per-client server communication for several settings of the domain size (2^{32}, 2^{64}, and 2^{128}) in Figure 15, and values of epsilon. In all the experiments, the value of T used by the heavy hitters protocols is reported, and h, i.e., the assumed maximum number of T-heavy hitters in the input, is set to $100\log(n)$. We can see how for large enough n the heavy hitters based solution is competitive, but never results in a remarkable advantage. Experimenting with alternatives to PrivTree, as well as a more detailed empirical analysis, including validating choices of h using real-world data, is left for future work.

F ADDITIONAL EXPERIMENTAL RESULTS

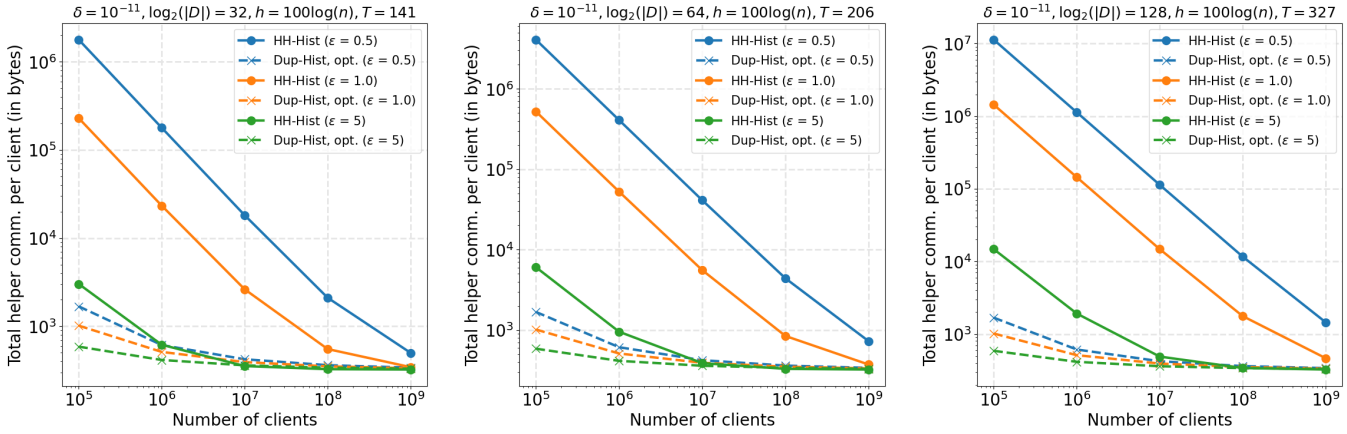

Figure 15: Comparison of the approaches based on duplication and a secure evaluation of Privtree [78].

ε	n	P1 offline	P1 online	P1 total	P1 comm.	P2 offline	P2 online	P2 total	P2 comm.	total time	total comm.
0.5	10^5	9.82	1.24	11.06	3077	1.13	0.84	1.96	353	13.03	3431
	10^6	3.08	0.64	3.71	965	1.03	0.27	1.30	324	5.01	1289
	10^7	1.87	0.53	2.40	587	1.02	0.16	1.19	321	3.59	908
	10^8	1.49	0.50	1.99	467	1.02	0.13	1.15	320	3.14	787
	10^9	1.34	0.48	1.83	422	1.02	0.12	1.14	320	2.97	742
1.0	10^5	5.63	0.86	6.49	1766	1.06	0.48	1.54	332	8.04	2098
	10^6	2.44	0.58	3.02	765	1.03	0.21	1.24	322	4.26	1087
	10^7	1.68	0.51	2.19	527	1.02	0.15	1.17	321	3.36	847
	10^8	1.42	0.49	1.91	445	1.02	0.13	1.15	320	3.05	765
	10^9	1.31	0.48	1.79	411	1.02	0.12	1.14	320	2.93	731
2.0	10^5	3.98	0.71	4.69	1247	1.04	0.34	1.38	325	6.07	1572
	10^6	2.10	0.55	2.65	660	1.02	0.18	1.21	321	3.86	981
	10^7	1.57	0.50	2.07	492	1.02	0.14	1.16	321	3.23	813
	10^8	1.37	0.49	1.86	431	1.02	0.12	1.14	321	3.00	751
	10^9	1.29	0.48	1.77	405	1.02	0.11	1.14	321	2.91	726

Table 3: Per-client computation (in ms) and communication cost (in bytes) of our protocol for different values of ε, n, and $\delta = 10^{-11}$, using Paillier as AHE. The client cost is 1.23 ms CPU time and 384 bytes of communication. See also table 2