Nebula: Efficient, Private and Accurate Histogram Estimation

Ali Shahin Shamsabadi[†], Peter Snyder[†], Ralph Giles[†], Aurélien Bellet[‡], and Hamed Haddadi[†,⋄]
[†] Brave Software [‡] Inria, Université de Montpellier [⋄] Imperial College London

Abstract

We present *Nebula*[1], a system for differentially private histogram estimation on data distributed among clients. *Nebula* allows clients to independently decide whether to participate in the system, and locally encode their data so that an untrusted server only learns data values whose multiplicity exceeds a predefined aggregation threshold, with (ε, δ) differential privacy guarantees. Compared to existing systems, *Nebula* uniquely achieves: *i)* a strict upper bound on client privacy leakage; *ii)* significantly higher utility than standard local differential privacy systems; and *iii)* no requirement for trusted third-parties, multi-party computation, or trusted hardware. We provide a formal evaluation of *Nebula*'s privacy, utility and efficiency guarantees, along with an empirical assessment on three real-world datasets. On the United States Census dataset, clients can submit their data in just 0.0036 seconds and 0.0016 MB (**efficient**), under strong $(\varepsilon = 1, \delta = 10^{-8})$ differential privacy guarantees (**private**), enabling *Nebula*'s untrusted aggregation server to estimate histograms with over 88% better utility than existing local differential privacy deployments (**accurate**). Additionally, we describe a variant that allows clients to submit multi-dimensional data, with similar privacy, utility, and performance. Finally, we provide an implementation of *Nebula*.

1 Introduction

Aggregated user data allows software developers and service providers to develop, deploy, and improve their systems in various use-cases such as browser telemetry [1], financial crime [2], and digital health [3]. However, large-scale collection of user data introduces privacy risks, as client data may contain privacy-sensitive information (e.g., client preferences/interests, transactions, and medical diagnoses [4, 5, 6, 7, 8, 9, 10, 11]).

In this work, we focus on the problem of *private distributed histogram estimation*, where a central server aims to estimate the frequencies of different data values distributed among a set of clients, while providing privacy guarantees to the clients.

Several approaches to private distributed histogram estimation have been proposed, either in published research or in deployed systems. One such technique is *threshold-aggregation*, where the server is able to learn client values if and only if sufficiently many clients contribute the exact same value [7, 6, 12]. Threshold-aggregation has the benefit of providing a simple and intuitive privacy model, but generally lacks robust, provable privacy guarantees, due to using the *deterministic* notion of K-anonymity for protecting the privacy of clients [13, 14].

A second type of approach to private distributed histogram estimation relies on *differential privacy* (DP) [15, 16] to provide formal privacy guarantees through statistical indistinguishability. A wide range of DP-based systems for privacy-preserving data collection have been proposed, all of which require implementers and deployers to make unappealing tradeoffs, even for state-of-the-art systems. Local DP systems [17, 18, 19, 20] provide strong privacy guarantees but generally poor utility, while central DP systems [21, 15, 16] provide high utility but require prohibitive levels of trust by clients.

A third category of private distributed histogram estimation systems attempt to achieve both high utility and privacy, but do so by introducing additional costs, relying for instance on computationally expensive, novel cryptography [22, 23, 24, 25, 8, 9], (e.g., multi-party computation, homomorphic encryption), and/or requiring multiple rounds of heavy communication between participants [9], among other concerns. These

[1] Accepted at the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS'25).

systems entail computational, bandwidth and financial costs that make adoption difficult-to-impossible for all but resource-rich organizations.

In this paper, we describe a novel system for the problem of private distributed histogram estimation[2] that avoids the limitations and trade-offs of existing approaches, achieving simultaneously provable differential privacy guarantees, high utility, and practical efficiency. Our system, called *Nebula*, is a novel combination of the recent sample-and-threshold mechanism [8] with verifiable client-side thresholding [7] to provide DP guarantees without the use of trusted third parties. *Nebula* relies on two *untrusted* (non-colluding) servers: a *randomness server* and an *aggregation server*, and in contrast to existing state-of-the-art systems (e.g., [25]), *no communication between the two servers* is required.

At a high level, each client participating in *Nebula* begins by randomly deciding whether to contribute any data. Clients that do decide to participate locally encode their value using a secret-sharing scheme, which prevents the server from observing uncommon values. This secret sharing process is very cheap, requiring only a single round of oblivious communication with the untrusted randomness server, which executes a verifiable oblivious pseudorandom function over the client's value. Participating clients then contribute their secret share to the aggregation server over an oblivious communication channel. The aggregation server then combines all received shares to recover values which have been contributed by a sufficient number of participants.

Nebula enforces formal differential privacy protection for all clients through three steps: *i)* the uncertainty of any particular client contributing *any* value; *ii)* blinding the aggregation server to uncommon values through the secret-sharing mechanism (i.e., thresholding); and *iii)* having some clients contribute precisely defined amounts of "dummy data" to obscure the distribution of uncommon (i.e., unrevealed) values.

In summary, we make the following contributions to the problem of private distributed histogram estimation:

1. the **design of a novel system** for conducting privacy preserving data aggregation under DP guarantees that achieves all of the following: *i)* high utility, particularly when compared to other DP-based systems with comparable privacy guarantees; *ii)* realistic trust assumptions; and *iii)* practical efficiency in terms of computational, bandwidth, and financial costs.
2. a **formal analysis** of the system's privacy, security, utility, and efficiency guarantees.
3. **empirical measurements** of the system's utility and efficiency over several real-world datasets. We provide an implementation demo of *Nebula* as supplementary material.

2 Problem & Threat Model

We consider a scenario where an untrusted service provider (the aggregation server) wants to obtain a histogram over N clients data, $D = \{x_i\}_{i=1}^{N}$, where data point x_i is held by the i-th client. Collecting data generated by clients and publishing the histogram might introduce privacy risks such as misusing information for profit or mass surveillance purposes [10] as clients' data contain privacy-sensitive information [4, 5, 6, 7, 8, 9, 10, 11]. Therefore, the data collection procedure and published histogram must protect the privacy of clients. We aim to design a system that enables the aggregation server to construct an **accurate** and **differentially private** histogram over clients data **without trusting servers** and **without imposing high computational and communication costs**. To achieve this, we rely on an additional untrusted party: the randomness server. We assume that the aggregation and the randomness server are non-colluding, which is a common assumption [9, 26, 7] as collusion can be made infeasible or too costly via physical means [27, 28] or strict legal regulations [29]. Following the literature [9], we consider honest-but-curious clients who submit their data through an anonymizing proxy.

To protect the clients' privacy, we use Differential Privacy (DP) [15, 16]. In particular, we design a randomized protocol $\mathcal{A}$ that outputs a histogram over clients data which is close to the true histogram of D while satisfying (ε, δ)-DP: $\Pr[\mathcal{A}(D) \in S] \leq e^{\varepsilon}\Pr[\mathcal{A}(D') \in S] + \delta$, for any subset of possible output histograms $S \in \text{Range}(\mathcal{A})$ and for any two neighboring datasets D and D' where D' is obtained by

[2] Our system can answer related problems such as heavy hitters and quantiles [8].

Algorithm 1: *Nebula*

Input: N clients, one randomness server, one aggregation server, Truncated Shifted Discrete Laplace distribution TSDLap$(\cdot)$, DP guarantee (ε, δ),
τ-out-of-N secret-sharing scheme $\Pi_{\tau,N}$, public key parameter pp, hash function $H(\cdot)$
Output: Clients' submissions revealed to the aggregation server

$(\varepsilon_{\text{Re}}, \delta_{\text{Re}}), (\varepsilon_{\text{Unre}}, \delta_{\text{Unre}}) \leftarrow (\varepsilon, \delta)$ ▷ `All parties agree on sample-and-threshold (Re) and dummy-data (Unre) DP guarantees`
$p_s, \tau \leftarrow (\varepsilon_{\text{Re}}, \delta_{\text{Re}})$ ▷ `Computing sampling rate and aggregation threshold`
for $i = 1, \ldots, N$ **do**
  $r_i = \textit{Client-RandomnessServer}(x_i, \text{pp}, H(\cdot))$ ▷ `Oblivious and verifiable randomness generation (Algorithm 2)`
  sbm, _ $\leftarrow \textit{LocalSecretSharing}(x_i, r_i, \Pi_{\tau,N})$ ▷ `Each client locally encrypts their data (Algorithm 3)`
  $z_i = \text{Random}([0, 1])$ ▷ `Each client locally performs a Bernoulli test to decide whether to participate`
  if $z_i \leq p_s$ **then**
    Submit sbm to the aggregation server
Dummy $= \textit{DummyDataCreation}(\tau, \text{TSDLap}(\cdot), (\varepsilon_{\text{Unre}}, \delta_{\text{Unre}}))$ ▷ `Dummy data creation to protect unrevealed submissions (Algorithm 4)`
Submit Dummy to the aggregation server
ReceivedData $= (\text{sbm} \cup \text{Dummy})$ ▷ `Received encrypted data containing indistinguishable dummy and real messages`
RecoveredData $= \textit{Aggregation}(\text{ReceivedData})$ ▷ `The aggregation server performs data aggregation and recovery (Algorithm 5)`
Return RecoveredData

removing one client's data from D. The privacy budget ε upper bounds the privacy leakage in the worst possible case. The smaller the ε, the stronger the privacy guarantees. Setting $\delta > 0$ allow to relax the privacy requirement for unlikely events. Specifically, our protocol guarantees that the final output (i.e., published histogram) satisfies DP, while the aggregation server's view satisfies computational DP [30], a restriction of DP to computationally bounded adversaries commonly considered in privacy-preserving secure protocols. Among other cryptographic schemes, we use a τ-out-of-N secret sharing scheme [31], $\Pi_{\tau,N}$, built out of two standard functionalities: 1) producing a random τ-out-of-N share of a private value through a probabilistic algorithm with explicit randomness received as input; 2) recovering the private value after receiving at least its τ valid secret shares. The randomness server generates randomness required for $\Pi_{\tau,N}$, without seeing any plaintext clients' data and in a verifiable manner (i.e., clients can verify whether the randomness server has correctly followed the protocol in zero knowledge).

To ensure *efficiency* by minimizing financial costs, computational overhead, and bandwidth consumption, our system incorporates the following design principles: 1) it avoids any communication between the randomness server and the aggregation server; 2) it precludes communication between clients; and 3) it requires minimal efforts from clients, with a single single round of interaction with each server. Avoiding such communications and interactions also makes the practical deployment of non-collusion assumptions more feasible and easier to maintain.

Note that the server and clients agree on two public parameters: *i)* the desired differential privacy guarantee (ε,δ); and *ii)* the security parameter κ used for the secret sharing scheme.

3 *Nebula* Design

We present the design of our novel DP and secure system, called *Nebula*. *Nebula* requires no communication between clients, and only requires two *non-cooperating servers* (one that operates an oblivious and verifiable pseudorandom function [32], and one that aggregates and learns threshold-meeting values from clients).

Algorithm 2: *Client-RandomnessServer*: Interaction between clients and the randomness server

Input: A client holding a private item x, a randomness server, a secret key msk, hash function $H(\cdot)$
Output: Randomness r

1: $h = H(x)$ ▷ `Client hashes its value`
2: $r' \leftarrow R$ ▷ `Client generates a random value`
3: $b = h^{r'}$ ▷ `Client sends blinded hash to the server`
4: $\mathrm{z} = b^{\mathrm{msk}}$ ▷ `Server responds with its ZKproof`
5: $w = \mathrm{z}^{\frac{1}{r'}}$ ▷ `Client unblinds the response`
6: $r = H(w, x)$ ▷ `Client obtains the randomness`
7: **Return** r

At a high level, *Nebula* (Algorithm 1) works as follows: *i)* Each client independent of other clients obliviously communicates with the randomness server and encrypts its data; *ii)* Each client performs a Bernoulli test on whether to participate: with probability p_s it participates and sends its encrypted data to the server, otherwise it abstains; *iii)* A randomly selected client submits dummy data by creating groups of dummy data for each possible group of unrevealed items in $\{1, ..., \tau - 1\}$ to bound the information that the aggregation server might learn from unrevealed submissions; *iv)* The aggregation server receives real submissions and dummy data, and performs the decoding such that it learns aggregate submissions shared by at least τ clients in the sampled set. Note that the system is designed such that dummy data does not impact the correctness and utility of the aggregations (see Section 3.3). In the rest of this section, we describe each of these steps in detail.

3.1 Oblivious & Verifiable Randomness

Each client starts by sampling randomness r from the randomness server that runs a Verifiable Oblivious PseudoRandom Function (VOPRF) [7] that adheres to the standard ideal functionality [33] with security guarantees proven in the Universal Composability framework [34]. This VOPRF construction allows clients that contribute the same original value to consistently receive the same randomness r, while ensuring that: i) the randomness server does not learn the clients' values; ii) the server cannot detect when multiple clients share the same input; iii) clients do not learn the server's PRF keys; and iv) no communication is required between clients.

Server-side setup. The randomness server initializes the VOPRF by generating public cryptographic parameters $\mathrm{pp} \leftarrow \mathrm{VOPRF.setup}(1^\kappa)$ given the security parameter κ. The randomness server then generates a keypair $(\mathrm{msk}, \mathrm{mpk}) \leftarrow \mathrm{KeyGen}(\mathrm{pp})$, consisting of a secret key msk and a public key mpk, using a Key Generation algorithm KeyGen parameterized by pp.

Once the VOPRF setup is complete, each client interacts with the randomness server to obtain its randomness r, as described in Algorithm 2 and outlined below.

Client-side requests. Each client produces a request using their input data as follows. The client first samples a local blinding factor r', then computes a blinded representation of their original data value x as $b = H(x)^{r'}$. The client then sends the blinded value b to the untrusted randomness server.

Server-side responses. Upon receiving b, the randomness server evaluates the VOPRF function by computing $z = b^{\mathrm{msk}}$ using their secret key msk. The randomness server returns z to the client.

Finally, the client unblinds the received response($w = z^{1/r'}$) and derives the final pseudorandom output as ($r = H(w, x)$).

3.2 Local Data Preparation and Submission

To secret-share the data (Algorithm 3), the client parses r into $\{r_1, r_2, r_3\}$ using a random oracle model hash function. Each of these three randomness components are used for different purposes.

Algorithm 3: *LocalSecretSharing*: Client data encoding

Input: A client holding a data point x, randomness r, τ-out-of-N secret-sharing scheme $\Pi_{\tau,N}$
Output: Encoded data sbm

$r_1, r_2, r_3 \leftarrow H(r_a\|1), H(r_b\|2), H(r_c\|3)$ such that $r_a\|r_b\|r_c = r$ ▷ `Parsing the randomness to three random values`
Key = PseudorandomGenerator(r_1) ▷ `Deriving a symmetric key using pseudorandom generator with` r_1 `as the seed`
$\mathbf{c} \leftarrow$ Enc(Key, $\mathbf{x}$) ▷ `Encrypting the data and generating a ciphertext`
$t \leftarrow r_3$ ▷ `Generating a tag for the data using` r_3
$s = \Pi_{\tau,N}(r_1; r_2)$ ▷ `Constructing a secret-share of the random value` r_1 `used for deriving the encryption key`
sbm $\leftarrow (\mathbf{c}, s, t)$ ▷ `Creating a tagged submission for the data`
Return sbm, Key

Algorithm 4: *DummyDataCreation*: Create groups of dummy data

Input: A public thresholding value τ, Truncated Shifted Discrete Laplace distribution TSDLap($\cdot$), DP guarantees $(\varepsilon_{\text{Unre}}, \delta_{\text{Unre}})$
Output: A set of dummy data

Dummy = {} ▷ `The set containing groups of dummy data`
Select a client for creating dummy data
for $i = 1, \ldots, \tau - 1$ **do**
  $\alpha \leftarrow \text{TSDLap}(\lambda = 2/\varepsilon_{\text{Unre}}, t = 2 + 2/\varepsilon_{\text{Unre}} \log(2/\delta_{\text{Unre}}))$
  $\{\text{t}_j\}_{j=1}^{\alpha}$ = UniqueTagGenerator(α) ▷ `Unique tags`
  for $j = 1, \ldots, \alpha$ **do**
    $s_j = \{(c_j, s_j, \text{t}_j)^i\}$ ▷ `The client creates a set containing` i `zero-value items with the same unique tag`
    Dummy.append(s_j)
Return Dummy

r_1 is used to seed a PR generator function and derive a Key for a symmetric encryption scheme which satisfies IND-CPA security and consists of two algorithms: i) encryption: producing ciphertext of a data with key; and ii) decryption: outputting a data given its ciphertext under the key. Using the encryption algorithm, we obtain the encrypted client's input data $\mathbf{c} = \text{Enc}(\text{Key}, \mathbf{x})$. r_2 is used as the randomness input to $\Pi_{\tau,N}$ for producing a random τ-out-of-N share[3] $s_k \in F_q$ of r_1. $\Pi_{\tau,N}$ operates in a finite field F_q for some prime $q > 0$ with information-theoretic security [31]. $\Pi_{\tau,N}$ consists of two algorithms: i) share: producing a random share of the data with a particular randomness; ii) recover: reconstructing the data given at least its τ valid shares. r_3 is used as a *tag* informing the aggregation server which shares to combine to recover the encryption key. Each client constructs their message as sbm $\leftarrow (\mathbf{c}, s, r_3)$.

Then, each client performs a Bernoulli test on whether to participate: with probability $p_s = n/N$ (where n is the expected size of the sampled clients) it sends its encrypted message sbm $\leftarrow (\mathbf{c}, s, r_3)$ to the aggregation server, otherwise it abstains.

3.3 Dummy Data Injection

While the aggregation server cannot recover the data values submitted by less than τ clients, it does observe the tags of these unrevealed submissions.[4] The aggregation server thus learns the multiplicity of unrevealed submissions sharing the same tag, which could potentially expose information about their underlying values if the server possesses additional side information. To control this leakage, *Nebula* adds dummy submissions such that the amount of information that the aggregation server can learn

[3]Note that our implementation of τ-out-of-N secret sharing produces random shares without any client's identity.
[4]The cost that we pay in favour of enabling the aggregation server to do the aggregation by itself without any interaction with other servers/clients in practical scenarios.

Algorithm 5: *Aggregation*: Aggregating and recovering client data

Input: Received M clients' submissions $\{\text{sbm}_i\}_{i=1}^{M}$ where each submission $\text{sbm}_i \leftarrow (\mathbf{c}, s, t)$ contains an encrypted data $\mathbf{c}$, a random τ-out-of-N share s and a tag t, τ-out-of-N secret-recovering scheme $\Pi^{-1}_{\tau,N}$, PseudorandomGenerator

Output: Decoded clients' data RecoveredData

1: RecoveredData = {}
2: {groups} = GroupBasedOnTags(sbm_i) ▷ `Grouping submissions based on their tags` t `such that all submissions in each group share the same tag`
3: **for** group ∈ groups **do**
4: $(r_1, \perp) \leftarrow \Pi^{-1}_{\tau,N}(\{s_k\} \quad \forall s_k \in \text{group})$ ▷ `Recovering the share if the group contains at least` τ `submissions (i.e., secret-shares)`
5: **if** r_1 **then**
6: Key = PseudorandomGenerator(r_1) ▷ `Recovering the decryption key using` r_1 `to seed the pseudorandom generator`
7: $\mathbf{x}$ = Dec(Key, $\mathbf{c}$) ▷ `Decrypting one of the data within the group`
8: RecoveredData.append($\underbrace{\mathbf{x}, \mathbf{x}, \ldots, \mathbf{x}}_{|\text{group}| \text{ times}}$) ▷ `Outputting as many data as the size of the group`
9: **Return** RecoveredData

about these tags is bounded within the DP guaranteed range (Algorithm 4[5]). Dummy data makes the histogram of unrevealed submissions differentially private. This dummy data injection can be done by randomly selecting a client, and it does not affect the correctness and utility of the aggregations, as dummy data is automatically filtered out because each dummy group is smaller than the threshold (see line 3 in Algorithm 4).

We use the truncated shifted discrete Laplace distribution supported on $\{0, ..., 2t\}$, denoted by TSDLap(λ, t), to generate a positive, bounded number of dummy data.

definition 1 (Truncated Shifted Discrete Laplace Distribution)**.** *The Truncated Laplace Distribution on* $\{0, ..., 2t\}$ *is defined as:*

$$f_{TSDLap(\lambda,t)}(c) = \begin{cases} \frac{\exp\left(-\frac{|c-t|}{\lambda}\right)}{A}, & \text{if } c \in \{0, ..., 2t\} \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where $\lambda \in (0, 1)$ *is the scale parameter and* $A = \sum_{c=0}^{2t} \exp\left(-\frac{|c-t|}{\lambda}\right) = 1 + 2\sum_{c=1}^{t} \exp\left(-\frac{c}{\lambda}\right)$ *is the normalization constant.*

Section 4 proves that subsampling clients, combined with dummy data and thresholding, provides DP guarantees [35, 8].

3.4 Data Aggregation and Recovery

Clients submit their secret-shared values to the aggregation server through an anonymizing proxy, delinking the submitted value from any other information identifying the submitter (e.g., IP address, etc). We deploy an Oblivious HTTP [36] server which is an IETF draft standard. Oblivious HTTP removes client-identifying information from HTTP requests containing client submissions to blind the aggregator server from learning which client is submitting which reports, and which reports are being submitted by the same user. Following the literature,[6] we assume that submissions do not contain timestamps of when data is generated. In practical deployments where timestamps are observed, we need to make

[5]The efficient version of Algorithm 7 used in our security proof (Appendix A). The only difference is that the client constructs dummy submissions locally instead of interacting with the randomness server, trivially tolerated by our security proof.

[6]https://machinelearning.apple.com/research/learning-with-privacy-at-scale#AppleSecurity

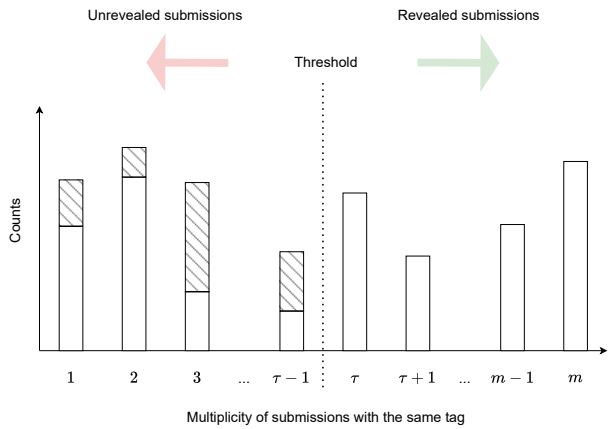

Figure 1: *Nebula*'s output to the aggregation server consists of a histogram $\mathcal{H}$ of multiplicities where $\mathcal{H}_i$ represents the number of submissions with the same tag, with multiplicity i and $i \in [m]$. This histogram is obtained based on submissions that each client sent with probability p_s (empty bar) and dummy data (hatched bar).

the distribution of each timestamp independent of the messages and their source such that it gives no additional information about the sender. This could be done in various ways. For instance, whenever a client's data is encoded on their device, the client locally draws a real number `num` uniformly into $[0, 1]$ and send the message at time `num`.

The aggregation server then recovers any values submitted by at least τ clients using the share recovery algorithm on the corresponding share values, s_k, to recover r_1 and thus the corresponding data value. As described in Algorithm 5, the aggregation groups submissions based on their tags such that all submissions in each group share the same tag. Then, the aggregation can learn the submission within groups with cardinality of at least τ through performing the following sequential recoveries: 1) the share value s from its τ secret shares s_k; 2) r_1 from s; 3) the encryption key Key from r_1; 5) the client submission using Key as the decryption key.

4 Privacy, Security, Utility and Communication Analysis

In this section, we analytically demonstrate that *Nebula* is a *secure* protocol for producing *private* and highly *accurate* data outputs with *low communication costs*.

4.1 Privacy Analysis

Theorem 1. *Consider N clients generating a dataset $D = \{x_i\}_{i=1}^{N}$. Let ε_{Unre} be the privacy budget used in the creation of dummy data (Algorithm 4). For $\varepsilon_{Re} > 0$ and $\delta_{Re} \in (0, 1)$, let $p_s = \alpha(1 - e^{-\varepsilon_{Re}})$ and $\tau = \frac{1}{C_\alpha} \ln(\frac{1}{\delta_{Re}})$ where $0 < \alpha \leq 1$ and $C_\alpha = \ln(\frac{1}{\alpha}) - \frac{1}{1+\alpha}$. Then, the view of the aggregation in Nebula satisfies computational (ε, δ)-DP with $\varepsilon = \max(\varepsilon_{Unre}, \varepsilon_{Re})$ and $\delta = \max(\delta_{Unre}, \delta_{Re})$.*

Proof. Let D and D' be two neighboring input datasets such that D' is obtained by removing one client's data from D. We aim to show that the view of the aggregation server satisfies (computational) (ε, δ)-DP. We split our analysis according to two mutually exclusive events (see Figure 1): either the value corresponding to the extra client in D is revealed (i.e., the corresponding count is greater than or equal to τ), or it is not.

Unrevealed submission. For unrevealed submissions, the server only learns the multiplicity of submissions with the same tag. Let $\mathcal{Z}_{\text{Unre}}$ be the histogram of the unrevealed submissions computed as $\mathcal{Z} + \mathcal{N}$ where $\mathcal{Z}$ is the histogram of multiplicities of the "genuine" submissions (i.e., $\mathcal{Z}_i$ counts the number

of unrevealed tags with multiplicity i) and $\mathcal{N}$ is the noise corresponding to the addition of dummy contributions. Recall that dummy data are drawn from a domain disjoint from the original domain, so adding i dummies with the same tag is equivalent to adding noise of value one to the i-th histogram entry $\mathcal{Z}_i$. As $\mathcal{Z}$ does not have any multiplicity above $\tau - 1$, the protocol only adds $i \in [\tau - 1]$ different such contributions. We know that noise $\mathcal{N}$ sampled from the truncated shifted discrete Laplace distribution $\mathrm{TSDLap}(\lambda, t)$ on $\{0, ..., 2t\}$ with $\lambda = \Delta/\varepsilon_{\mathrm{Unre}}$ and $t = \Delta + \Delta/\varepsilon_{\mathrm{Unre}} \log(2/\delta_{\mathrm{Unre}})$ to ensure $(\varepsilon_{\mathrm{Unre}}, \delta_{\mathrm{Unre}})$-DP [9]. We compute the sensitivity Δ as follows. Removing a client's data from D decreases the count of the corresponding multiplicity i by one while increasing the count of multiplicity $i-1$ by one, resulting in $\mathcal{Z}$ and $\mathcal{Z}'$ (computed on D and D' respectively) that differ in two adjacent entries i and $i-1$:

$$\begin{cases} \mathcal{Z}_i & = \mathcal{Z}'_i + 1 \quad \text{entry } i \\ \mathcal{Z}_{i-1} & = \mathcal{Z}'_{i-1} - 1 \quad \text{entry } i-1 \\ \mathcal{Z}_y & = \mathcal{Z}'_y \quad \text{other entries } \forall y \notin \{i, i-1\} \end{cases} \tag{2}$$

Therefore the sensitivity $\Delta = 2$.

Revealed submission. In the event where the differing submission is revealed, we can leverage DP guarantees of the sample-and-threshold approach [8]. For completeness and clarity, we give the full proof below. The bound on the ratio of the probability of the aggregation server receiving and decoding a group of submissions with the same tag and multiplicity i on D and D' is computed as follows. Let k be the multiplicity of the extra client's data item in D. The probability of seeing a count of $v \geq \tau$ copies of this item in the output of D is given by the Binomial theorem:

$$\binom{k}{v}(1-p_s)^{k-v}(p_s)^v, \tag{3}$$

and the probability of seeing a count of v copies of the same data in the output of D' who holds $k-1$ copies of the data is

$$\binom{k-1}{v}(1-p_s)^{((k-1)-v)}(p_s)^v. \tag{4}$$

Now, we can bound the ratio of probabilities of seeing data with a given count v by dividing Eq. 3 by Eq. 4 which is $\frac{(1-p_s)k}{k-v}$.

Next, we show that the ratio $\frac{(1-p_s)k}{k-v}$ is between the interval $(e^{-\varepsilon_{\mathrm{Re}}}, e^{\varepsilon_{\mathrm{Re}}})$ except with some small probability.

For the lower bound, we have

$$e^{-\varepsilon_{\mathrm{Re}}} \leq \frac{(1-p_s)k}{k-v}, \tag{5}$$

for any $v \geq 0$, which is satisfied if we ensure $p_s \leq 1 - e^{-\varepsilon_{\mathrm{Re}}} < 1$ (since $v = 0$ is the worst case).

For the upper bound, we have:

$$\frac{(1-p_s)k}{(k-v)} \leq e^{\varepsilon_{\mathrm{Re}}}. \tag{6}$$

Rearranging the upper bound, we have:

$$v \leq k(1 - e^{-\varepsilon_{\mathrm{Re}}} + e^{-\varepsilon_{\mathrm{Re}}} p_s) \tag{7}$$

Note that:

1. Since $p_s < 1$, then $p_s(1 - e^{-\varepsilon_{\mathcal{A}_{\mathrm{Re}}}}) < 1 - e^{-\varepsilon_{\mathrm{Re}}}$ and so $p_s < (1 - e^{-\varepsilon_{\mathrm{Re}}} + e^{-\varepsilon_{\mathrm{Re}}} p_s)$.
2. The bound in eq. (6) is greater than kp_s, the mean value.
3. Since $p_s < 1$, then $(1 - e^{-\varepsilon_{\mathrm{Re}}} + e^{-\varepsilon_{\mathrm{Re}}} p_s) < 1$, so we can define the probability $q = (1 - e^{-\varepsilon_{\mathrm{Re}}} + e^{-\varepsilon_{\mathrm{Re}}} p_s)$.

As all revealed submissions have a count at least equal to τ, we can thus obtain $(\varepsilon_{\text{Re}}, \delta_{\text{Re}})$-DP by bounding the probability δ_{Re} of choosing a v that is more than $\max(kq, \tau)$. Using the Chernoff-Hoeffding bound for the binomial distribution as done in [8], we get $\delta_{\text{Re}} \leq \exp(-\frac{\tau}{q} D(q\|p))$. Therefore, sampling with probability $p_s = \alpha(1 - e^{-\varepsilon_{\text{Re}}})$ and thresholding with $\tau = \frac{1}{C_\alpha} \ln(\frac{1}{\delta_{\text{Re}}})$ where $0 < \alpha \leq 1$ and $C_\alpha = \ln(\frac{1}{\alpha}) - \frac{1}{1+\alpha}$ provides $(\varepsilon_{\text{Re}}, \delta_{\text{Re}})$-DP [8]. □

Note that the aggregation server can publish the final histogram, which satisfies DP with the same ε and δ. Also, as discussed in Section 2 and Section 3, the randomness server learns nothing about the plaintext version of the client's data thanks to the underlying cryptographic schemes.

4.2 Cryptographic Security

Figure 2 represents the ideal functionality for *Nebula*. We leverage the methodology of [7] for producing consistent data encryption. Appendix A provides all correctness and security proofs of *Nebula* following similar arguments to [7].

4.3 Communication Analysis

Each client performs only one round of interaction with the randomness server to obtain the necessary randomness for the secret sharing. In particular, each client submits a 32 byte message consisting of their blinded hash value (see b line 3 in Algorithm 2). In response, the client receives another 32 bytes (see z line 5 in Algorithm 2) from the randomness server. Each client performs only one single interaction with the aggregation server. In particular, each client submits a 266 byte message, sbm (see line 6 in Algorithm 3), consisting of an alignment tag (32 bytes), a share of the encryption key (192 bytes), and their value (approximately 42 bytes). In addition to this, one client needs to send dummy data to the aggregation server.

Proposition 1. *The expected and worst-case number of dummy data is* $t\frac{(\tau-1)\tau}{2}$ *and* $2t\frac{(\tau-1)\tau}{2}$ *where* $t = 2 + 2/\varepsilon_{Unre} \log(2/\delta_{Unre})$ *is the expectation of the truncated shifted discrete Laplace distribution and* τ *is the threshold for pruning values.*

Proof. As discussed in Section 3, we use truncated shifted discrete Laplace distribution, $\text{TSDLap}(\lambda, t)$ on $\{0, ..., 2t\}$ to generate dummy data. The expectation of $\text{TSDLap}(\lambda, t)$ is t and its maximum value is $2t$. We sample $\tau - 1$ times from the truncated shifted discrete Laplace distribution and each time generate a group of submissions whose cardinality is the same as the bin value. Therefore, the expected and the maximum number of dummy submissions are $t\frac{(\tau-1)\tau}{2}$ and $2t\frac{(\tau-1)\tau}{2}$, respectively. □

As alignment tags for the dummy data are random they can be generated locally without any communication with the randomness server. However, the submitting client needs to send as many messages as the size of the group to the aggregation server.

4.4 Utility Analysis

The utility of *Nebula* is unaffected by the inclusion of dummy data: only the sampling rate p_s and the threshold τ affect the accuracy of the histogram estimated by *Nebula* relative to the true histogram. Leveraging utility guarantees of sample-and-threshold approach [8], we can show that *Nebula* reveals to the aggregation server, with high probability, any value whose frequency is sufficiently above the threshold.

Lemma 2. *Nebula removes a value that is shared by* W *clients with probability at most* $\exp(-(p_s W - \tau)^2 \frac{1}{2Wp_s})$*, where* p_s *and* τ *are Nebula's parameters: the client sampling rate and pruning threshold.*

Ideal Functionality $\mathcal{F}_{\text{Nebula}}$

Participants:

- Aggregation server S_A
- Randomness server S_R
- Clients $\{C_i\}_{i=1}^N$

Public parameters:

- DP parameters ε, δ where $\varepsilon = \max(\varepsilon_{\text{Unre}}, \varepsilon_{\text{Re}})$ and $\delta = \max(\delta_{\text{Unre}}, \delta_{\text{Re}})$
- Noise parameters $\lambda = \frac{2}{\varepsilon_{\text{Unre}}}$ and $t = 2 + \frac{2}{\varepsilon_{\text{Unre}}} \log(\frac{2}{\delta_{\text{Unre}}})$.
- Threshold $\tau = \frac{1}{C_\alpha} \ln(\frac{1}{\delta_{\text{Re}}})$ and Subsampling rate $p_s = \alpha(1 - e^{-\varepsilon_{\text{Re}}})$

Inputs:

- Client $C_i \in \{C_i\}_{i=1}^N$: provides input (x_i, aux_i)
- S_R: provides VOPRF keypair (msk, mpk)
- S_A: provides nothing $\perp$

Functionality:

1. For each client $C_i \in \{C_i\}_{i=1}^N$, sample $b_i \leftarrow \text{Bern}(p_s)$
2. $\mathcal{C} \leftarrow \{C_i | b_i = 1\}$
3. Sample $\{\alpha_i\}_{i=1}^{\tau-1}$ where each α_i is sampled independently from $\alpha_i \leftarrow \text{TSDLap}(\lambda = 2/\varepsilon_{\text{Unre}}, t = 2 + 2/\varepsilon_{\text{Unre}} \log(2/\delta_{\text{Unre}}))$.
4. For each α_i, construct α_i groups of dummy clients of size i with input $(\omega_{i,j}, \text{aux}_{i,j})$ for all $j \in \alpha_i$. Each $\omega_{i,j}$ is a distinct measurement outside the set of client measurements $\{x_i\}_{i=1}^N$. Call the set of all dummy clients $\mathcal{D}$.
5. For each unique x_ℓ received from $\mathcal{C} \cup \mathcal{D}$, construct:

$$\mathcal{E}_\ell = \{(x_\ell, \{x_j\}_{j \in J}, \tau_\ell) : (J \subseteq [N]) \wedge (x_j = x_\ell)\}$$

   where $\tau_\ell = |\{\text{x}_j\}_{j \in J}|$ is the number of sampled client measurements in $\mathcal{E}_\ell$.
6. Let $\mathcal{Y}$ be an empty map.
7. For each $\mathcal{E}_\ell$ where $\tau_\ell \geq \tau$, set $\mathcal{Y}[x_\ell] = \mathcal{E}_\ell$.

Outputs:

- C_i: learns nothing $\perp$
- S_R: learns what it would normally learn during the VOPRF exchange, $\{\mathcal{F}_{\text{VOPRF}}(\text{msk}, x_i)\}_{i=1}^N$
- S_A: learns a (ε, δ) differentially private histogram of clients' data, $\mathcal{Y}$, and the cardinality of each group of inputs $n_\ell \equiv |S_\ell|$ where S_ℓ is the set of submissions from $\mathcal{C} \cup \mathcal{D}$ which share the same unique measurement x_ℓ.

Figure 2: Ideal functionality for *Nebula*.

Algorithm 6: Nested-*Nebula*

Input: N clients, each client i holding a multi-dimensional data point $\mathbf{x}_i = [x_i^{(1)}, x_i^{(2)}, ..., x_i^{(L)}]$ with L attributes, all other inputs to Algorithm 1

Output: Clients' multi-dimensional submissions revealed to the aggregation server

$(\varepsilon_{\text{Re}}, \delta_{\text{Re}}), (\varepsilon_{\text{Unre}}, \delta_{\text{Unre}}) \leftarrow (\varepsilon, \delta)$ and $p_s, \tau \leftarrow (\varepsilon_{\text{Re}}, \delta_{\text{Re}})$
for $i = 1, \ldots, N$ **do**
  SBM $= \{\}$
  for $\ell = 1, \ldots, L$ **do**
    $\mathbf{x}_i^{(:\ell)} = [x_i^{(1)}, \cdots, x_i^{(\ell)}]$
    $r_i^{(\ell)} =$ *Client-RandomnessServer*$(\mathbf{x}_i^{(:\ell)}, \text{pp}, H(\cdot))$
    $\text{sbm}_i^{(\ell)}, \text{Key}_i^{(\ell)} \leftarrow$ *LocalSecretSharing*$(\mathbf{x}_i^{(:\ell)}, r_i^{(\ell)}, \Pi_{\tau,N})$
    if $\ell == 1$ **then**
      $\widehat{\text{sbm}_i}^{(\ell)} \leftarrow \text{sbm}_i^{(\ell)}$
    else
      $\widehat{\text{sbm}_i}^{(\ell)} \leftarrow \text{Enc}(\text{Key}_i^{(\ell-1)}, \text{sbm}_i^{(\ell)})$
    SBM.append$(\widehat{\text{sbm}_i}^{(\ell)})$
  $z_i = \text{Random}([0, 1])$
  if $z_i \leq p_s$ **then**
    Submit SBM to the aggregation server
Dummy $=$ *DummyDataCreation*$(\tau, \text{TSDLap}(\cdot), (\varepsilon_{\text{Unre}}, \delta_{\text{Unre}}))$
Submit Dummy to the aggregation server
ReceivedData $= (\text{SBM} \cup \text{Dummy})$
for $\ell = 1, \ldots, L$ **do**
  if $\ell == 1$ **then**
    RecoveredData$^{(\ell)} \leftarrow$ *Aggregation*(ReceivedData$^{(\ell)}$)
  else
    ReceivedDataWithKeys$^{(\ell)} \leftarrow \text{Dec}(\text{Key}^{(\ell-1)}, \text{ReceivedData}^{(\ell)})$
    RecoveredData$^{(\ell)} =$ *Aggregation*(ReceivedDataWithKeys$^{(\ell)}$)
Return RecoveredData

5 Nested-*Nebula*: a variant for high-dimensional marginal histograms

In some scenarios, clients' data consist of multiple attributes [37, 38, 39]. In this case, each client i holds a multi-dimensional data point that is a vector of $L \geq 2$ attributes of the form $\mathbf{x}_i = [x_i^{(1)}, x_i^{(2)}, ..., x_i^{(L)}]$. Consider the following motivating and practical example in the case of telemetry [11]. Clients generate some five-dimensional crash reports while using a Web Browser. Clients are anonymous and each dimension is a separate attribute: *i)* URL visited; *ii)* the underlying operating system; *iii)* the state of the device's battery; *iv)* session; and *v)* token IDs [40]. These attributes form a client's crash report. The service provider would like to learn the *marginal histogram* (i.e., the frequency among any joint sequence of attributes) to optimize and improve their application. Therefore, the service provider wants to maximize the utility of marginal histogram estimations and get better utility than treating client data that are made up of multiple attributes as a single data. Indeed, treating all attributes as a single data point means only those clients whose multi-dimensional data match exactly across all attributes are considered to have the same value, and this typically rarely happens in real datasets with many attributes. One straightforward solution would be to share each individual attribute or sequence of joint attributes, but this increases privacy risks.

To address these limitations, we propose Nested *Nebula* (Algorithm 6) that employs a novel multi-dimensional data encoding with a "hierarchy-of-priority" in which each client constructs a ciphertext by iteratively encrypting their ordered attributes such that the decrypting process halts when facing a low-frequency attribute that might risk the privacy of clients.

Multi-dimensional local data encryption. In Nested-*Nebula*, each client i creates L sequential prefixes

$\mathbf{x}_i^{(:1)}, \ldots, \mathbf{x}_i^{(:L)}$ such that each prefix $\mathbf{x}_i^{(:\ell)} = [x_i^{(1)}, \cdots, x_i^{(\ell)}]$ contains the sequence of attributes from the beginning to its corresponding index $\ell \in \{1, \ldots, L\}$. These prefixes enable to capture joint histograms of multiple attributes instead of each individual attribute. Each client encodes prefixes $\mathbf{x}^{(:\ell)}$ such that rare prefixes (i.e., a sequence of attributes which are not common across clients) cannot be decoded (i.e., kept hidden from the server). Each client i secret-shares each prefix ℓ through running Algorithm 2 and Algorithm 3 and construct message $\text{sbm}_i^{(\ell)}$. Submitting $\text{sbm}_i^{(1)}, \text{sbm}_i^{(2)}, ..., \text{sbm}_i^{(\ell)}$ separately in ℓ individual messages would result in two issues: *i)* it increases privacy loss and reveals all tags that might leak information; and *2)* it increases the overhead for both clients and servers. To address these issues, we chain the prefix contributions of each client (see lines 6-9 of Algorithm 6) and create one single super-message SBM such that decoding the attributes in the previous prefix would only then allow unlocking the next-longer prefix. In particular, we construct super-messages that can be iteratively opened to reveal higher levels of granularity (more attributes) when the previous prefix is shared by at least τ clients. To do this, client i encrypts each prefix $\text{sbm}_i^{(\ell)}$ with the key of its previous prefix which gets revealed once the previous prefix is decoded. Each prefix at layer ℓ (except for the first) is then encrypted with the key of the previously encoded prefix. In particular, each client computes an encrypted ciphertext $\widehat{\text{sbm}}_i^{(\ell)} \leftarrow \text{Enc}(\text{Key}^{(\ell-1)}, \text{sbm}_i^{(\ell)})$ with the described symmetric encryption operation for each $\ell \in \{2, \ldots, L\}$. Finally, each client i creates the super-message as the tuple $\mathsf{SBM} = (\widehat{\text{sbm}}_i^{(1)}, \ldots, \widehat{\text{sbm}}_i^{(L)})$ and sends it to the server based on the outcome of Bernoulli test discussed in Section 3.

We assume that attributes come with a natural order (i.e., hierarchy-of-priority). However, this ordering affects the utility of Nested-*Nebula* as attributes towards the end of the clients' submissions are less likely to be learned by the aggregation server. An interesting future direction is to optimize the ordering of client attributes based on domain knowledge (e.g. the distribution of data itself) to achieve high utility.

We also note that the utility improvement of this multi-dimensional encoding might come at a cost in terms of privacy. An aggregation server with perfect background knowledge (full knowledge of all records in D, and full knowledge of the victim record), aiming to infer whether the victim record is in the input dataset or not, can recover some information. In particular, the aggregation server can observe tags of unrevealed prefixes $\widehat{\text{sbm}}_i^{(\ell)}$ whose immediate preceding prefixes $\widehat{\text{sbm}}_i^{(\ell-1)}$ are decoded. However, it is common to ignore this leakage in practice [41] as the above privacy leakage happens for pathological datasets with extremely skewed distribution and mostly binary values. Several ways have been proposed to address this leakage in the literature, such as relaxing the definition of differential privacy by considering practical data distributions [42, 43, 44, 45, 41, 35].

6 Experiments

We implement *Nebula* (https://github.com/brave-experiments/Nebula-CCS2025) and empirically validate: **i) Effectiveness in estimating accurate but private histograms**: in complement to the analytical privacy and utility guarantees of Section 4, we empirically demonstrate that the histogram estimated by *Nebula* is close to the true histogram constructed from all clients' original data while ensuring strong privacy guarantees. **ii) Efficiency in private and secure data collection**: In complement to the analytical communication costs of Section 4, we empirically demonstrate the ability of *Nebula* to scale to real-world use cases thanks to its low computational, bandwidth and financial costs.

We assess the performance of *Nebula* on real-world three datasets. Two of these are *privacy-sensitive in nature*—the IPUMS Census dataset and the Foursquare dataset [46]. We also assess the performance of *Nebula* on the Complete Works of Shakespeare as it is commonly used in histogram estimation literature [8].

IPUMS. We use the Integrated Public Use Microdata Series of United States census data (https://usa.ipums.org/usa/). We consider 15,537,785 data points representing persons through 5 attributes: SEX, marriage status (MARST), RACE, education (EDUC), AGE.

Foursquare dataset is derived from the mobile app "Foursquare City Guide", which takes advantage of a user's location to guide them to highly-rated places like restaurants and bars, while a social networking feature lets the user's friends know what places they visit. The dataset contains 33,263,633 check-in events

at 3,680,126 venues (in 415 cities in 77 countries). Each venue in the dataset (e.g. a restaurant) comes with a latitude and longitude granular enough to identify it uniquely. We pre-process the dataset by extracting the country code and latitude/longitude pairs of each check-in event. The result is a CSV file of 33,263,633 rows where each line contains the location information for one venue visit by one client.

Shakespeare dataset. We also consider the complete works of William Shakespeare,[7] as if clients were each contributing an individual word to generate a frequency distribution. We split the text on whitespace, and apply basic normalization of punctuation and capitalization. This results in a sequence of 832,301 values out of a set of 29,257 unique words. Note that the frequency distribution is highly peaked, in part because no stop words were removed. To study the effect of domain size, we also sort words into bins based on the lower b bits of their SHA256 hash, and apply the same algorithms to the bin index, creating a deterministic mapping consistent with what clients could perform before submission.

Evaluation metrics. We evaluate the effectiveness and efficiency of *Nebula* as follows. **Effectiveness**: We measure utility as a (scalar) error that quantifies the difference between the estimated and the true histograms: i) we represent each histogram as a vector, where the length of the vector corresponds to the total number of bins, and each entry represents the count in the corresponding bin; ii) we normalize each histogram by dividing each bin counts by the total counts; iii) we compute the l_1 norm of the difference between the estimated and original normalized histograms. **Efficiency**: We evaluate the various costs of our framework through (1) Computational costs measured as CPU running time for both the client-side encoding step and the server-side aggregation step; (2) Financial costs based on those running times and per-CPU-hour server rental prices, and (3) Bandwidth costs by measuring the size of a submission to the aggregation and randomness servers.

Parameters. As discussed in Section 4, *Nebula*'s privacy budget is computed as $\varepsilon = \max(\varepsilon_{\text{Unre}}, \varepsilon_{\text{Re}})$ and $\delta = \max(\delta_{\text{Unre}}, \delta_{\text{Re}})$. We set the parameters of *Nebula*—threshold τ, sampling rate p_s and shift t in TDSLap—such that we obtain a desired (ε, δ) privacy guarantees and a trade-off between utility and communication costs as follows: (1) Set $\varepsilon_{\text{Re}} = \varepsilon \leq 1$ (smaller ε provides a stronger privacy guarantee) and $0 < \alpha \leq 1$ to a desired privacy budget and a constant, respectively, and compute the sampling rate as $p_s = \alpha(1-e^{-\varepsilon})$; (2) Set $\delta_{\text{Re}} = \delta$ to be very small (less than the reciprocal of the total number of clients) and compute the threshold as $\tau = \frac{1}{C_\alpha} \ln(\frac{1}{\delta})$ where $C_\alpha = \ln(\frac{1}{\alpha}) - \frac{1}{1+\alpha}$; and (3) set $t = 2 + 2/\varepsilon_{\text{Unre}} \log(2/\delta_{\text{Unre}}))$ such that $\varepsilon_{\text{Unre}} \leq \varepsilon_{\text{Re}}$ and $\delta_{\text{Unre}} \leq \delta_{\text{Re}}$. In particular, setting $\varepsilon = 1$, $\alpha = 1/6$ and $\delta = 10^{-8}$ yields $p_s = 0.105$, $\tau = 20$ and $t = 15$.

Implementation of cryptographic primitives. We use the Adept Secret Sharing framework, implemented in v0.2.3 of the "adds" crate. For randomness stretching / OPRF, we use the Puncturable Partially Oblivious Pseudorandom Function algorithm, implemented with v0.4.1 of the "pporpf" crate. Finally, for Symmetric cryptography and hash functions, we pull from the Strobe protocol framework, implemented in v0.10.0 of the "strobe-rs" crate.

6.1 Utility Comparison to Existing Works

We evaluate our DP data collection framework, *Nebula*, against existing DP methods, and (non-DP) threshold-aggregation systems. Specifically, we compare *Nebula* to four baselines: **Local DP based on the generalized randomized response mechanism** [48]: Dividing the total privacy budget ε evenly across all attributes, each attribute is locally perturbed by the client using randomized response under the corresponding per-attribute budget, and the final report aggregates these noisy responses. **Shuffle DP via multi-message Bernoulli noise addition** [47]: Each client reports a value 1 for their true bin value, and independently reports a value 1 for each bin of the histogram with probability $p_{\text{Shuffle}} = 1 - \frac{50}{\varepsilon^2 N} \ln(2/\delta)$. A shuffler collects the messages, concatenates and randomly permutes them, and forwards the shuffled message to the aggregation server. The aggregator server can then recover an unbiased estimate of the histogram. **Central DP via Laplace noise addition [16]**: the aggregation server collects the raw client's data, computes the true histogram and adds Laplace noise to each bin). **STAR** [7] with K-anonymity protection (no DP): Each client encodes their value as K-out-of-N secret

[7] Plain text edition from https://cs.stanford.edu/people/karpathy/char-rnn/shakespeare_input.txt

Table 1: Comparison of the utility of *Nebula* against: i) DP approaches–including all local, shuffle and central models– under a fixed DP guarantee of $\varepsilon = 1$; and ii) the threshold-aggregation technique STAR implementing a K-anonymity privacy protection ($K = 20$). Utility is assessed as the error between the estimated and true histograms (↓: the lower, the better) on Shakespeare (Shak.), IPUMS and Foursquare (Fours.) datasets. *Nebula* enforces differential privacy (unlike STAR) and achieves lower error than both local and shuffle DP methods. It brings utility closer to that of central DP, without requiring trust in a central server.

Approaches	DP	Utility (↓)		
		Shak.	IPUMS	Fours.
Secure threshold-aggregation (STAR [7])	✗	0.1934	0.0314	0.6217
Central (Laplace noise [16])	✓	0.0348	0.0044	0.0005
Shuffle (Multi-message Bernoulli noise [47])	✓	1.2910	0.4344	1.4184
Local (General-Randomized Response [48])	✓	1.5921	1.6622	2.0000
Nebula	✓	0.5772	0.1932	1.3999

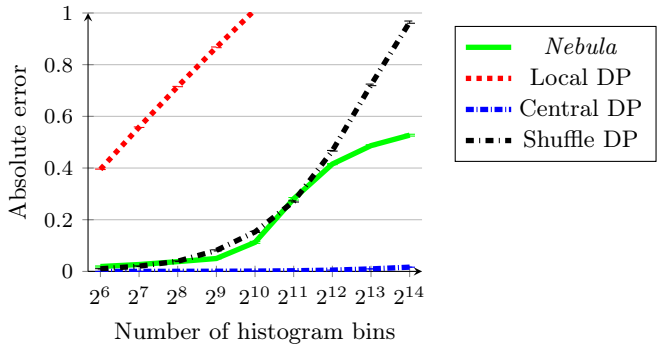

Figure 3: Utility of *Nebula* compared with local, Shuffle and central differential privacy applied to the Shakespeare database as a function of histogram bins using an $\varepsilon = 1$ DP privacy guarantee. The word-frequency estimate of *Nebula* is more accurate than local and shuffle DP while while removing the trust of the central DP models on the server.

shares and sends to the aggregation server (without Poisson sampling and dummy data that *Nebula* introduces for achieving DP guarantees). The aggregation server decrypts values submitted by at least K clients.

Table 1 shows the histogram estimation error of *Nebula*, local DP, Shuffle DP, central DP and STAR on the three datasets. Results demonstrate that *Nebula* is more effective than the alternative local DP and shuffle DP approaches in the collection of high-utility data with strong DP guarantees: the utility of *Nebula* is closer to the utility of the central model of DP in which clients must trust the server. The absolute error in Shakespeare is the lowest. This is because Shakespeare's domain size is smaller than those of the other two datasets, IPUMS and Foursquare. In these datasets, a value represents a combination of multiple attributes, causing the domain size to grow exponentially with the number of attributes.

We further analyze the effect of the domain size in Figure 3. Words from the Shakespeare dataset are mapped by hash value into between 64 and 16384 bins. The smaller the number of bins, the lower the absolute error. As the domain size shrinks, the error trends toward that of the central DP method, consistent with our above explanation of the performance difference between datasets. Across all domain sizes, *Nebula* consistently achieves significantly lower absolute error in the estimated histogram compared to local DP. As the domain size approaches the true domain of the dataset, the utility advantage of *Nebula* over Shuffle DP becomes increasingly evident. Additionally, *Nebula* is highly communication-efficient: each client transmits information solely about its held item, while Shuffle DP produces a message for each possible value. The domain-independent communication complexity of *Nebula* is especially beneficial in

large domains [8].

Conversely, the absolute error in Foursquare is very high: this is because Foursquare consists of multi-attribute data, resulting in a large domain consisting of specific geographic coordinates. Next, we discuss a variant of *Nebula* that can decrease the error in multi-attribute cases such as Foursquare and IPUMS census.

6.2 Utility Improvements via Nested-*Nebula*

We analyze the impact of multi-dimensional encoding on the utility of *Nebula* in estimating the marginal histogram across both multi-dimensional dataset: the IPUMS dataset and the Foursquare dataset. We compare the absolute error of *Nebula* against Nested *Nebula* when estimating the histogram for each prefix.

Figure 4 (first row) demonstrates that multi-dimensional encodings improve the ability of *Nebula* to collect high-utility multi-dimensional IPUMS data. As the number of attributes increases in a prefix, the absolute error of the histogram estimation increases (when including all attributes, we recover the results of Table 1). This is because increasing the number of attributes in a prefix decreases the chance of having more copies of items, amplifying the costs of sampling and pruning on revealing the item at the output to the server: the chance of sampling a low-frequency item decreases and it is more likely the items will be pruned (see Lemma 2).

We now turn to the Foursquare dataset where each element consists of geographic coordinates and a country code, which are not independent attributes. However, the chained prefix encoding can still be applied to improve utility by coarse-graining the venue locations. If each visit is split into the country code and successive digits of the coordinates, a sequence of 8 attributes is produced reporting the event location with increasing granularity. With this encoding, partial recovery of joint attributes amounts to recovering regional aggregate popularity at multiple scales. Figure 4 (second row) shows the improvement in the absolute error using this multi-dimensional nested encoding for the Foursquare dataset. To further demonstrate the effectiveness of *Nebula* and our multi-dimensional nested encoding, we compare the estimated Nested *Nebula* histogram and the true Foursquare histogram of country codes in Figure 5. We observe that Nested *Nebula* preserves the relative frequencies across attribute values. For example, the most popular items stay popular in the estimated histogram.

6.3 *Nebula* is Efficient

We evaluate computational, bandwidth, and financial costs.

Computational costs. We measure the CPU running time of our framework on AWS r6a machine for each client, the aggregation server and the randomness server, separately. Each client performs two sets of computations: (1) obtain randomness by interacting with the randomness server (Algorithm 2); and (2) encode attributes to be submitted to the aggregation server (Algorithm 3). As shown in Table 2, these steps (1) and (2) take 4.96 and 0.85 milliseconds, respectively, for each submission from the Foursquare dataset with the chained-prefix encoding and 8 attributes. Running times are even lower for the IPUMS dataset as each client holds fewer attributes (five). This running time scales proportionately for the Shakespeare dataset, where each client submission consists of a single word without prefix encoding. Per-submission running time is comparable for all three datasets when the whole value is encoded as a single attribute. The running time of the randomness server (last row in Table 2) is very low; as it only needs to perform one OPRF evaluation on each client's request using its secret key. This takes only 0.48 milliseconds for each Foursquare client submission. For simplicity, we omit the zero-knowledge proof steps of the verifiable OPRF in these benchmarks. Table 2 shows the CPU running time of our framework for the aggregation server. It takes 345 seconds for the server to process all 33,263,633 client submissions from the Foursquare dataset. Therefore, ***Nebula* introduces only a very little computational overhead for all parties—clients, the randomness server and the aggregation server**.

Bandwidth costs. Table 4 shows the bandwidth costs that *Nebula* introduces for each client. The total communication costs of running *Nebula* for each client is at most 2.7 KB (not including framing and transport overhead) for the multi-attribute Foursquare encoding, combining the interaction of each

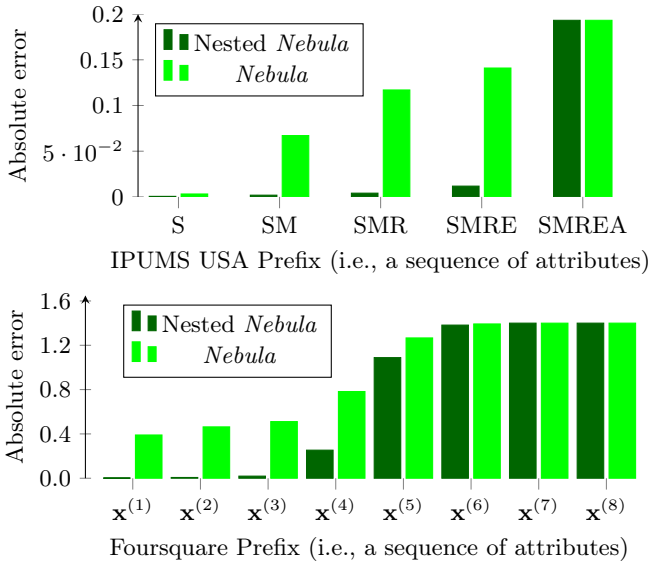

Figure 4: Improving the utility of *Nebula* in estimating the histogram on the multi-attribute IPUMS dataset and Foursquare dataset using multi-dimensional data encoding, Nested *Nebula* (Algorithm 6). IPUMS contains 5 attributes–S: Sex; M: Marriage status; R: Race; E: Education; A: Age. Foursquare dataset contains 8 prefixes: $\mathbf{x}^{(1)} = [x_1]$, $\mathbf{x}^{(2)} = [x_1, x_2]$, $\cdots$, $\mathbf{x}^{(8)} = [x_1, \cdots, x_8]$. We compute the utility as the absolute error between the original histogram and the estimated histogram. Multi-dimensional data encoding significantly improves the absolute error of each marginal histogram (i.e., histogram of each sequence of joint attributes).

Table 2: Efficiency of *Nebula* in terms of running time on Foursquare and IPUMS datasets, in *milliseconds per submission*. **The computational overhead of *Nebula* for clients and the randomness server is very small.**

Party	Function	Dataset		
		Foursquare	IPUMS	Shakespeare
Client	Encode	4.96	3.07	0.42
	Randomness	0.85	0.53	0.21
Server	OPRF evaluation	0.48	0.30	0.06

Table 3: Efficiency of *Nebula* in terms of running time on three datasets, in *seconds for all submissions*. **The computational overhead of *Nebula* for the aggregation server is small.**

Party	Function	Dataset		
		Foursquare	IPUMS	Shakespeare
Server	Decode	345	101	8

Table 4: Efficiency of *Nebula* in terms of bandwidth costs on Foursquare and IPUMS datasets in bytes per submission. **The bandwidth cost of *Nebula* is very small**.

Interaction	Foursquare	IPUMS	Shakespeare
Client&RandomnessServer	256	160	32
Client&AggregationServer	2465	1470	373

client with both the randomness and the aggregation servers. Each client submits about 300 bytes per attribute with some fixed overhead for internal framing combining all communication, with most of that

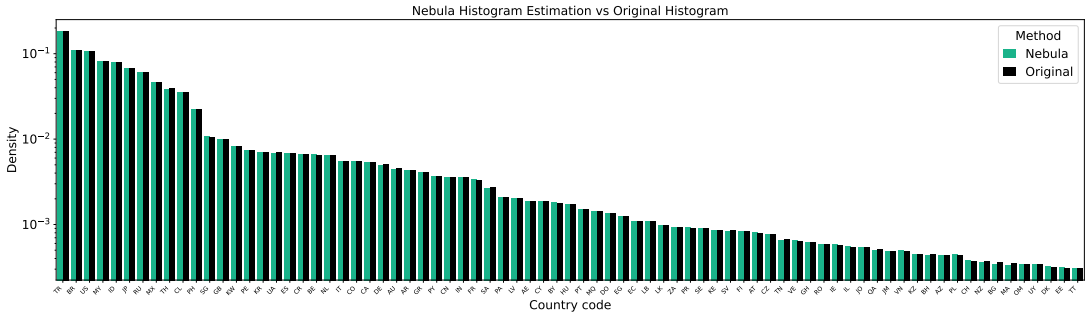

Figure 5: Original and estimated histogram obtained privately by *Nebula* using Foursquare dataset. The private histogram estimated by *Nebula* is close to the histogram of the original data. *Nebula* also preserves the relative order across values.

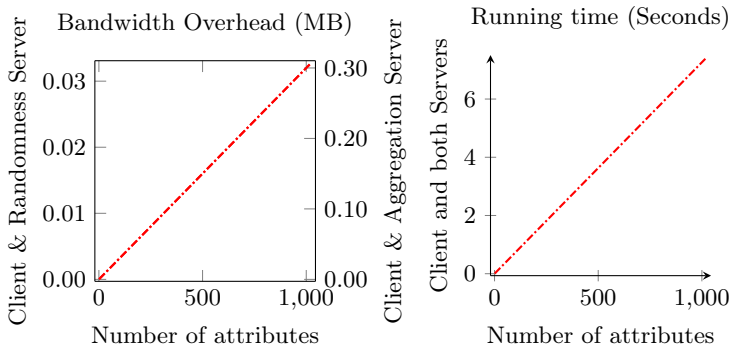

Figure 6: Scalability of *Nebula* in terms of the computational and bandwidth overhead introduced for clients. ***Nebula* scales to a large number of attributes with negligible costs.**

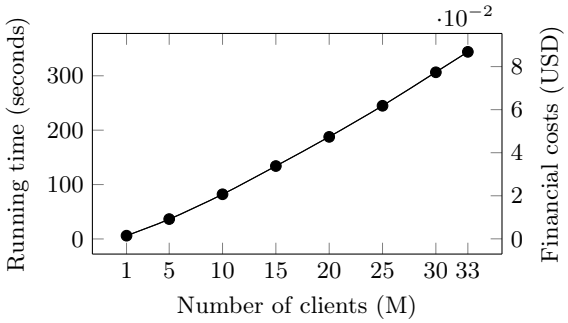

Figure 7: Scalability of *Nebula* in terms of the computational and financial costs for the aggregation server. ***Nebula* scales to a large number of clients with negligible costs.**

traffic going to the aggregation server. See Section 4.3 for a detailed breakdown. Dummy data submitted to hide below-threshold submissions is $t\frac{(\tau-1)\tau}{2}$ (expectation) and $2t\frac{(\tau-1)\tau}{2}$ (worst-case) where t is the expectation of the truncated discrete Laplace distribution and τ is the threshold for pruning values. Given our parameter values $t = 14$ and $\tau = 20$, this amounts to a few thousand extra aggregation server reports which is still quite small in absolute terms, and completely negligible on the server side. Therefore, ***Nebula* introduces very small bandwidth costs for clients**.

Financial costs. We compute the financial costs of running *Nebula* for the aggregation and randomness servers. We benchmarked *Nebula* on an AWS r6a.4xlarge instance, currently priced at US$0.9072 per hour. As such, the amortized cost of aggregating submissions from the IPUMS dataset is 0.03 USD, the

Foursquare dataset 0.09 USD, and the Complete Works of Shakespeare only 0.002 USD.

6.4 *Nebula* is Scalable

We further analyze the scalability of *Nebula* by considering various numbers of attributes using the same hardware described in Section 6.3. Timings for the IPUMS and Foursquare datasets report the more expensive multi-attribute encoding scheme (Algorithm 6), while the Shakespeare dataset does not use multiple attributes and represents the whole-value reporting scheme in general. Figure 6 shows the effect of the number of attributes on the bandwidth and computational costs of clients needed to interact with the randomness server and prepare the submission to the aggregation server. Interaction with the randomness server scales linearly with the number of attributes, since the OPRF must be evaluated separately for each prefix. Bandwidth costs of each client interacting with the aggregation server also scale linearly with the number of attributes. Finally, we observe that the time of *Nebula* run by each client scales linearly with the number of attributes. This is because run time is dominated by the key share and encryption steps which in the multi-attribute scheme need to be done once per attribute to allow partial recovery of multivariate joint attributes.

Finally, we analyzed the scalability of *Nebula* by considering various numbers of clients. Figure 7 shows the running time and financial costs of the aggregation server as a function of number of clients. ***Nebula* can collect multi-dimensional data from a very large number of clients with small costs.**

7 Related work

Threshold-aggregation data collection systems [7, 6, 49, 17, 50, 51, 52, 25, 53, 54] allow a central party to learn submitted values if and only if a predefined number of clients send the exact same value. Poplar [6] uses distributed point functions to create a secret sharing pair of a vector in which only a single element with an index corresponding to the client's data is non-zero. Clients then send their secret shares to *two non-colluding aggregation servers*, which compute the sum of submitted shares and publish the sum values. STAR [7] uses a different τ-out-of-N secret-sharing scheme to avoid the need for two aggregation servers. Clients encode their values as secret shares, and send their shares to a *single aggregation server*, which is to decrypt values that have been encoded by at least τ submitted shares. STAR provides high efficiency and more desirable trust requirements than Poplar, though at the cost of leaking the histogram of unrecovered values (i.e., values submitted by less than τ clients). POPSTAR [12] tries to hide the distribution of unrevealed values in STAR, though in a manner that requires significantly more computation (7x computation, and an estimated 2-3x increase in the required time, compared to the dataset as STAR).

Existing threshold aggregation systems have significant limitations. First, all threshold-aggregation systems, like all deterministic k-anonymity systems, lack robust, provable privacy guarantees. Furthermore, current threshold-aggregation systems (including Poplar and STAR) are only well-suited to handle single-dimension values. Trying to use these systems to handle multi-dimensional records entails significant utility loss. One option is to flatten multi-dimensional records into a single dimension (e.g., concatenation, summation, etc.), and run the system on that "flattened" value. This approach significantly harms utility, since submitted records would need to match across all original dimensions to count towards each record's recovery threshold, decreasing the amount of information the server is able to recover. The second option is to have clients submit each dimension independently, treating a record with three attributes as three independent records. This also harms utility, though in a different way: the aggregation server is unable to learn any relationships or correlations between data attributes.

Our proposed system, *Nebula*, works better than these systems in terms of both utility and privacy by *i)* allowing clients to submit data with multiple attributes such that the utility of marginal histogram estimations is maximized; and *ii)* satisfying strong differential privacy guarantees (through sampling followed by pruning, and dummy data) without trusting servers.

Differentially private data collection systems. Differential Privacy (DP) [16] uses statistical indistinguishably to ensure privacy. DP is typically implemented in one of two forms: first, a *central model* where the aggregation server receives unmodified user data, who then applies privacy protections to the

data before sharing it, and second, a *local model*, where users apply privacy protections to their own data before sharing it with the aggregation server. These approaches achieve different privacy-utility trade offs.

Central DP systems provide high utility, but suffer from often prohibitive trust assumptions (i.e., clients must trust the aggregation server and send their raw data to the server). This is a practical problem, as many servers do not provide the privacy protections they promise (intentionally or otherwise) [55]. Central DP systems also carry the risk of a single point of failure for data breaches [56, 57].

Local DP systems, on the other hand, provide strong privacy guarantees, typically by perturbing data before revealing it to untrusted parties [49, 52]. This greatly improves the privacy and security properties of the system, but at the cost of reducing the utility of the aggregated data. The shuffle model of DP [47] improves the utility by allowing to perturb data with less noise, while trusting an intermediate shuffler to apply a uniform random permutation to all data before the aggregation server views them.

More recent DP proposals attempt to achieve better utility through using multi-party computation or homomorphic encryption to actually reduce the level of trust required in central DP systems [26, 58, 59, 23, 10, 60, 56, 9]. For example, Bell et.al., [9] present a protocol for computing DP histograms carried out between two non-colluding aggregation servers. In each round of server-to-server communication, each server injects carefully crafted dummy data into its message, ensuring that the additional leakage revealed to other server beyond the output remains differentially private–a DP anonymized histogram of indices to one server and a DP anonymized histogram of values to other server. But these proposals suffer from their own drawbacks, including *i)* requiring networks of non-colluding servers, a majority of whom are assumed to behave honestly [10], *ii)* imposing high computation and communication overheads costs [23], *iii)* only being suited for simple aggregation functions [60, 58, 26, 59], and *iv)* requiring interactive communication between clients and servers. Requiring interactions between the servers (more than 1 aggregation server) makes it more difficult to guarantee the non-collusion requirement and it may also have practical costs (e.g., it may be harder to recruit collaborative partners). Finally, most DP systems are also limited to one-dimensional data, limiting their utility or applicability to many scenarios.

Our system, *Nebula*, obtains better utility than both local DP randomized and shuffling[8] as: *i)* the utility error of *Nebula* is independent of the number of attributes as opposed to local DP randomizers in which the noise grows significantly as the number of attributes increases; *ii)* in contrast to existing DP systems, *Nebula* does not add explicit noise, thus introducing no spurious attribute values. In addition to this, our system avoids prohibitive trust assumptions required in the central model deployment of DP while being efficient because of not requiring expensive multi-party computations and homomorphic encryption operations.

Trusted hardware. Finally, a third-general approach to private data collection uses trusted hardware to enforce privacy guarantees. For example, Prochlo [25] uses trusted hardware to collect unmodified data from clients. This trusted hardware is able to collect, shuffle, and modify user data *before* privacy protections are applied to the data. Once these trusted servers have received sufficient data, it is modified and passed onto untrusted hardware, which does the primary data summarizing and aggregation. Trusted hardware carries a wide range of downsides and limitations though, including relatively high cost, resource limitations (in some cases), and (in many cases) merely re-shuffled trust requirements. Approaches like mix-nets [53] and verifiable shuffling [54] can provide security and privacy guarantees similar to (but without requiring) trusted hardware, though at the cost of increased interactivity.

8 Discussion and Future work

In this paper, we proposed *Nebula* that can be used to privately estimate and publish histogram of data generated by clients. *Nebula* introduces necessary randomness for the privacy protection of clients through sampling, thresholding, and dummy data injection, and removes the trust assumption on the server through a customized secret-sharing protocol. Incorporating synergies and optimizations on both fronts of sample-and-threshold differential privacy and secure threshold aggregation enables *Nebula* to provide high utility without imposing prohibitive trust requirements, relying on computationally expensive

[8] Bharadwaj and Cormode [8] analytically and empirically demonstrated the superior performance of the sample-and-threshold compared to the shuffle model of DP.

cryptographic operations, or requiring bandwidth-intensive multi-round communications between clients and servers. We analytically and empirically demonstrated that *Nebula* is effective, efficient and scalable.

We conclude by discussing some limitations of our approach and by outlining promising directions for future work. As discussed in Section 2, our threat model, like those commonly found in the literature, assumes honest-but-curious clients. An interesting future work is to extend *Nebula* to defend against malicious clients who either deviate from the protocol or collude with one or both servers to compromise the privacy of honest clients. Another interesting direction involves designing efficient protocols for the verifiable selection of a single client (or a subset of clients) to generate and submit dummy data, as outlined in Section 3.3.

Acknowledgements. We thanks Olive Franzese, Sofía Celi, and Alex Davidson for useful comments and feedback.

References

[1] H. Corrigan-Gibbs, D. Boneh, G. Chen, S. Englehardt, R. Helmer, C. Hutten-Czapski, A. Miyaguchi, E. Rescorla, and P. Saint-Andre, "Privacy-preserving firefox telemetry with prio," 2020, https://rwc.iacr.org/2020/slides/Gibbs.pdf.

[2] D. Bogdanov, M. Jõemets, S. Siim, and M. Vaht, "Privacy-preserving tax fraud detection in the cloud with realistic data volumes," *T-4-24, Cybernetica AS*, 2016.

[3] J. Andrew, R. J. Eunice, and J. Karthikeyan, "An anonymization-based privacy-preserving data collection protocol for digital health data," *Frontiers in Public Health*, 2023.

[4] A. Bharadwaj and G. Cormode, "Federated computation: a survey of concepts and challenges," *Distributed and Parallel Databases*, 2023.

[5] K. Chadha, J. Chen, J. Duchi, V. Feldman, H. Hashemi, O. Javidbakht, A. McMillan, and K. Talwar, "Differentially private heavy hitter detection using federated analytics," in *Secure and Trustworthy Machine Learning (SaTML)*, 2023.

[6] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," in *Symposium on Security and Privacy (SP)*, 2021.

[7] A. Davidson, P. Snyder, E. V. Quirk, J. Genereux, B. Livshits, and H. Haddadi, "Star: Distributed secret sharing for private threshold aggregation," in *Computer and Communications Security (CCS)*, 2022.

[8] G. Cormode and A. Bharadwaj, "Sample-and-threshold differential privacy: Histograms and applications," in *Artificial Intelligence and Statistics (AISTATS)*, 2022.

[9] J. Bell, A. Gascon, B. Ghazi, R. Kumar, P. Manurangsi, M. Raykova, and P. Schoppmann, "Distributed, private, sparse histograms in the two-server model," in *Computer and Communications Security (CCS)*, 2022.

[10] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *Networked Systems Design and Implementation (NSDI)*, 2017.

[11] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Computer and Communications Security (CCS)*, 2014.

[12] H. Li, S. Navot, and S. Tessaro, "Popstar: Lightweight threshold reporting with reduced leakage," in *USENIX Security Symposium (USENIX)*, 2024.

[13] A. Narayanan and V. Shmatikov, "How to break anonymity of the netflix prize dataset," *arXiv*, 2006.

[14] N. Holohan, S. Antonatos, S. Braghin, and P. M. Aonghusa, "(k, ϵ)-anonymity: k-anonymity with ϵ-differential privacy," *arXiv*, 2011.

[15] C. Dwork, "Differential privacy," in *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2006.

[16] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, 2014.

[17] R. Bassily and A. Smith, "Local, private, efficient protocols for succinct histograms," in *Symposium on Theory of Computing (STOC)*, 2015.

[18] T. Wang, J. Blocki, N. Li, and S. Jha, "Locally differentially private protocols for frequency estimation," in *USENIX Security Symposium (USENIX)*, 2017.

[19] J. Acharya, Z. Sun, and H. Zhang, "Hadamard response: Estimating distributions privately, efficiently, and with little communication," in *Artificial Intelligence and Statistics (AISTATS)*, 2019.

[20] N. Wang, X. Xiao, Y. Yang, J. Zhao, S. C. Hui, H. Shin, J. Shin, and G. Yu, "Collecting and analyzing multidimensional data with local differential privacy," in *International Conference on Data Engineering (ICDE)*, 2019.

[21] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett, "Differentially private histogram publication," *The VLDB journal*, 2013.

[22] B. Balle, J. Bell, A. Gascón, and K. Nissim, "The privacy blanket of the shuffle model," in *Advances in Cryptography (CRYPTO)*, 2019.

[23] A. Cheu, A. Smith, J. Ullman, D. Zeber, and M. Zhilyaev, "Distributed differential privacy via shuffling," in *Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2019.

[24] Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta, "Amplification by shuffling: From local to central differential privacy via anonymity," in *Symposium on Discrete Algorithms (SODA)*, 2019.

[25] A. Bittau, Úlfar Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *Symposium on Operating Systems Principles (SOSP)*, 2017.

[26] J. Böhler and F. Kerschbaum, "Secure multi-party computation of differentially private median," in *USENIX Security Symposium (USENIX)*, 2020.

[27] M. Lepinksi, S. Micali, and A. Shelat, "Collusion-free protocols," in *Proceedings of the 37th annual ACM symposium on Theory of computing*, 2005.

[28] J. Alwen, A. Shelat, and I. Visconti, "Collusion-free protocols in the mediated model," in *International Cryptology Conference (CRYPTO)*, 2008.

[29] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation," *Cryptology ePrint Archive*, 2011.

[30] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan, "Computational differential privacy," in *International Cryptology Conference (CRYPTO)*, 2009.

[31] M. Bellare, W. Dai, and P. Rogaway, "Reimagining secret sharing: Creating a safer and more versatile primitive by adding authenticity, correcting errors, and reducing randomness requirements," in *Privacy Enhancing Technologies (PETS)*, 2020.

[32] N. Tyagi, S. Celi, T. Ristenpart, N. Sullivan, S. Tessaro, and C. A. Wood, "A fast and simple partially oblivious prf, with applications," in *Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2022.

[33] M. R. Albrecht, A. Davidson, A. Deo, and N. P. Smart, "Round-optimal verifiable oblivious pseudorandom functions from ideal lattices," in *Practice and Theory of Public-Key Cryptography (PKC)*, 2021.

[34] S. Jarecki, A. Kiayias, and H. Krawczyk, "Round-optimal password-protected secret sharing and t-pake in the password-pnly model," in *Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2014.

[35] N. Li, W. H. Qardaji, and D. Su, "Provably private data anonymization: Or, k-anonymity meets differential privacy," *arXiv*, 2011.

[36] M. Thomson and C. A. Wood, "Oblivious http," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-ohai-ohttp-05, 2022.

[37] Z. Zhang, T. Wang, N. Li, S. He, and J. Chen, "Calm: Consistent adaptive local marginal for marginal release under local differential privacy," in *Computer and Communications Security (CCS)*, 2018.

[38] T. Wang, B. Ding, J. Zhou, C. Hong, Z. Huang, N. Li, and S. Jha, "Answering multi-dimensional analytical queries under local differential privacy," in *International Conference on Management of Data (ICDAM)*, 2019.

[39] D. J. Leith, "Mobile handset privacy: Measuring the data ios and android send to apple and google," in *Security and Privacy in Communication Networks (SecureComm)*, 2021.

[40] K. Satvat and N. Saxena, "Crashing privacy: An autopsy of a web browser's leaked crash reports," *arXiv*, 2018.

[41] D. Desfontaines and B. Pejó, "Sok: Differential privacies," in *Privacy Enhancing Technologies (PETS)*, 2020.

[42] R. Bassily, A. Groce, J. Katz, and A. Smith, "Coupled-worlds privacy: Exploiting adversarial uncertainty in statistical data privacy," in *Foundations of Computer Science (FOCS)*, 2013.

[43] Y. Duan, "Privacy without noise," in *Information and Knowledge Management (CIKM)*, 2009.

[44] D. Kifer and A. Machanavajjhala, "A rigorous and customizable framework for privacy," in *Principles of Database Systems (PODS)*, 2012.

[45] R. Bhaskar, A. Bhowmick, V. Goyal, S. Laxman, and A. Thakurta, "Noiseless database privacy," in *Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2011.

[46] D. Yang, D. Zhang, and B. Qu, "Participatory cultural mapping based on collective behavior data in location-based social networks," *Transactions on Intelligent Systems and Technology (TIST)*, 2016.

[47] V. Balcer and A. Cheu, "Separating local & shuffled differential privacy via histograms," in *Information-Theoretic Cryptography (ITC)*, 2019.

[48] P. Kairouz, S. Oh, and P. Viswanath, "Extremal mechanisms for local differential privacy," *Journal of Machine Learning Research (JMLR)*, 2017.

[49] R. Bassily, K. Nissim, U. Stemmer, and A. Thakurta, "Practical locally private heavy hitters," *Journal of Machine Learning Research (JMLR)*, 2020.

[50] M. Bun, J. Nelson, and U. Stemmer, "Heavy hitters and the structure of local privacy," *Transactions on Algorithms (TALG)*, 2019.

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

[52] W. Zhu, P. Kairouz, B. McMahan, H. Sun, and W. Li, "Federated heavy hitters discovery with differential privacy," in *Artificial Intelligence and Statistics (AISTATS)*, 2020.

[53] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, p. 84–90, Feb. 1981.

[54] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *Computer and Communications Security (CCS)*, 2001.

[55] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang, "Privacy loss in apple's implementation of differential privacy on macos 10.12," *arXiv*, 2017.

[56] A. R. Chowdhury, C. Wang, X. He, A. Machanavajjhala, and S. Jha, "Cryptϵ: Crypto-assisted differential privacy on untrusted servers," in *International Conference on Management of Data (SIGMOD)*, 2020.

[57] T. Venturini and R. Rogers, ""api-based research" or how can digital sociology and journalism studies learn from the facebook and cambridge analytica data breach," *Digital Journalism*, 2019.

[58] J. Boehler and F. Kerschbaum, "Secure sublinear time differentially private median computation," 2022, uS Patent 11,238,167.

[59] M. Pettai and P. Laud, "Combining differential privacy and secure multiparty computation," in *Annual Computer Security Applications Conference (ACSAC)*, 2015.

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

[61] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," Cryptology ePrint Archive, 2000.

A Security Proof

Formal Security Model. Aggregation Server S_A may be corrupted by a *malicious* adversary with arbitrary behavior. It is assumed not to collude with the randomness server or any clients; Randomness Server S_R may be corrupted by a *malicious* adversary with arbitrary behavior. It is assumed not to collude with S_A or any clients; Clients $\{C_i\}_{i=1}^N$ may be corrupted by a *semi-honest* adversary. Corrupted clients follow the protocol, but seek to learn information about other parties. They are assumed not to collude with other parties.

Nebula effectively employs a protocol for secure computation of private threshold aggregation reporting (STAR [7]) as a subroutine, with additional elements for guaranteeing differential privacy over its outputs. To reason about the security of *Nebula*, we will begin by recalling the elements that are the same between the two protocols.

In particular, Algorithm 2 plus lines 1-2 of Algorithm 3 are equivalent to the Randomness Phase of [7] with a VOPRF as instantiated in [33], assuming the hash function H is a secure random oracle. The remainder of Algorithm 3 is equivalent to the Message Phase of [7] for the subset of clients which are selected by the local Bernoulli tests in lines 6-8 of Algorithm 1, and for the clients that are not selected it is equivalent to aborting immediately before the Message Phase (this is tolerated trivially by the security proofs in [7]). Algorithm 7 similarly replicates the Randomness Phase and Message Phase of [7] in lines 7 and 8 respectively, followed by submission to S_A in Algorithm 1 line 10. Finally, Algorithm 5 is equivalent to the Aggregation Phase of [7].

Algorithm 7: *Modified DummyDataCreation*: Create groups of dummy data

Input: Threshold τ, Truncated Shifted Discrete Laplace distribution TSDLap$(\cdot)$, DP guarantees $(\varepsilon_{\text{Unre}}, \delta_{\text{Unre}})$

Output: A set of dummy data

```
Dummy = {}
Select a client for creating dummy data
for i = 1, ..., τ − 1 do
    α ← TSDLap(λ = 2/ε_Unre, t = 2 + 2/ε_Unre log(2/δ_Unre))
    for j ∈ α do
        x_j ← DummyObservation()
        r_j = ClientRandomnessServer(x_j, pp, H(·))
        sbm_j, _ ← LocalSecretSharing(x_j, r_j, Π_{τ,N})
        Dummy.append({(sbm_j)^i})
Return Dummy
```

These elements can be considered an implementation of the protocol from [7], realizing the functionality $\mathcal{F}_{\text{STAR}}$ which we recapitulate in Figure 8. We make one addition, formalizing the fact that their protocol enables clients to participate in the Randomness Phase and then abort before submitting data in the Message Phase. We use these observations to prove the security of *Nebula* based on the Universal Composition paradigm [61], a standard cryptographic proof technique. Before we proceed to the proof, we will recall elements of STAR and *Nebula* that are *dissimilar*.

The Bernoulli tests in lines 6-8 of Algorithm 1 are not present in [7] and neither are lines 9-11 of Algorithm 1, which call Algorithm 4 as a subroutine. As previously mentioned, the former is trivially tolerated by [7]. The latter is equivalent to increasing the number of client inputs received by the STAR functionality, which are chosen so that they reveal no information about the input data of any client. Thus our protocol for *Nebula* can be rewritten as: i) S_R samples a VOPRF keypair (msk, mpk); ii) Clients $\{C_i\}_{i=1}^N$ sample $b_i \leftarrow \text{Bern}(p_s)$; iii) A randomly chosen client C^* samples $\{\alpha_i\}_{i=1}^{\tau-1}$ where each α_i is sampled independently from $\alpha_i \leftarrow \text{TSDLap}(\lambda = 2/\varepsilon_{\text{Unre}}, t = 2 + 2/\varepsilon_{\text{Unre}} \log(2/\delta_{\text{Unre}}))$; iv) For each α_i, C^* constructs α_i groups of dummy clients of size i with input $(\omega_{i,j}, \text{aux}_{i,j})$ for all $j \in \alpha_i$. Each $\omega_{i,j}$ is a distinct measurement outside the set of client measurements $\{x_i\}_{i=1}^N$. Call the set of all dummy clients $\mathcal{D}$; v) S_A, S_R, and client pool $\{C_i\}_{i=1}^N \cup \mathcal{D}$ call $\mathcal{F}_{\text{STAR}}$ using their respective inputs from above. Assume additional leakage of the cardinality of each group of inputs $n_\ell \equiv |S_\ell|$ where S_ℓ is the set of submissions from $\mathcal{C} \cup \mathcal{D}$ which share the same unique measurement x_ℓ, as in the protocol from [7].

We use this formulation of our protocol to demonstrate that *Nebula* realizes the ideal functionality $\mathcal{F}_{\text{Nebula}}$ shown in Figure 2. This augmented functionality ensures that the output histogram has the DP guarantees proven in Section 4. Note that we also output to S_A the cardinality of each group of inputs provided by $\mathcal{C} \cup \mathcal{D}$ which shares a unique measurement x_ℓ. This is also included in STAR, but excluded from the ideal functionality and formalized as a leakage function $\mathcal{L}$. We include it as an output of $\mathcal{F}_{\text{Nebula}}$ since the method is designed specifically to make this leakage uninformative using differential privacy.

Now ready to prove the security and correctness of our protocol.

Theorem 3. *(Correctness.) The Nebula protocol is correct with all but negligible probability.*

Proof. Clients perform Bernoulli tests to decide whether to submit their real data (Algorithm 1), and craft and submit dummy data (Algorithm 7). Therefore, the set of data received by the aggregation server in the protocol is equivalent to the inputs specified in $\mathcal{F}_{\text{Nebula}}$. The remaining parts of the protocol reduce to performing the STAR protocol on this modified set of data (Poisson subsampled real data joint with dummy data), resulting in the aggregation server learning only a DP histogram of revealed submissions and a DP histogram of unrevealed submissions as proven in our Theorem 1. Thus the rest of the proof reduces to the correctness proof for STAR, which demonstrates correctness except with negligible probability ([7] Theorem 2). Therefore, *Nebula* correctly produces the outputs shown in $\mathcal{F}_{\text{Nebula}}$ (i.e., revealing only the expected DP histogram to the aggregation server as shown in Figure 2). □

Ideal Functionality $\mathcal{F}_{\text{STAR}}$

Participants:

- Aggregation server S_A
- Randomness server S_R
- Clients $\{C_i\}_{i=1}^N$

Public parameters:

- Threshold τ

Inputs:

- S_R: provides VOPRF keypair (msk, mpk).
- Client $C_i \in \{C_i\}_{i=1}^N$ provides input (x_i, aux_i) and a bit b_i to indicate protocol abort immediately after the Randomness Phase.
- S_A has no inputs, provides $\perp$

Functionality:

1. For each unique x_ℓ construct:

$$\mathcal{E}_\ell = \{(x_\ell, \{\text{aux}_j\}_{j \in J}, \tau_\ell) : (J \subseteq [N]) \wedge (x_j = x_\ell) \wedge (b_j = 1)\}$$

   where $\tau_\ell = |\{\text{x}_j\}_{j \in J}|$ is the number of sampled client measurements in $\mathcal{E}_\ell$.

2. Let $\mathcal{Y}$ be an empty map.
3. For each $\mathcal{E}_\ell$ where $\tau_\ell \geq \tau$, set $\mathcal{Y}[x_\ell] = \mathcal{E}_\ell$.

Outputs:

- output $\mathcal{Y}$ to S_A
- output $\{\mathcal{F}_\Gamma(\text{msk}, x_i)\}_{i=1}^N$ to S_R, where $\mathcal{F}_\Gamma$ is an ideal functionality for VOPRF protocol Γ.
- output $\perp$ to each $\{C_i\}_{i=1}^N$

Figure 8: Ideal functionality for STAR [7].

Theorem 4. *(Malicious aggregation server.) The Nebula protocol is secure against any $\mathcal{A}$ that corrupts S_A, assuming a secure protocol which realizes $\mathcal{F}_{STAR}$.*

Proof. Given any $\mathcal{A}$, we define a simulator $\mathcal{S}$ which interacts with $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\text{Nebula}}$ as follows: $\mathcal{S}$ submits $\perp$ to $\mathcal{F}_{\text{Nebula}}$; $\mathcal{S}$ receives from $\mathcal{F}_{\text{Nebula}}$ the histogram $\mathcal{Y}$, and cardinality of each group of inputs n_ℓ for every unique measurement x_ℓ; $\mathcal{S}$ simulates $\mathcal{F}_{\text{STAR}}$: when $\mathcal{A}$ submits $\perp$ to $\mathcal{F}_{\text{STAR}}$, $\mathcal{S}$ sends $\mathcal{Y}$ back; $\mathcal{S}$ simulates the leakage $\mathcal{L}$ by submitting each n_ℓ to $\mathcal{A}$.

$\mathcal{S}$ simulates the view of $\mathcal{A}$. Uncorrupted clients correctly perform Poisson sampling and dummy data injection locally in the real-world execution, resulting in a set of outputs to $\mathcal{F}_{\text{STAR}}$ which is sampled from the same distribution as the outputs of $\mathcal{F}_{\text{Nebula}}$. Since our security model specifies that clients do not collude with S_A, we can assume all clients are uncorrupted when $\mathcal{A}$ corrupts S_A. So the simulated and real-world views are statistically indistinguishable. □

Operations conducted between the clients and the randomness server S_R are the same as in STAR [7].

Theorem 5. *(Malicious randomness server.) The Nebula protocol is secure against any adversary $\mathcal{A}$ that corrupts S_R, assuming a secure protocol which realizes $\mathcal{F}_{STAR}$*

Proof. Given any $\mathcal{A}$, we define a simulator $\mathcal{S}$ which interacts with $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\text{Nebula}}$ as follows: simulating $\mathcal{F}_{\text{STAR}}$, $\mathcal{S}$ receives inputs for (msk, mpk) from $\mathcal{A}$; $\mathcal{S}$ submits (msk, mpk) to $\mathcal{F}_{\text{Nebula}}$ and receives $\{\mathcal{F}_{\text{VOPRF}}(\text{msk}, x_i)\}_{i=1}^{N}$; $\mathcal{S}$ uses (msk, mpk) to simulate $\mathcal{F}_{\text{VOPRF}}(\text{msk}, x_j)$ for all dummy clients in $\mathcal{D}$. Call this set of outputs W; $\mathcal{S}$ sends $\{\mathcal{F}_{\text{VOPRF}}(\text{msk}, x_i)\}_{i=1}^{N} \cup W$ to $\mathcal{A}$ as the output of $\mathcal{F}_{\text{STAR}}$. This simulates the view of $\mathcal{A}$. □

Theorem 6. *(Semi-honest corrupted clients.) The Nebula protocol is secure against any adversary $\mathcal{A}$ that corrupts a subset of clients $T \subset \{C_i\}_{i=1}^{N}$, assuming a secure protocol which realizes $\mathcal{F}_{STAR}$.*

Proof. The simulator takes inputs submitted to $\mathcal{F}_{\text{STAR}}$ and submits them to $\mathcal{F}_{\text{Nebula}}$. It then returns $\perp$ to each client in T. This trivially simulates the view of $\mathcal{A}$.

To show that the joint distribution over inputs and outputs is statistically indistinguishable for real and ideal world execution, we recall that clients are corrupted only by semi-honest adversaries in our security model. Thus we can assume that all corrupted clients in T follow the protocol faithfully. This means that the clients perform Poisson sampling and dummy data injection correctly even if they are corrupted in the real-world execution. Thus, the joint distributions in the real and ideal world executions are the same. □