Nebula: Efficient, Private and Accurate Histogram Estimation

Ali Shahin Shamsabadi, Peter Snyder, Ralph Giles, Aurélien Bellet, Hamed Haddadi

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

September 17, 2024

Abstract

We present *Nebula*, a system for differential private histogram estimation of data distributed among clients. *Nebula* enables clients to locally subsample and encode their data such that an untrusted server learns only data values that meet an aggregation threshold to satisfy differential privacy guarantees. Compared with other private histogram estimation systems, *Nebula* uniquely achieves all of the following: *i)* a strict upper bound on privacy leakage; *ii)* client privacy under realistic trust assumptions; *iii)* significantly better utility compared to standard local differential privacy systems; and *iv)* avoiding trusted third-parties, multi-party computation, or trusted hardware. We provide both a formal evaluation of *Nebula*'s privacy, utility and efficiency guarantees, along with an empirical evaluation on three real-world datasets. We demonstrate that clients can encode and upload their data efficiently (only 0.0058 seconds running time and 0.0027 MB data communication) and privately (strong differential privacy guarantees $\varepsilon = 1$). On the United States Census dataset, the *Nebula*'s untrusted aggregation server estimates histograms with above 88% better utility than the existing local deployment of differential privacy. Additionally, we describe a variant that allows clients to submit multi-dimensional data, with similar privacy, utility, and performance. Finally, we provide an open source 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 the centralized data collector aims to estimate the histogram of data distributed among clients, while providing privacy guarantees to 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 a 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 making other unappealing trade-offs, such as relying on expensive,

novel cryptography [22, 23, 24, 25, 8, 9], (e.g., multi-party computation, homomorphic encryption), or multiple rounds of expensive communication between participants [9], among other concerns. These systems entail computational, bandwidth and financial costs that make adoption difficult-to-impossible for all but resource-rich organizations.

In this work, we describe a novel system for the problem of private distributed histogram estimation 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 sample-and-threshold DP [8] with verifiable client-side thresholding [7] to provide DP guarantees without the use of trusted third parties. *Nebula* uses two *untrusted* 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, similar to [8]; *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 present *Nebula*, a system that makes 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, utility, and efficiency guarantees.

3. **empirical measurements** of the system's utility and efficiency over multiple real-world datasets.

2 Problem Formulation and Threat Model

We consider a scenario where there are N clients generating a dataset, $D = \{x_i\}_{i=1}^{N}$, where each data point x_i is generated by i-th client. We consider an *untrusted* third party (i.e., aggregation server) who wants to collect these data. Collecting data generated by clients 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 must protect the privacy of data contributors i.e., the final output provably leaks as little as possible about each client's private data.

Our objective in this work is to design a system that enables an aggregation server to construct an accurate histogram over clients data with the following desiderata:

- protecting client's privacy;
- avoiding trust in servers;
- avoiding communication between servers;
- avoiding communication between clients;

- being efficient with little client-facing costs.

To protect the client's 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:

$$\Pr[\mathcal{A}(D) \in S] \leq e^{\varepsilon} \Pr[\mathcal{A}(D') \in S] + \delta, \tag{1}$$

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 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. δ relaxes the privacy guarantees for unlikely events. Our protocol guarantees that the final output (i.e., histogram) is DP, and furthermore the view of each server corrupted by an adversary satisfies a restriction of DP to computationally bounded adversary known as computational DP [26], which is designed to handle protocols that combine DP with cryptographic techniques.

To not force clients to trust the server, we use τ-out-of-N secret sharing scheme, $\Pi_{\tau,N}$, [27] such that only the final output of our DP protocol $\mathcal{A}$ is revealed to the server and nothing else. Our secure protocol provides information-theoretic security and is 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.

To avoid communication between servers and clients, we generate the above randomness for running $\Pi_{\tau,N}$ by another untrusted party (i.e., randomness server) in a verifiable manner for clients without learning clients' data. We assume that the aggregation server and the randomness server are non-colluding which can be imposed via strict legal bindings.

As our goal is to protect clients privacy, following the literature [9], we consider honest-but-curious clients. However, protecting against malicious clients who do not follow the protocol or collude with any server to violate privacy of honest clients is an interesting future direction.

Finally, we design our system to be *efficient* such that financial costs, computational costs and bandwidth consumption are low, lightweight and small, by 1) not requiring any communication between the randomness server and the aggregation server; and 2) requiring very little effort and one single round of interaction with each server from clients.

Note that the server and clients agree on two public parameters: *i)* the total differential privacy budget, ε; and *ii)* the security parameter λ used for the secret sharing scheme.

Table 1 shows the notation used throughout this paper.

Table 1: Notation table.

Notation	Meaning
D	Dataset
N	Number of clients
x	Client data
sbm	Client encrypted data
p_s	Sampling rate
ε	Privacy budget
λ	Security Parameter
τ	Threshold for pruning values
$\Pi_{\tau,N}$	τ-out-of-N secret sharing scheme
$H(\cdot)$	Hash function

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

$(\varepsilon_{\text{Re}}, \delta_{\text{Re}}), (\varepsilon_{\text{Unre}}, \delta_{\text{Unre}}) \leftarrow (\varepsilon, \delta)$ ▷ 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 $i \in N$ **do**
  $z_i = \text{Random}([0,1])$ ▷ Each client locally performs a Bernoulli test to decide whether to participate
  $r_i = \textit{Client-RandomnessServer}(x_i, \text{pp}, H(\cdot))$ ▷ Oblivious and verifiable randomness generation (Algorithm 2)
  if $z_i \leq p_s$ **then**
    $\text{sbm}, - \leftarrow \textit{LocalSecretSharing}(x_i, r_i, \Pi_{\tau,N})$ ▷ Each client locally encrypts their data (Algorithm 3)
    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

3 *Nebula* Design

We design a novel Differentially Private 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 [28], and one that aggregates and learns threshold-meeting values from clients).

At a high level, *Nebula* (Algorithm 1) works as follows: *i)* 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; *ii)* Each participating client independent of other clients obliviously communicates with the randomness server and encrypts its data; *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.

Unique among other proposed and deployed systems, *Nebula* achieves all of the following: *(i)* DP guarantees for participating clients with *(ii)* realistic and light trust assumptions and *(iii)* high utility and *(iv)* high efficiency.

Next, we describe *Nebula*'s steps in detail.

3.1 Oblivious and Verifiable Randomness Generation

Each participating client starts by sampling randomness r from an untrusted randomness server running a Verifiable Oblivious Pseudorandom Function (VOPRF) function. Each client sends their data in an oblivious manner such that the randomness server cannot learn the values held by the client. As shown in Algorithm 2, the client locally generates a random value r', and then hashes and then blinds their original data value x to generate b from $b = H(x)^{r'}$. The client then sends b to the untrusted randomness server, which executes the OPRF function, yielding z from $z = b^{\text{msk}}$, using the server's secret msk which is kept hidden from the client. The server returns z to the client, which unblinds ($w = z^{1/r'}$), and then unhashes ($r = H(x, w)$). We note that clients contributing the same original value receive the same randomness r from the randomness server, without i) the randomness server knowing multiple clients hold the same underlying value, and ii) without requiring any communication between clients. The randomness server is therefore untrusted, since the randomness server never sees the plain-text versions of client values, and clients can verify whether the server is correctly following the protocol in zero knowledge.

3.2 Local Data Preparation

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 $r_t = H(r \| t)$. Each of these three randomnesses are used for different purposes: 1) r_1 is used to seed a pseudorandom generator function and derives a symmetric key $\text{Key} \leftarrow \{0,1\}^{\lambda'}$ to be

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

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

1: $h = H(x)$ ▷ The client hashes its input
2: $r' \leftarrow R$ ▷ The client generates a random value
3: $b = h^{r'}$ ▷ The client blinds its hash value and sends it to the randomness server
4: $(\text{msk}, \text{mpk}) \leftarrow \text{KeyGen}(\text{pp})$ ▷ The randomness server generates a keypair based on the public key parameter
5: $\text{z} = b^{\text{msk}}$ ▷ The randomness server produces a response and sends it back to the client with its ZKproof
6: $w = \text{z}^{\frac{1}{r'}}$ ▷ The client unblinds the response using its local and private randomness
7: $r = H(w, x)$ ▷ The client obtains the randomness and verifies the proof
8: **Return** r

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

1: $\{r_1, r_2, r_3\} = \text{RandomOracleHash}(r)$ ▷ Parsing the randomness to three random values
2: $\text{Key} = \text{PseudorandomGenerator}(r_1)$ ▷ Deriving a symmetric key using pseudorandom generator with r_1 as the seed
3: $\mathbf{c} \leftarrow \text{Enc}(\text{Key}, \mathbf{x})$ ▷ Encrypting the data and generating a ciphertext
4: $t \leftarrow r_3$ ▷ Generating a tag for the data using r_3
5: $s = \Pi_{\tau,N}(r_1; r_2)$ ▷ Constructing a secret-share of the random value r_1 used for deriving the encryption key
6: $\mathsf{sbm} \leftarrow (\mathbf{c}, s, t)$ ▷ Creating a tagged submission for the data
7: **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

1: $\text{Dummy} = \{\}$ ▷ The set containing groups of dummy data
2: Select a client for creating dummy data
3: **for** $i \in \tau - 1$ **do**
4: $c \leftarrow \text{TSDLap}(\lambda = 2/\varepsilon_{\text{Unre}}, t = 2 + 2/\varepsilon_{\text{Unre}} \log(2/\delta_{\text{Unre}}))$ ▷ The client generates a random value
5: $\{\text{tag}_j\}_{j=1}^{c} = \text{UniqueTagGenerator}(c)$ ▷ The client creates random number of unique tags
6: **for** $j \in c$ **do**
7: $s_j = \{(\text{enc}_j, \text{tag}_j)^i\}$ ▷ The client creates a set containing i zero-value items with the same unique tag
8: $\text{Dummy.append}(s_j)$
9: **Return** Dummy

used for encrypting client's input data $\mathbf{c} = \text{Enc}(\text{Key}, \mathbf{x})$; 2) r_2 is used as the randomness input to $\Pi_{\tau,N}$ for producing a random τ-out-of-N share[1] $s_k \in F_q$ of r_1 while providing an information-theoretic security [27]; and 3) r_3 is used as a tag informing the aggregation server which shares to combine to recover the encryption key.

Next, each client constructs their message as $\mathsf{sbm} \leftarrow (\mathbf{c}, s_i, r_3)$. 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 participates by sending their encrypted message $\mathsf{sbm} \leftarrow (\mathbf{c}, s_i, r_3)$, otherwise it abstains.

3.3 Dummy Data Injection

The aggregation server can see the tags of unrevealed submissions[2] which might leak information about their underlying values. To control this leakage, *Nebula* adds dummy submissions such that the amount of information that the aggregation server can learn about these tags is bounded within the DP guaranteed range (Algorithm 4). In particular, dummy data makes the histogram of unrevealed submissions differentially private. Remarkably, this is accomplished without impacting the correctness and utility of the aggregations, as the dummy data is automatically filtered out due to each dummy data group being smaller than the threshold (see line 3 in Algorithm 4). This dummy data injection can be done by randomly selecting a client.

[1]Note that our implementation of τ-out-of-N secret sharing produces random shares without any client's identity.

[2]This is 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.

We use truncated shifted discrete Laplace distribution, TSDLap(λ, t) on $\{0, ..., 2t\}$ 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:*

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

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

Section 4 provides a proof that sampling, combined with dummy data and thresholding, provides strict DP guarantees [29, 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 [30] 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. In practical deployments where timestamps (e.g., the order of messages sent/received) are observed, we make the distribution of each timestamp independent of the messages and their source such that it gives no additional information about the sender. This can be done efficiently in multiple ways. For instance, whenever a client's data is generated on their device, 1) the data gets temporarily stored on-device, and after a delay all clients simultaneously submit their data; or 2) the client locally draws a real number t uniformly into $[0, 1]$ for their message and send the message at time t.

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. In particular, the aggregation groups submissions based on their tags r_3 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* $\tau = \frac{1}{C_\alpha} \ln\left(\frac{1}{\delta_{Re}}\right)$ *where* $0 < \alpha \leq 1$ *and* $C_\alpha = \ln\left(\frac{1}{\alpha}\right) - \frac{1}{1+\alpha}$. *Then, Nebula enables the aggregator server to obtain an* (ε, δ)*-DP histogram over* D *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 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.

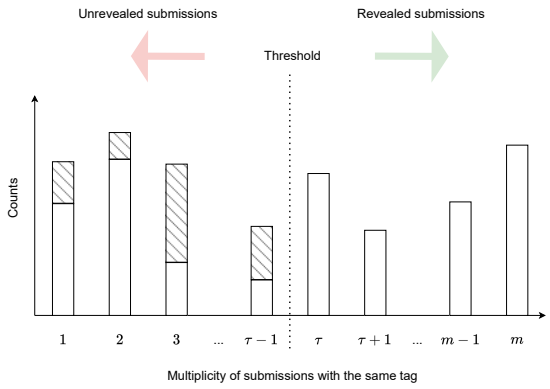

Figure 1: *Nebula*'s output to the aggregation server which is a histogram $\mathcal{H}$ of multiplicities where $\mathcal{H}_i$ represents the number of submissions with the same tag, with multiplicity i and $i \in [m]$. This histogram is obtained based on submissions that each client sent with probability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

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 $\text{TSDLap}(\lambda, t)$ on $\{0, ..., 2t\}$ with $\lambda = \Delta/\varepsilon_{\text{Unre}}$ and $t = \Delta + \Delta/\varepsilon_{\text{Unre}} \log(2/\delta_{\text{Unre}})$ to ensure $(\varepsilon_{\text{Unre}}, \delta_{\text{Unre}})$-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 $i - 1$:

$$\begin{cases} \mathcal{Z}_i & = \mathcal{Z}'_i + 1 \quad \text{entry } i \\ \mathcal{Z}_{i-1} & = \mathcal{Z}'_{i-1} - 1 \quad \text{entry } i - 1 \\ \mathcal{Z}_y & = \mathcal{Z}'_y \quad \text{other entries } \forall y \notin \{i, i-1\} \end{cases} \tag{3}$$

Therefore the sensitivity $\Delta = 2$.

Revealed submission. In the event where the differing submission is revealed, we can leverage DP guarantees of 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:

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

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

$$\binom{k-1}{v} (1 - p_s)^{((k-1)-v)} (p_s)^v. \tag{5}$$

Now, we can bound the ratio of probabilities of seeing data with a given count v by dividing Eq. 4 by Eq. 5 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

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

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:

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

Rearranging the upper bound, we have:

$$v \leq k(1 - e^{-\varepsilon_{\mathrm{Re}}} + e^{-\varepsilon_{\mathrm{Re}}} p_s) \tag{8}$$

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. (7) is greater than kp_s, which is 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 $(\varepsilon_{\mathrm{Re}}, \delta_{\mathrm{Re}})$-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 $\delta_{\mathrm{Re}} \leq \exp(-\frac{\tau}{q} D(q \| p))$. Therefore, sampling with probability $p_s = \alpha(1 - e^{-\varepsilon_{\mathrm{Re}}})$ and thresholding with $\tau = \frac{1}{C_\alpha} \ln(\frac{1}{\delta_{\mathrm{Re}}})$ where $0 < \alpha \leq 1$ and $C_\alpha = \ln(\frac{1}{\alpha}) - \frac{1}{1+\alpha}$ provides $(\varepsilon_{\mathrm{Re}}, \delta_{\mathrm{Re}})$-DP [8]. □

4.2 Cryptographic Security

We leverage the methodology of [7] for producing consistent data encryption. Therefore, the security and robustness of the data encoding procedure against malicious adversaries follows a similar argument to [7], in the random-oracle model.

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 aggregator 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, $\mathrm{TSDLap}(\lambda, t)$ on $\{0, ..., 2t\}$ to generate dummy data. The expectation of $\mathrm{TSDLap}(\lambda, t)$ 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 independent of dummy data, and only the sampling rate p_s and the threshold τ affect the closeness of the histogram estimated by *Nebula* to the true histogram. By leveraging utility guarantees of sample-and-threshold approach [8], *Nebula* will reveal to the aggregation server any value whose frequency is sufficiently above the threshold with high probability.

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

5 Experiments

We implement *Nebula* and empirically validate its performance as follows:

- **Effectiveness in collecting highly accurate and private data**: In complement to the analytical privacy and accuracy guarantees obtained in Section 4, we empirically demonstrate that the estimated histogram by *Nebula* is close to the true histogram constructed from all clients' original data while ensuring that it meets strong privacy guarantees.
- **Efficiency in private and secure data collection**: In complement to the analytical communication costs obtained in Section 4, we empirically demonstrate the ability of *Nebula* to scale to real-world use cases because of its low computational, bandwidth, and financial costs.

5.1 Datasets

We assess the performance of *Nebula* in collecting three datasets. Two of these are *privacy-sensitive in nature*—the IPUMS Census dataset[3] and the Foursquare dataset [31]. 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. 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. Yang et al. took advantage of Twitter's public stream to extract tweets that contained Foursquare-tagged submissions over 18 months, from April 2012 to September 2013. 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.[4] 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,[5] 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

[3] `https://usa.ipums.org/usa/`

[4] An example is 41.029717, 28.974420: a restaurant in Istanbul, Turkey.

[5] plain text edition from `https://cs.stanford.edu/people/karpathy/char-rnn/shakespeare_input.txt`

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 distribution 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.

5.2 Setup

Evaluation metrics. We evaluate the effectiveness and efficiency of *Nebula* as follows.

- Effectiveness: We compute the error as the absolute difference between the estimated normalized frequencies (i.e., per bin counts divided by the total number of counts) and the original normalized frequencies.
- Efficiency: We evaluate the various costs of our framework through (1) Computational costs measured as CPU running time for both the client-side encode 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 $t = 15$.

5.3 Utility Comparison to Existing Works

We compare our DP data collection framework, *Nebula*, with existing state-of-the-art methods in DP data collection. To demonstrate the effect of *Nebula* in improving privacy-utility tradeoffs, we compare against *i)* local differential privacy based on [20] in which each client locally adds Laplace noise to their data; and *ii)* central DP in which the server receives the raw client's data, computes the true histogram and adds Laplace noise to each bin.

Table 2 shows the absolute error of *Nebula*, local DP approaches and central DP approaches on the three datasets. We compute the absolute error of each method in estimating the histogram. Results demonstrate that *Nebula* is more effective than the alternative local DP approach 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, in the Shakespeare dataset, client submissions are necessarily single values, and its domain size is smaller than the other two datasets. We analyse further the effect of the domain size in Figure 2. 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 global DP method, consistent with our above explanation of the performance difference between datasets. Consistently across all bins, *Nebula* offers significantly lower absolute error in the estimated histogram compared to local DP.

Conversely, the absolute error in Foursquare is 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.

Table 2: Comparing the utility of *Nebula* with local DP and central DP approaches on the example datasets in terms of the sum absolute error between the estimated and true histograms using an $\varepsilon = 1$ DP privacy guarantee. *Nebula* significantly improves the absolute error of local DP based approaches, pushing the utility closer to central DP based approaches while removing the trust of the central DP models on the server.

Dataset	Approach		
	Central DP	*Nebula*	Local DP
Shakespeare	0.0001	0.0197	0.3955
IPUMS	0.0044	0.1932	1.6622
Foursquare	0.0005	1.3999	2.0000

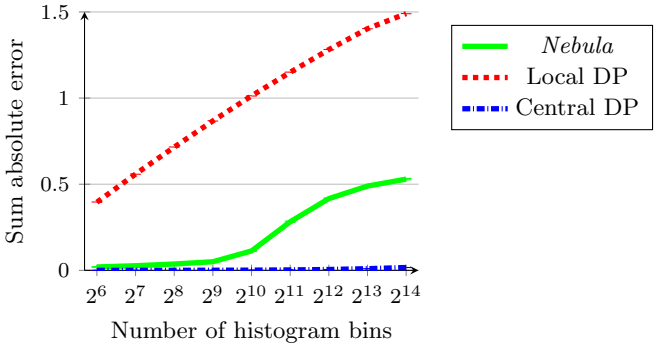

Figure 2: Utility of *Nebula* compared with local and central differential privacy applied to the Shakespeare database as a function of histogram bins using an $\varepsilon = 1$ DP privacy guarantee. The word-frequency estimate of *Nebula* is more accurate than local DP while while removing the trust of the central DP models on the server.

5.4 Towards Capturing Marginal Histograms

In some scenarios, clients' data consist of multiple attributes [32, 33, 34]. In particular, each client generates a multi-dimensional data point that is a vector of $\ell \geq 1$ attributes of the form $\mathbf{x} = [x_1, x_2, ..., x_\ell]$. Consider the following motivating and practical example in the case of telemetry [11]. Clients generate some five-dimensional crash reports shown in Table 3 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 [35]. 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 a novel multi-dimensional data encoding (Algorithm 5) with a "hierarchy-of-priority" in which each client constructs a ciphertext, without any client-client communication, 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. Next, we describe the multi-dimensional data encoding in detail.

Multi-dimensional local data encryption. Each client creates ℓ sequential prefixes $\mathbf{x}^{(1)} = [x_1], \cdots,$ $\mathbf{x}^{(\ell)} = [x_1, \cdots, x_\ell]$ such that each prefix contains a sequence of attributes from the beginning to its corre-

Table 3: Motivating example of multi-dimensional data points generated by clients while using a Browser.

Anon. ID	URL visited	Operating system	Version	Region	Installation Date
A	`https://www.wikipedia.org/`	Windows	1.5	UK	October 2023
B	`https://www.reddit.com/`	iOS	0.2	US	September 2023
C	`https://stackoverflow.com/`	Android	4.8	Africa	July 2023
D	`https://www.wikipedia.org/`	Linux	10.4	Europe	December 2022

Algorithm 5: Client's multi-dimensional data encoding

Input: A client holding a multi-dimensional data point $\mathbf{x} = [x_1, x_2, ..., x_\ell]$ with ℓ attributes, K-out-of-N secret-sharing scheme $\Pi_{K,N}$
Output: Encoded attributes SBM

1: SBM = {}
2: **for** $i \in \ell$ **do**
3: $\mathbf{x}^{(i)} = [x_j]_{j=0}^{i}$ ▷ Generating a prefix containing a sequence of attributes from the beginning
4: $r^{(i)} = \textit{Client-RandomnessServer}(\mathbf{x}^{(i)})$ ▷ Running Algorithm 2 to receive randomness while keeping the prefix secret
5: $\mathrm{sbm}^{(i)}, \mathrm{Key}^{(i)} \leftarrow \textit{LocalSecretSharing}(x_i, r_i)$ ▷ Each client locally encrypts the prefix (running Algorithm 3)
6: **if** $i == 1$ **then**
7: $\hat{\mathsf{sbm}}^{(i)} \leftarrow \mathsf{sbm}^{(i)}$
8: **else**
9: $\hat{\mathsf{sbm}}^{(i)} \leftarrow \mathrm{Enc}(\mathrm{Key}^{(i-1)}, \mathsf{sbm}^{(i)})$ ▷ Encrypting the tagged submission with the key of its previous prefix
10: SBM.append($\hat{\mathsf{sbm}}^{(i)}$)
11: **Return** SBM

sponding index. These prefixes enable to capture joint histograms of multiple attributes instead of each individual attribute. Each client encodes prefixes $\mathbf{x}^{(i)}$, where $i \in \{1, \ldots, \ell\}$, 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 secret-shares each prefix through running Algorithm 2 and Algorithm 3 and construct messages, $\mathsf{sbm}^{(i)}$. Submitting $\mathsf{sbm}^{(1)}, \mathsf{sbm}^{(2)}, ..., \mathsf{sbm}^{(\ell)}$ 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 5) 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 we encrypt each prefix $\mathsf{sbm}^{(i)}$ with the key of its previous prefix which gets revealed once the previous prefix is decoded. Each prefix at layer i (except for the first) is then encrypted with the key of the previously encoded prefix. In particular, each client computes an encrypted ciphertext $\hat{\mathsf{sbm}}^{(i)} \leftarrow \mathrm{Enc}(\mathrm{Key}^{(i-1)}, \mathsf{sbm}^{(i)})$ with the described symmetric encryption operation for each $i \geq 2$. Each client creates the super-message as the tuple $\mathsf{SBM} = (\mathsf{sbm}^{(1)}, \hat{\mathsf{sbm}}^{(2)}, \ldots, \hat{\mathsf{sbm}}^{(\ell)})$ and sends it to the server based on the outcome of Bernoulli test discussed in Section 3.

Results. Figure 3 shows the absolute error of *Nebula* when using the multi-dimensional encoding (Algorithm 5) on the IPUMS dataset. We compute the absolute error in estimating the histogram for each prefix. Results demonstrate that multi-dimensional encodings improve the ability of *Nebula* to collect high-utility multi-dimensional 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 2). 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

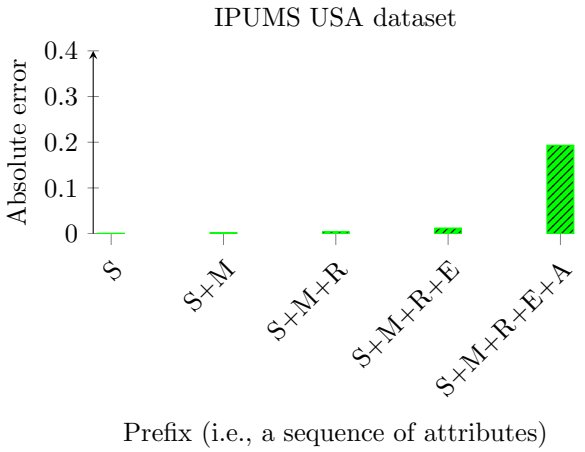

Figure 3: Improving the utility of *Nebula* in estimating the histogram on the multi-attribute IPUMS dataset using multi-dimensional data encoding (Algorithm 5). IPUMS contains 5 attributes–S: Sex; M: Marriage status; R: Race; E: Education; A: Age. We compute the utility as the absolute error between 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).

location with increasing granularity. With this encoding, partial recovery of joint attributes amounts to recovering regional aggregate popularity at multiple scales. Figure 4 shows the improvement in the absolute error using this scheme for the Foursquare dataset. To further demonstrate the effectiveness of *Nebula*, we compare the estimated *Nebula* histogram and the true Foursquare histogram of country codes in Figure 5. We observe that *Nebula* preserves the relative frequencies across attribute values. For example, the most popular items stay popular in the estimated histogram.

Note that this utility improvement might come at a cost in terms of privacy. An adversary aggregation server who has perfect background knowledge (full knowledge of all records in D, and full knowledge of the victim record) and aims to infer whether the victim record is in the input dataset or not can recover some information. However, it is common to ignore this leakage in practice [36] as this above privacy leakage happens for theoretical datasets coming from extremely skewed distribution and mostly binary values. Several ways have been proposed to address this leakage in the literature including relaxing the definition of differential privacy by considering practical data distributions [37, 38, 39, 40, 36, 29]. For example, [29] relaxes the assumption that the adversary knows all attributes of the client, and in addition statistical information about the rest of the input dataset [29].

5.5 *Nebula* is Efficient

We evaluate computational, bandwidth, and financial costs of *Nebula*.

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 4, 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 4) 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

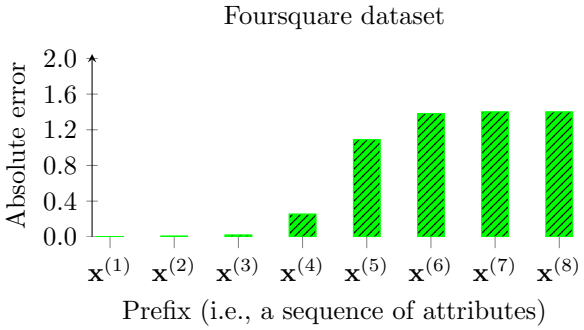

Figure 4: Improving the utility of *Nebula* in estimating the histogram on the multi-attribute Foursquare dataset using multi-dimensional data encoding (Algorithm 5). We compute the histogram of each prefix $\mathbf{x}^{(1)} = [x_1]$, $\mathbf{x}^{(2)} = [x_1, x_2]$, $\cdots$, $\mathbf{x}^{(8)} = [x_1, \cdots, x_8]$. 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).

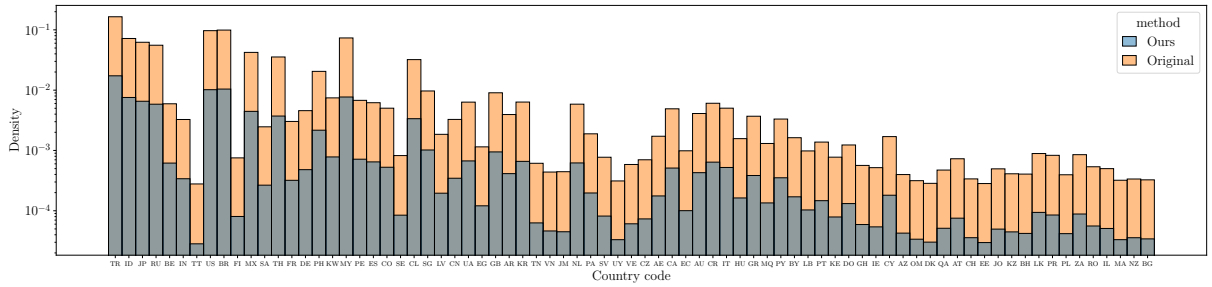

Figure 5: Original and estimated histogram obtained privately by *Nebula* using Foursquare dataset. The private histogram estimated by *Nebula* (Ours) is close to the histogram of the original data (Original). *Nebula* also preserves the relative order across attribute values.

Table 4: 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

milliseconds for each Foursquare client submission. For simplicity, we omit the zero-knowledge proof steps of the verifiable OPRF in these benchmarks. Table 4 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 6 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 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

Table 5: Efficiency of *Nebula* in terms of running time on the example 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 6: 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 w/ randomness server	256	160	32
Client w/ aggregation server	2465	1470	373

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 Amazon Web Services 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.

5.6 *Nebula* is Scalable

We further analyze the scalability of *Nebula* by considering various numbers of attributes using the same hardware described in Section 5.5. Timings for the IPUMS and Foursquare datasets report the more expensive multi-attribute encoding scheme (Algorithm 5), 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. These results demonstrate that ***Nebula* can collect multi-dimensional data from a very large number of clients with small costs.**

6 Related work

Threshold-aggregation data collection systems [7, 6, 41, 17, 42, 43, 44, 25, 45, 46] 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

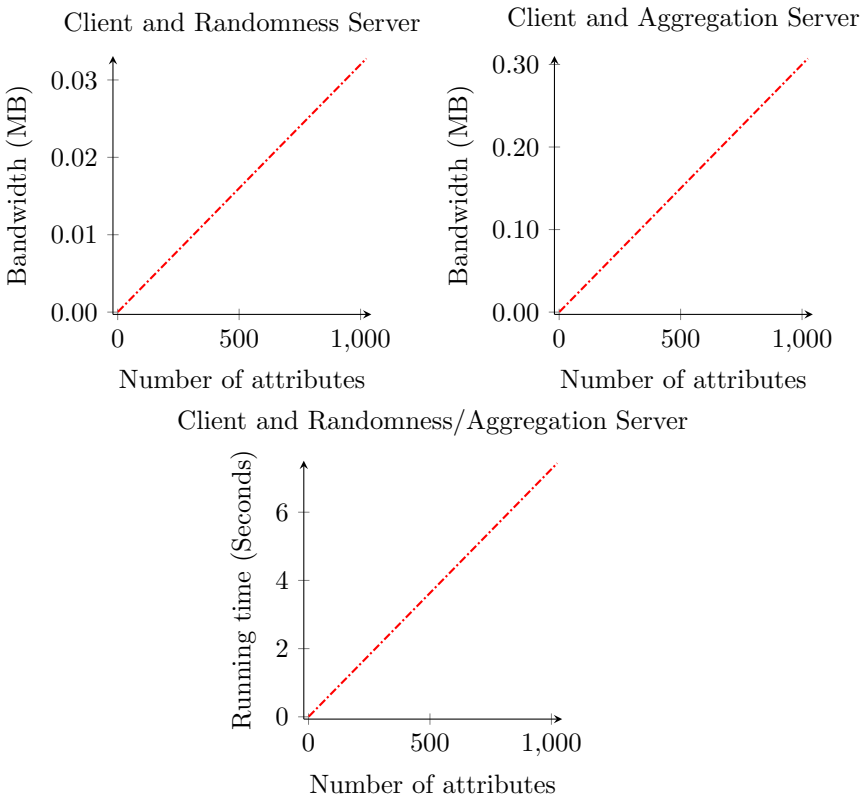

Figure 6: Scalability of *Nebula* in terms of the computational and bandwidth overhead introduced for clients. ***Nebula* scales to a large number of attributes with negligible costs.**

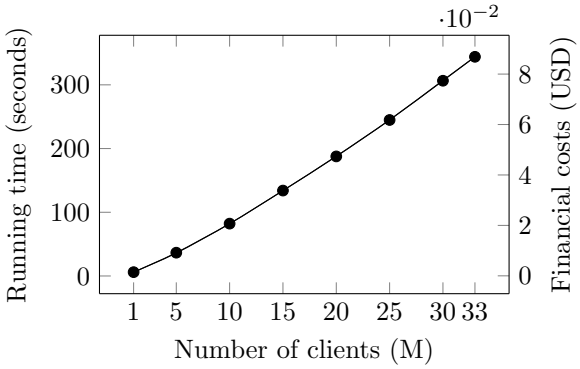

Figure 7: Scalability of *Nebula* in terms of the computational and financial costs introduced for the aggregation server. ***Nebula* scales to a large number of clients with negligible costs.**

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 (e.g., $[x, y, z]$) as three independent records (e.g., $[x]$, $[y]$, and $[z]$). 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 make different utility-privacy 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 aggregation servers do not provide the privacy protections they promise (intentionally or otherwise) [47]. Central DP systems also carry the risk of a single point of failure for data breaches [48, 49].

Local DP systems, on the other hand, provide strong privacy guarantees, typically by perturbing data before revealing it to untrusted parties. [50, 51, 52, 44] 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 [53] 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 [54, 55, 56, 23, 10, 57, 48, 9]. But these proposals suffer from their own drawbacks, including *i)* requiring network 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 [57, 55, 54, 56], and *iv)* requiring interactive communication between clients and servers. Requiring interactions between the servers (more than 1 aggregator 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 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. *Nebula* avoids communication and interactions between servers, making the practical deployment of non-collusion assumptions more feasible and easier to maintain.

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 [45] and verifiable shuffling [46] can provide security and privacy guarantees similar to (but without requiring) trusted hardware, though at the cost of increased interactivity.

7 Discussion

In this paper, we proposed *Nebula* that can be used to privately estimate histogram of data generated by clients. *Nebula* introduces necessary randomness for the privacy protection of clients through sampling, pruning, and dummy data, and removes the trust assumption on the server through a customized secret-sharing protocol. Incorporating synergies and optimizations on both fronts enables *Nebula* to provide high utility without prohibitive trust requirements, and without requiring computationally expensive cryptographic operations, or bandwidth consuming multi-round communications between the clients and the servers. We analytically and empirically demonstrated that *Nebula* is effective, efficient and scalable.

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.

[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*, vol. 11, p. 1125011, 2023.

[4] A. Bharadwaj and G. Cormode, "Federated computation: a survey of concepts and challenges," *Distributed and Parallel Databases*, pp. 1–37, 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," *arXiv preprint arXiv:2307.11749*, 2023.

[6] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," 2023.

[7] A. Davidson, P. Snyder, E. V. Quirk, J. Genereux, B. Livshits, and H. Haddadi, "STAR: Distributed Secret Sharing for Private Threshold Aggregation," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. Association for Computing Machinery, 2022. [Online]. Available: https://arxiv.org/abs/2109.10074

[8] G. Cormode and A. Bharadwaj, "Sample-and-threshold differential privacy: Histograms and applications," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 1420–1431.

[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 *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 307–321.

[10] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, 2017, pp. 259–282.

[11] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 1054–1067.

[12] H. Li, S. Navot, and S. Tessaro, "Popstar: Lightweight threshold reporting with reduced leakage," *Cryptology ePrint Archive*, 2024.

[13] A. Narayanan and V. Shmatikov, "How to break anonymity of the netflix prize dataset," *arXiv preprint cs/0610105*, 2006.

[14] N. Holohan, S. Antonatos, S. Braghin, and P. Mac Aonghusa, "(k, ϵ)-anonymity: k-anonymity with ϵ-differential privacy," *arXiv preprint arXiv:1710.01615*, 2017.

[15] C. Dwork, "Differential privacy," in *International colloquium on automata, languages, and programming*. Springer, 2006, pp. 1–12.

[16] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[17] R. Bassily and A. Smith, "Local, private, efficient protocols for succinct histograms," in *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, 2015, pp. 127–135.

[18] T. Wang, J. Blocki, N. Li, and S. Jha, "Locally differentially private protocols for frequency estimation," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 729–745.

[19] J. Acharya, Z. Sun, and H. Zhang, "Hadamard response: Estimating distributions privately, efficiently, and with little communication," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1120–1129.

[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 *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 638–649.

[21] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett, "Differentially private histogram publication," *The VLDB journal*, vol. 22, pp. 797–822, 2013.

[22] B. Balle, J. Bell, A. Gascón, and K. Nissim, "The privacy blanket of the shuffle model," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part II 39*. Springer, 2019, pp. 638–667.

[23] A. Cheu, A. Smith, J. Ullman, D. Zeber, and M. Zhilyaev, "Distributed differential privacy via shuffling," in *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*. Springer, 2019, pp. 375–403.

[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 *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2019, pp. 2468–2479.

[25] A. Bittau, U. 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 *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 441–459. [Online]. Available: https://doi.org/10.1145/3132747.3132769

[26] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan, "Computational differential privacy," in *Annual International Cryptology Conference*. Springer, 2009, pp. 126–142.

[27] 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," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 4, 2020.

[28] N. Tyagi, S. Celi, T. Ristenpart, N. Sullivan, S. Tessaro, and C. A. Wood, "A fast and simple partially oblivious prf, with applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pp. 674–705.

[29] N. Li, W. H. Qardaji, and D. Su, "Provably private data anonymization: Or, k-anonymity meets differential privacy," *CoRR, abs/1101.2604*, vol. 49, p. 55, 2011.

[30] M. Thomson and C. A. Wood, "Oblivious http," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-ohai-ohttp-05, September 2022, https://www.ietf.org/archive/id/draft-ietf-ohai-ohttp-05.txt. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-ohai-ohttp-05.txt

[31] 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*, vol. 7, no. 3, p. 30, 2016. [Online]. Available: https://dl.acm.org/doi/10.1145/2814575

[32] Z. Zhang, T. Wang, N. Li, S. He, and J. Chen, "Calm: Consistent adaptive local marginal for marginal release under local differential privacy," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 212–229.

[33] 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 *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 159–176.

[34] D. J. Leith, "Mobile handset privacy: Measuring the data ios and android send to apple and google," in *Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part II 17*. Springer, 2021, pp. 231–251.

[35] K. Satvat and N. Saxena, "Crashing privacy: An autopsy of a web browser's leaked crash reports," 2018. [Online]. Available: https://arxiv.org/abs/1808.01718

[36] D. Desfontaines and B. Pejó, "Sok: differential privacies," *arXiv preprint arXiv:1906.01337*, 2019.

[37] R. Bassily, A. Groce, J. Katz, and A. Smith, "Coupled-worlds privacy: Exploiting adversarial uncertainty in statistical data privacy," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 2013, pp. 439–448.

[38] Y. Duan, "Privacy without noise," in *Proceedings of the 18th ACM conference on Information and knowledge management*, 2009, pp. 1517–1520.

[39] D. Kifer and A. Machanavajjhala, "A rigorous and customizable framework for privacy," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, 2012, pp. 77–88.

[40] R. Bhaskar, A. Bhowmick, V. Goyal, S. Laxman, and A. Thakurta, "Noiseless database privacy," in *Advances in Cryptology–ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings 17*. Springer, 2011, pp. 215–232.

[41] R. Bassily, K. Nissim, U. Stemmer, and A. Thakurta, "Practical locally private heavy hitters," *Journal of Machine Learning Research*, vol. 21, no. 16, pp. 1–42, 2020. [Online]. Available: http://jmlr.org/papers/v21/18-786.html

[42] M. Bun, J. Nelson, and U. Stemmer, "Heavy hitters and the structure of local privacy," *ACM Trans. Algorithms*, vol. 15, no. 4, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3344722

[43] 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 *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 192–203.

[44] W. Zhu, P. Kairouz, B. McMahan, H. Sun, and W. Li, "Federated heavy hitters discovery with differential privacy," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 108. PMLR, 26–28 Aug 2020. [Online]. Available: http://proceedings.mlr.press/v108/zhu20a.html

[45] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, p. 84–90, Feb. 1981. [Online]. Available: https://doi.org/10.1145/358549.358563

[46] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *Proceedings of the 8th ACM conference on Computer and Communications Security*, 2001, pp. 116–125.

[47] 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 preprint arXiv:1709.02753*, 2017.

[48] A. Roy Chowdhury, C. Wang, X. He, A. Machanavajjhala, and S. Jha, "Cryptϵ: Crypto-assisted differential privacy on untrusted servers," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 603–619.

[49] 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*, vol. 7, no. 4, pp. 532–540, 2019.

[50] P. Kairouz, S. Oh, and P. Viswanath, "Extremal mechanisms for local differential privacy," *CoRR*, vol. abs/1407.1338, 2014. [Online]. Available: http://arxiv.org/abs/1407.1338

[51] R. Bassily, K. Nissim, U. Stemmer, and A. Thakurta, "Practical locally private heavy hitters," *CoRR*, vol. abs/1707.04982, 2017. [Online]. Available: http://arxiv.org/abs/1707.04982

[52] P. Kairouz, K. Bonawitz, and D. Ramage, "Discrete distribution estimation under local privacy," 2016. [Online]. Available: https://arxiv.org/abs/1602.07387

[53] V. Balcer and A. Cheu, "Separating local & shuffled differential privacy via histograms," *arXiv preprint arXiv:1911.06879*, 2019.

[54] J. Böhler and F. Kerschbaum, "Secure multi-party computation of differentially private median," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2147–2164.

[55] J. Boehler and F. Kerschbaum, "Secure sublinear time differentially private median computation," Feb. 1 2022, uS Patent 11,238,167.

[56] M. Pettai and P. Laud, "Combining differential privacy and secure multiparty computation," in *Proceedings of the 31st Annual Computer Security Applications Conference*, 2015, pp. 421–430.

[57] 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 *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.