Selective MPC: Distributed Computation of Differentially Private Key-Value Statistics

Thomas Humphries\textsuperscript{*}  
University of Waterloo  
thomas.humphries@uwaterloo.ca

Shannon Veitch\textsuperscript{*}  
University of Waterloo  
ssveitch@uwaterloo.ca

Rasoul Akhavan Mahdavi\textsuperscript{*}  
University of Waterloo  
rasoul.akhavan.mahdavi@uwaterloo.ca

Florian Kerschbaum  
University of Waterloo  
florian.kerschbaum@uwaterloo.ca

1 INTRODUCTION

Key-value data naturally occurs in many internet applications, such as video ad and mobile app analytics where keys are identifiers and values are time or frequencies [52]. In these applications, a provider collects key-value data from a large number of users to compute aggregate statistics which provide crucial insights. However, as individuals become more privacy-aware, they become increasingly unwilling to divulge their sensitive data to a central party. Consequently, recent work has investigated systems that simultaneously allow for the computation of useful statistics while respecting the privacy of individuals and their data. A popular method for achieving this goal is to use local differential privacy (LDP). Under this model, users take privacy into their own hands, so to speak, by perturbing their data before sending it to a central party to be processed. This technique provides a limit on the amount of information the central party can infer about each user’s data while allowing the central party to compute a meaningful aggregate result. These properties have made local differential privacy a preferential solution for many organizations such as Google [25], Apple [16], Mozilla [23], and Microsoft [17] to collect user statistics. For example, the Chrome Web browser uses local differential privacy to collect aggregate statistics about settings such as the homepage and default search engine to counter software that maliciously alters these settings [25]. As another example, Mozilla experimented with using local differential privacy techniques to collect browser telemetry data in a private manner [23].

Applying LDP to key-value data is more challenging than numerical or categorical data. Naively applying noise to the key implies a high loss in the accuracy of aggregate statistics, since it breaks the correlation between the key and the value. Both Ye et al. [52] and later Gu et al. [30] develop improved LDP mechanisms that reduce this accuracy loss. While improving over naive solutions, this line of work is still confined by the inherent accuracy limitations of the LDP model. Intuitively, this is because each user individually adds randomness, which compounds in the final aggregated result.

An approach for achieving both the accuracy of the central model in DP and the trust of the local model is to use multi-party computation (MPC). A popular type of MPC uses secret sharing, wherein the parties jointly compute a function without any one party learning the input. Applying MPC to key-value data is challenging because it may introduce a prohibitively large computational overhead to keep the inputs concealed. Specifically, simultaneously computing over the keys while keeping them secret is a complex issue. To compute and update aggregate statistics, we require an array which
maintains the totals for each key. If the key is secret shared, and thus kept confidential from each party, then indexing each value into this array using MPC incurs a cost linear in the size of the array. This results in a total computation cost on the order of \(|S|n\), where \(|S|\) and \(n\) denote the total number of key-value pairs and the number of distinct keys, respectively. An alternative approach, using ORAM [38], has polylogarithmic cost; however, requires large constants and is typically applied to huge data set sizes. Hence, these approaches remain impractical. If the key is revealed to the parties or encrypted in a way that reveals the access pattern (e.g., public-key searchable symmetric encryption [14]), the parties will learn the count of each keyword, which violates DP. To prevent this violation, one could pad the counts by adding dummy key-value pairs, restoring an approximate differentially private guarantee. This requires adding an abundance of dummy records in order to achieve any reasonable privacy guarantees. Thus, this approach incurs a large communication cost.

In this paper, we design a new method of MPC called selective multi-party computation. Selective MPC does not conceal keys, resulting in an efficient protocol which does not incur prohibitively high communication costs. In typical MPC solutions, all parties receive shares for all key-value pairs. In contrast, our design involves randomly selecting a subset of parties to receive shares for each key-value pair. Thus, each party only views shares for a random subset of the entire dataset. In other words, our approach simulates removing some data from the view of each party, capturing subtractive noise (similar to how adding dummies in previous approaches captured additive noise). We compose the two approaches (selecting a subset of parties and adding dummies) to emulate a two-sided distribution without removing any data. We obtain a new theoretical result, proving that one can achieve pure differential privacy from joining two different one-sided distributions.

Our approach substantially reduces the computation costs, since our construction reveals keys and access patterns while preserving DP guarantees. The computation complexity of the MPC is reduced by a factor of \(O(|S|)\) over the naive MPC approach, where \(|S|\) is the number of key-value pairs. By simulating both subtractive and additive noise, selective MPC requires a very low number of dummy records. We produce 10–20 times less dummies than the padding approach and achieve low \(\epsilon\) in a pure-DP model (\(\delta = 0\)). Compared to the previous LDP mechanisms, we improve accuracy by a factor of \(O(\sqrt{m})\) where \(m\) is the number of clients. We give a detailed comparison to other approaches in Section 8.

In summary, our contribution is a secure multi-party computation of differentially private frequency and mean estimations over key-value data with the following properties.

- Distributed trust and provable security guarantees.
- Pure differential privacy in the local trust model.
- High accuracy independent of the number of users.
- Scalability up to 10,000 keys with results in at most 20 seconds.

2 PRELIMINARIES

2.1 Differential Privacy

Differential privacy (DP), introduced by Dwork et al. [20], is a common privacy notion that enables the calculation of aggregate statistics on users’ data in a privacy-preserving manner. Intuitively, differential privacy states that a user’s participation has a bounded effect on the final output. More formally, differential privacy can be defined as follows.

**Definition 2.1 (Differential Privacy).** A randomized algorithm \(M : X^n \rightarrow Y\) is \((\epsilon, \delta)\)-DP, if for any pair of neighbouring datasets \(X, X' \in X^n\), and for any \(T \subseteq Y\) we have

\[
Pr[M(X) \in T] \leq e^\epsilon Pr[M(X') \in T] + \delta.
\]

If \(\delta \neq 0\), then we say that the mechanism provides approximate differential privacy. When \(\delta = 0\) it satisfies pure differential privacy.

There are two main trust models in differential privacy. In the central model, we assume the existence of a trusted curator that gathers the users’ data, calculates the noisy aggregate statistic, and publishes it. In the local model, users send noisy data to a (potentially untrusted) curator to calculate the aggregate statistic. Although this model eliminates the need to trust the aggregator, it comes at the cost of utility. This is because there is noise added to each data point on the order of the number of clients.

The most common approach to satisfying DP is to add random noise to the output of a function. To prove that this satisfies DP, the function must have a bounded output. We formalize this notion as the sensitivity of a function.

**Definition 2.2 (Sensitivity).** Let \(f : X^n \rightarrow \mathbb{R}^k\). If \(D\) is a distance metric between elements of \(\mathbb{R}^k\) then the \(D\)-sensitivity of \(f\) is

\[
\Delta_D^f = \max_{(X,X') \in \mathcal{T}} D(f(X), f(X')),
\]

where \((X,X')\) are pairs of neighbouring datasets.

Common examples of the distance metric are the \(l_1\)-norm and \(l_2\)-norm. With the notion of sensitivity, we can now define the Laplace mechanism, which uses sensitivity based on the \(l_1\)-norm, which we denote \(\Delta_1^f\) for simplicity.

**Definition 2.3 (Laplace Mechanism).** Let \(f : X^n \rightarrow \mathbb{R}^k\). The Laplace mechanism is defined as

\[
M(X) = f(X) + \operatorname{Lap}(\Delta_1^f / \epsilon)\alpha^n,
\]

where \(\operatorname{Lap}(\Delta_1^f / \epsilon)\alpha^n\) represents \(n\) i.i.d draws from the Laplace distribution with parameter \(\Delta_1^f / \epsilon\).

It is well known that the Laplace mechanism satisfies \(\epsilon\)-DP [21].

2.2 Secret Sharing

A secret sharing scheme [8, 44] enables a secret to be split into \(n\) shares to be distributed among a set of \(n\) participants. The secret is split such that the \(n\) shares permit reconstruction of the secret, but no set of \(n - 1\) or fewer shares yield any information about the secret.

For example, consider the additive secret sharing scheme. In this scheme, to split the secret \(s\), we select \(n - 1\) shares \(s_1, \ldots, s_{n-1}\) at random from the domain and let \(s_n = s - \sum_{i=1}^{n-1} s_i\). Then all players can recover the secret \(s\) by computing \(s = \sum_{i=2}^{n} s_i\). Note that each share is computed as a fixed linear function of the secret and random elements of a field, so participants can locally compute any linear function of shared values. We use \([x]\) to denote that the value
of $x$ is shared among participants. For example, given constants $v_1, v_2, v_3$ and shares of values $[x], [y]$, one can locally compute

$$v_1[x] + v_2[y] + v_3 = [v_1 \cdot x + v_2 \cdot y + v_3]$$

to obtain shares of the value $v_1 \cdot x + v_2 \cdot y + v_3$. This property of the additive secret sharing scheme enables the construction of more general multi-party computation protocols [13].

### 2.3 Secure Multi-party Computation

Secure multi-party computation (MPC) allows a set of participants \{p_1, \ldots, p_n\}, where $p_i$ holds private data $d_i$, to jointly compute a function $f(d_1, \ldots, d_n)$, while keeping their input private and revealing only the output. We focus on a variant of MPC which performs operations over shares of data [6].

In MPC based on secret sharing, participants distribute shares of their data among a set of computation nodes. The computation nodes perform a computation over the secret shared data to obtain shares of a function of the participants’ data, $f(d_1, \ldots, d_n)$. Most MPC protocols represent the desired function as a circuit using basic operations over a finite field (addition and multiplication) and provide subprotocols for performing those basic operations [15, 37].

We can consider two different models for secure multi-party computation: the semi-honest (or passive) model and the malicious (or active) model. In the semi-honest model, introduced by Goldreich [28, §7.2.2], adversaries do not deviate from the protocol but may gather information to infer private information. In the malicious model [28, §7.2.3], adversaries may deviate from the protocol by altering messages or other means to infer private information.

### 3 PROBLEM STATEMENT

Our goal is to present a protocol that permits the efficient computation of differentially private statistics over key-value data while sacrificing minimal trust to any party. A naive solution might first divide the key-value pairs into a set of keys and a set of values and then apply existing methods for computing differentially private histograms to each set. However, as Ye et al. [52] argue, this does not maintain the correlation between keys and values. For this reason, computing meaningful differentially private statistics over key-value data is more challenging than histograms.

#### 3.1 Notation

We have a set of $m$ clients, each denoted by an index $i \in [m]$. Each client has a set of key-value pairs $S_i$. We denote the $j$th key-value pair of client $i$ to be $(k_{ij}, v_{ij})$, where each $k_{ij} \in \mathcal{K}$ and each $v_{ij} \in \mathcal{V}$ for some sets $\mathcal{K}, \mathcal{V}$. The multiset of key-value pairs owned by all $m$ clients is denoted $\mathcal{S}$. For a value or algorithm, we adopt the convention of using a subscript $k$ to restrict it to a key $k$. We summarize the notation in Table 1.

In this work, we focus on the following two statistics over key-value data. Let $\mathcal{A}_k = \{(k_{ij}, v_{ij}) \in \mathcal{S} \mid k_{ij} = k\}$ be the multiset of key-value pairs corresponding to key $k$.

- **Frequency estimation.** The frequency of some key $k$, $q_k$, is defined as the number of instances of the key $k$ in the set $\mathcal{S}$. That is, $q_k(\mathcal{S}) = |\mathcal{A}_k|$. We denote the vector of frequencies of all keys by $q(\mathcal{S})$.

| Symbol | Description |
|--------|-------------|
| $\mathcal{K}$ | domain of the keys |
| $\mathcal{V}$ | domain of the values |
| $n$ | the size of the key space $\mathcal{K}$ |
| $m$ | the number of clients |
| $\mathcal{S}$ | multiset of all key-value pairs |
| $S_i$ | set of key-value pairs owned by client $i$ |
| $\mathcal{R}$ | bound on the values in $\mathcal{V}$ |
| $t$ | the number of computation nodes |
| $\lambda$ | maximum # of keys held by a client |
| $\gamma$ | minimum frequency of any key |
| $q$ | vector of frequencies |
| $\mu$ | vector of means |
| $r$ | parameter for geometric distribution |
| $p$ | ratio of $t/\ell$ |

#### Table 1: Notation

- **Mean estimation.** The mean of the key $k$, $\mu_k$, is the mean of all values in $\mathcal{S}$ whose key is $k$. Thus, the mean is

$$\mu_k(\mathcal{S}) = \frac{\sum_{(k_{ij}, v_{ij}) \in \mathcal{A}_k} v_{ij}}{|\mathcal{A}_k|}.$$

We denote the vector of means for all keys by $\mu(\mathcal{S})$.

#### 3.2 Assumptions

We aim to compute statistics which satisfy the requirements of differential privacy, as described in Section 2.1. In the case of telemetry data, frequency and mean are very helpful statistics. For example, a browser vendor may want to calculate how many users have added known trackers to a blocklist [23]. We make the following assumptions about the data in the set $\mathcal{S}$.

- **Bounded values.** We assume that the domain of the values is bounded by some interval of size $2\mathcal{R}$. That is, every value $v_{ij} \in [x - \mathcal{R}, x + \mathcal{R}]$, for some $x$. We assume this to bound the sensitivity of statistics on the values.

- **Distinct keys.** We assume that each key appears in $\mathcal{S}_i$ at most once. This is done for simplicity because in the case where a client has multiple values for a single key, they could simply aggregate their data for said key beforehand.

- **Maximum size of $\mathcal{S}_i$.** We assume that each user has at most $\lambda$ key-value pairs. That is, $|\mathcal{S}_i| \leq \lambda$. This is assumed to tighten the analysis but is not restrictive because in the worst case $\lambda = n$.

- **Minimum frequency.** We assume that the frequency of each key is at least $\gamma$, for some $\gamma \geq 1$. This is to bound the sensitivity of our mean calculation. We require this term to obtain a data-independent bound on the sensitivity since both the frequency and mean are sensitive in our setting.

We consider user-level differential privacy where neighbouring datasets change by adding or removing a single user’s key-value pairs. Here, the users correspond to the clients which we discuss throughout the paper. We apply concepts from secure multi-party computation to remove the need for a single trusted server. In particular, the clients distribute their data between a small set of
untrusted servers, which collectively compute statistics over the data. We call these servers the computation nodes, and we require that there are \( t \geq 3 \) computation nodes. We make the following assumptions about the entities involved in the computation.

- **Anonymous channels.** We assume that there exist anonymous channels between clients and computation nodes. Section 5.3 discusses how our protocol functions in the shuffle model, where there is an intermediary party between the clients and the computation nodes [7, 24]. Our assumption of anonymous channels is a relaxation of the shuffle model.

- **Trust model.** For simplicity, we begin by describing our protocol in the semi-honest model, i.e., we assume that the clients and the computation nodes follow the protocol. Any party may however attempt to infer other information from the data that they observe (semi-honest model). Section 7 explains how to account for malicious clients and malicious computation nodes who do not follow the protocol.

- **Secure channels.** We assume pairwise secure and authenticated communication channels between the computation nodes. We also assume secure, anonymous communication channels between the clients and computation nodes, in the absence of a shuffler.

The final computed statistics may be published to all parties. Given this output and the information within their view throughout the protocol, the computation nodes should only be able to infer information that is bounded by our differential privacy guarantees. In terms of formally proving security, we aim to satisfy the definition of indistinguishable computationally differential privacy (IND-CDP) by Mironov et al. [39]. We discuss the security model in more detail in Section 7.

### 3.3 System Architecture

As described in the previous subsection, we have a set of clients and a set of computation nodes. First, the set of clients will send their data to the computation nodes. This is called the data collection phase. Then, in a computation phase, the computation nodes will compute some statistic and publish the output. In our solutions, we introduce an optional dummy generator who acts similarly to a set of clients and computation nodes. We make the following assumptions about the entities involved in the computation.

- **Anonymous channels.** We assume that there exist anonymous channels between clients and computation nodes. Section 5.3 discusses how our protocol functions in the shuffle model, where there is an intermediary party between the clients and the computation nodes [7, 24]. Our assumption of anonymous channels is a relaxation of the shuffle model.

We can distinguish between solutions which conceal keys and those which do not. Solutions that hide the keys incur a high computational cost to index the records into an array. So, as preliminary solutions, we only consider solutions that do not conceal keys. We begin with a naive solution that minimizes trust but requires high communication costs. The solution involves two types of participants: clients who own the data and computation nodes who calculate the statistic. We assume that there exists secure, anonymous channels between the clients and the computation nodes. We call this the full padding solution, and it is executed in the following two steps.

**Step 1: Padding, Share Generation, and Encryption.** Clients create a dummy key-value pair for each key not in their set, \( S_i \). For each key-value pair, the client applies a secret sharing scheme to generate \( \ell \) shares of the value. The clients forward the key-share pairs to the corresponding computation nodes.

**Step 2: Computation.** Upon receiving the data, the computation nodes perform an MPC protocol to compute the differentially private statistic for each key and reconstruct the final result. Since the computation nodes only see shares of the values, they cannot distinguish between real key-value pairs and dummies. Also, since every client sends a key-value pair, real or fake, for each key, it is impossible for the computation nodes to determine the true frequency of a key.

The obvious disadvantage to this protocol is the communication cost. Each client has \( n \) key-value pairs, for which they generate \( \ell \) shares, resulting in a communication cost of \( O(mn\ell) \). These costs are prohibitively expensive, so we consider methods of improving our approach. We remark that this cost is incurred to prevent the computation nodes from learning who has what data and the true frequency of the keys.

To improve upon this naive solution, we can leverage differentially private padding to mask the true frequency of the keys. This is accomplished by adding a semi-trusted dummy generator. The protocol, which we refer to as the one-sided dummy solution, requires the following three steps.

**Step 1: Share Generation.** For each key-value pair they own, the client applies a secret sharing scheme to the value and generates \( \ell \) shares. They forward this data to the corresponding computation nodes.
Step 2: Padding. The dummy generator generates "enough" dummy key-value pairs to satisfy differential privacy. They generate shares for these new values via the same secret sharing scheme used by the clients. The data is forwarded to the corresponding computation node.

Step 3: Computation. The computation nodes receive all of the data and then perform a multi-party computation protocol to compute the desired differentially private statistic. They collectively output the result.

This solution employs differential privacy in two ways: to bound the information learned as a result of adding fewer dummies and to output a noisy statistic. In the former, we require a randomized algorithm to add dummies; however, only adding key-value pairs creates a one-sided distribution and thus has a chance of being blatantly non-private.

There exists work that uses a similar approach of only adding fake records from one-sided distributions in different settings [5, 38, Appendix A], using a shifted and truncated two-sided geometric distribution to sample dummies and achieve approximate DP. In our setting, achieving a reasonable privacy parameter requires adding a large number of dummies. Although it improves on the full padding solution, there remains a high communication cost. We elaborate on the details of this protocol and the expected number of dummies and compare it to our final solution in the full paper [32, Appendix D].

Now that we have established the drawbacks of some preliminary solutions, we develop a solution which corrects these weaknesses. In our final solution, we reduce the number of dummies and improve the privacy guarantee from approximate to pure differential privacy. Clearly, adding one-sided noise would not allow for pure differential privacy. Intuitively, when only adding data and never removing data, the output will reveal an upper bound on the true number of data points. Thus, we want to simulate the effects of removing without actually removing data, as that would compromise the accuracy. We accomplish this using a technique we call selective multi-party computation. This involves forwarding each piece of data to a select subset of computation nodes. This simultaneously emulates subtractive noise and reduces the communication costs. We describe this in detail in the following section.

5 PROTOCOL DESCRIPTION

This section provides a detailed description of our new protocol designed to compute differentially private statistics over key-value data. Our protocol consists of two phases: data collection and multi-party computation. The main contribution of our work is in the data collection phase. Here, rather than having clients secret share their data among all computation nodes, they only send their shares to a select subset of computation nodes. This atypical approach enables us to improve efficiency and compute differentially private outputs. The multi-party computation phase adapts previously known techniques to account for this change in the data collection phase.

5.1 Data Collection Phase

Recall that we have $m$ clients, each denoted by an index $i \in [m]$ and $t \geq 3$ computation nodes. We also require an entity for dummy generation, which we assume is semi-honest. In Section 5.3, we discuss an alternative approach for dummy generation which does not require a single semi-honest entity. Our protocol aims to return a statistic over the multiset, $S$, while revealing only differentially private information about any particular client’s set of key-value pairs $S_i$. In the example of collecting telemetry data, this phase corresponds to the step where usage data is sent from users’ browsers to the computation servers.

The data flow is in this phase is similar to the one-sided dummy solution and is summarized in Figure 1.

1. Let $t \in \{2, \ldots, t-1\}$ be a (publicly-known) system parameter chosen ahead of time. For each key-value pair belonging to each client $i \in [m]$, the client $i$ chooses a subset of computation nodes of size $t$, uniformly at random, without replacement. The client generates $t$ shares of their value, one for each computation node they selected, using additive secret sharing. The client also generates $t$ shares of a flag with value 1. This is summarized in Figure 2. The client forwards each tuple consisting of the key, a share of the flag, and a share of the value, to the corresponding computation node.

2. Meanwhile, for each key $k$ in the set $K$, the dummy generator samples $x_k$ from a geometric distribution with parameter $r$. The two-sided geometric distribution is commonly used as the discrete version of the Laplace mechanism [3, 27]. We use the standard one-sided geometric distribution here to avoid having to shift and truncate the distribution. The dummy generator generates $x_k$ dummy key-value pairs with key $k$, flag 0, and value 0. They repeat the process performed by the clients by choosing a subset of size $t$ of computation nodes uniformly at random and generating $t$ shares of each dummy value and flag. The dummy generator forwards the dummy data to their respective computation nodes.

3. Upon receiving the anonymized data, the computation nodes perform a multi-party computation protocol, based on which statistic is desired, and output the result.

The main difference between this data collection phase and a typical data collection phase in MPC is in step (1) where clients only send their data to a subset of computation nodes. Although conceptually simple, this effectively acts as subtractive noise which will complement the additive noise from the dummy key-value pairs in step (2), thereby ensuring pure differential privacy.

5.2 Multi-party Computation Phase

In this section, we describe how the computation nodes process the data once it is received. To compute statistics on the data, they must be able to perform basic operations, such as multiplication and...
division, and add noise to perturb the final result. A protocol by Wu et al. [51] allows for secure multi-party addition of Laplace noise. We refer readers to the original paper for details of the protocol and proofs of security. Specifically, the protocols allow us to obtain shares of a random variate drawn from a univariate cumulative distribution function. Additionally, we can use a protocol from Catrina and Saxena to perform division [9, §3.4]. In the following descriptions, we let \(|x|\) denote that the value of \(x\) is distributed among the parties. We remark that the secret sharing scheme used in the multi-party computation step to generate noise need not be the same additive secret sharing scheme used during data collection; although, a change in schemes may require a resharpening step to update the shares accordingly. Finally, we note that due to the selective way in which shares are distributed to computation nodes, the types of statistics which may be computed are restricted to those which can be derived from initially taking the sum of all shares. To compute statistics other than those which we describe below, one might consider encoding the data in various ways [12] to achieve compatibility with the selective step of the data collection phase.

5.2.1 Frequency estimation. As per the data collection phase, each computation node receives a subset of shares of key-value pairs. The computation nodes see tuples which contain a key, a share of the flag, and a share of the value. To compute the frequency estimation, the computation nodes compute over the shares of the flag. Recall that real key-value pairs will have a flag with value 1 and dummies will have a flag with value 0 (and this value is not revealed to the computation node since it is secret shared). For each key in \(K\), each computation node executes Protocol 1 with \(\Delta = \lambda\), where \(\lambda\) denotes the maximum number of distinct keys held by any client. The value of \(\lambda\) is publicly known. Essentially, Protocol 1 enables the computation nodes to compute the sum of all flags corresponding to some key \(k\) and add noise to the result.

Protocol 1 Frequency estimation

Input. A set of triples \((k_{ij}, f_{ij}, s_{ij})\). A target key \(k\). A sensitivity \(\Delta\).
1: Compute the sum \(f_i = \sum_{k_{ij}=k} f_{ij}\). These are shares of the frequency \([q_k]\).
2: Generate a random variate, \([\xi]\), drawn from \(\text{Lap}(\Delta/\epsilon_F)\) by invoking Protocol [51, §7] with \(C(t) = \frac{\Delta}{\epsilon_F} \int_{-\infty}^{t} \exp\left(-\frac{\xi(s)}{\epsilon_{F}}\right) ds\).
3: Return \([q_k] + [\xi]\).

Correctness. Before the noise addition, the \(i\)th computation node has a value \(f_i\). The sum of all of these values is

\[
\sum_{i=1}^{\ell} f_i = \sum_{i=1}^{\ell} \sum_{j, k_{ij}=k} f_{ij} = \sum_{k, r} f_{ij}.
\]

The sum of all shares of the flags must be the sum of the flags themselves by the properties of the additive secret sharing scheme. As real data points have flag equal to 1 and dummies have flag 0, this must be the frequency of key \(k\), \(q_k\). By the properties of Protocol [51, §7], the computation nodes ultimately obtain the frequency of the key \(k\) with Laplace noise drawn from \(\text{Lap}(\Delta/\epsilon_M)\) where \(\Delta = \lambda\).

5.2.2 Mean estimation. To compute a noisy mean estimation, the computation nodes execute Protocol 2 with \(\Delta = \frac{2R}{\epsilon}\), where \(\lambda\) is the maximum number of distinct keys held by any client, \(\gamma\) is the minimum frequency of any key, and \(2R\) is the size of the interval bounding the values. To summarize Protocol 2, the computation nodes compute the frequency of the key \(k\), sum the values corresponding to \(k\), divide by the frequency to get the mean, and add noise. Let \(S_k\) denote the sum of all values corresponding to key \(k\).

Protocol 2 Mean estimation

Input. A set of triples \((k_{ij}, f_{ij}, s_{ij})\). A target key \(k\). A sensitivity \(\Delta\).
1: Compute the sum \(f_i = \sum_{k_{ij}=k} f_{ij}\). These are shares of the frequency \([q_k]\).
2: Compute the sum \(s_i = \sum_{k_{ij}=k} s_{ij}\). These are shares of the sum of the values \([S_k]\).
3: Invoke Protocol [9, §3.4] with input \([S_k]\), \([q_k]\) to obtain \([S_k]/[q_k]\).
4: Generate a random variate, \([\xi]\), drawn from \(\text{Lap}(\Delta/\epsilon_M)\) by invoking Protocol [51, §7] with \(C(t) = \frac{\Delta M}{\epsilon_{F}} \int_{-\infty}^{t} \exp\left(-\frac{\xi(s)}{\epsilon_{F}}\right) ds\).
5: Return \([S_k]/[q_k] + [\xi]\).

Correctness. Steps 1-2 accurately compute the frequency of the key \(k\) and the sum of all the values corresponding to key \(k\), by the properties of the additive secret sharing scheme. This is also due to the fact that the dummies have a value of 0, so this does not affect the sum. Step 3 computes the true mean. By the properties of Protocol [51, §7], the computation nodes obtain the mean of the values corresponding to key \(k\) with Laplace noise drawn from \(\text{Lap}(\Delta/\epsilon_M)\) where \(\Delta = \frac{2R}{\epsilon}\).

5.3 Implementation Considerations

Anonymous Channels. In our protocol description, we assume that there exist anonymous channels between the clients and the computation nodes. Although this can be accomplished by using a variety of anonymity systems [18], we discuss one approach here. We may rely on a semi-trusted party to act as an intermediary between the clients and the computation nodes. This is similar to the shuffle model in DP [7], where an intermediary shuffler performs anonymization, shuffling, thresholding, and batching. The presence of the shuffler induces a privacy amplification which allows each client to add less randomness [24]. Our protocol does not require an equally strong trust assumption on the shuffler. Rather, we can have clients encrypt their data with the public keys of the computation node that is responsible for receiving each share of data. Then, the clients forward their data to the shuffler, which removes any user-specific metadata, before forwarding the shares to the respective computation nodes in random order. This ensures that data is anonymized from the perspective of the computation nodes without revealing the data to the shuffler. The only information that the shuffler learns is how many key-value pairs each client holds. Ultimately, our use of this intermediary party requires less trust than the trust required in the shuffle model. Given that the shuffle model is used in practice today by organizations such as Google [24], it is reasonable to assume the existence of anonymous channels. Collusion between the shuffler and the server is not permitted in the shuffle model. Similarly, collusion between the computation
we address the case where dummy generators behave maliciously.

\[ \epsilon \]

view of 10 keys. We set \( p = 2/3 \) and \( r = 0.4 \) and plot the distribution with and without dummies.

nodes and the nodes implementing the anonymous channel is not permitted in our protocol.

In addition to anonymizing data, the shuffler may also be responsible for generating the dummy values. This intermediary party can fulfill two goals, while we retain minimal trust assumptions.

**Dummy Generation.** Another requirement in the protocol description is a semi-trusted party to generate dummies. A single party that generates dummies poses a threat if that party colludes with the computation nodes. If the total number of dummies generated for key \( k \) is known to a computation node, they can derive the expected number of dummies it received for said key. For \( x_k \) dummies in total, one computation node receives \( px_k \) dummies on average. An estimate of the true frequency of the key is derived by deducting \( px_k \) from the total number of key-value pairs it receives.

To circumvent this issue, one alternative is using \( d \) parties to generate the dummies. Specifically, for each key \( k \), \( d \) parties independently sample \( x_k \) from a geometric distribution with parameter \( r \) and generate \( x_k \) dummies for key \( k \). Collusion between dummy generators and clients does not affect the security of the protocol. Hence, for each key, dummy generators can be chosen from amongst the clients to eliminate the requirement for extra entities in the protocol. While the total number of dummies that is generated for each key is roughly \( d \) times higher than required, this approach distributes trust and prevents a single point of failure. Moreover, our evaluation shows that the expected number of dummies with a single dummy generator is very small (less than 3). Hence, this approach adds a small overhead to the overall protocol. In Section 7, we address the case where dummy generators behave maliciously.

**Offline Noise Generation.** The random noise generated in line 2 of Algorithm 1 and line 4 of Algorithm 2 depends only on the parameters of the protocol, not the data provided by the clients. Hence, these steps, which constitute the bulk of the runtime, can be performed offline before the clients provide their data. This reduces the perceived latency from the client’s perspective.

6 PRIVACY

In this section, we analyze the privacy of our protocols. First, we formally define the privacy leakage as a function and show that it is \( \epsilon_L \cdot \text{DP} \). We find that limiting the view of each computation node by randomly selecting which key-value pairs a node sees is equivalent to drawing noise from a binomial distribution. We visualize this distribution in Figure 3 (Selective MPC - No Dummies). In expectation, each node will see \( 1/3 \) of the keys (2/3 of 10 in the figure), the center of the binomial distribution. Despite being a smooth, two-sided distribution, the binomial selection alone does not satisfy pure DP. That is because simply distributing the key-value pairs ensures it is impossible for a computation node to observe more than the true count of keys (the probability of 11 or 12 keys is zero in Figure 3). For this reason, we add a small number of dummy key-value pairs following a geometric distribution. We can see that this shifts the distribution to the right and avoids clipping the right tail in Figure 3. We formalize our unique combination of additive noise from a geometric distribution and subtractive noise from a binomial distribution and prove that it satisfies pure DP. We show in the full paper [32, Appendix B] that our frequency estimation satisfies \( \epsilon_F \cdot \text{DP} \) and that our mean estimation satisfies \( \epsilon_M \cdot \text{DP} \). As a result, the total \( \epsilon \) spent for the frequency estimation protocol is \( \epsilon_F + \epsilon_L \) by naive composition. Similarly, the privacy budget for the mean estimation is \( \epsilon_M + \epsilon_L \). The mean and frequency computation can be executed simultaneously, leading to a total cost of \( \epsilon_F + \epsilon_M + \epsilon_L \).

Let \( L \) be the algorithm that distributes the key-value pairs and dummies among the \( t \) computation nodes. Let \( L_k \) denote the corresponding algorithm for a single key \( k \). For the sake of the privacy analysis, we treat \( L \) as an ideal functionality: we assume it is run by a trusted party and consider only the output of the algorithm in our privacy analysis. An arbitrary output of \( L \) is an observation of a single computation node, which we denote by \( Z \). For simplicity, we assume that the nodes do not collude with each other and address the case of collusion separately. Recall that upon receiving data, the computation nodes obtain a set of tuples. The shares reveal nothing about the corresponding values and the computation nodes do not know the origin of the data. Therefore, \( Z \) can be described as a histogram of keys observed by the computation node. We show that the output of this algorithm is bounded by differential privacy. The input to \( L \) is the list of all key-value pairs held by all users. We say that two inputs, \( S \) and \( S' \), are *neighbouring* if they differ in at most the addition or deletion of \( \lambda \) entries (user-level privacy).

We represent the ground truth of this algorithm as the vector \( q(S) \), i.e., the true histogram of the keys. Let \( q_k(S) \) represent the count of key \( k \) on the input \( S \). We assume that each user has at most one value for each key and thus \( \forall k \in [n], |q_k(S) - q_k(S')| \leq 1 \). The ideal functionality of algorithm \( L \) can be described as follows.

1. Compute the noisy histogram \( Y \) for a given input \( S \) by computing \( Y_k = q_k(S) + \eta \) where \( \eta \in \mathbb{N} \) is sampled i.i.d from a geometric distribution with parameter \( r \).
2. Then \( Z \) is obtained by sampling from the histogram \( Y \), following our protocol. This selective part of our procedure is equivalent to sampling \( \|Y\|_1 \) times from a binomial distribution with parameter \( p = t/f \).

We denote the probability mass function (PMF) of the binomial distribution\(^\dagger\) with parameters \( a \) and \( p \) as

\[
B(z; a, p) = \binom{a}{z} p^z (1 - p)^{a-z},
\]

\(^\dagger\)We follow the standard convention that \( B(z; a, p) = 0 \) whenever \( a < z \).
and the PMF of the geometric distribution as
\[ G(z; r) = (1 - r)^z r. \] (5)

Then, the PMF of \( L_k(S) \) is defined as
\[ \Pr[L_k(S) = Z_k] = \sum_{v=0}^{\infty} G(v; r) \mathbb{B}(Z_k; q_k(S) + v, p). \] (6)

Finally, the PMF of \( L(S) \) is
\[ \Pr[L(S) = Z] = \prod_{k=0}^{n} \Pr[L_k(S) = Z_k]. \] (7)

where this equality follows from the fact that \( L \) applies randomness independently for each key. Using this PMF we can prove the following theorem.

**Theorem 6.1.** The algorithm \( L \) satisfies \( \epsilon_L \)-DP with
\[ \epsilon_L = \lambda \ln \left( \max \left( \frac{1}{1 - r}, \frac{1}{1 - p} + 1 - r \right) \right). \] (8)

We defer the proof of Theorem 6.1 to the full version [32, Appendix A]. The proof relies on the assumption that computation nodes only see a subset of the keys. This simulates adding noise from a binomial distribution. A natural question to ask is how the privacy guarantee degrades under collusion. In the case where computation nodes collude, the privacy degrades.

**Claim 6.1.** Let \( c \) be the collusion threshold which is the maximum number of computation nodes that can collude (Theorem 6.1 assumes \( c = 1 \)). Then (8) becomes
\[ \epsilon_L = \lambda \ln \left( \max \left( \frac{1}{1 - r}, \frac{\ell}{\ell c + 1} + 1 - r \right) \right). \] (9)

We note that we consider the minimal value of \( t = c + 1 \). This is because values of \( t \) less than \( c + 1 \) may allow colluding computation nodes to reconstruct the values.

The result in Equation (9) follows from a similar argument to the proof of Theorem 6.1, substituting \( p \) with the maximum probability that a node observes a specific share in the presence of collusion. The probability of a computation node observing a single share, either directly or through the nodes it colludes with, is \( 1 - \binom{\ell - c}{c + 1} / \binom{\ell}{c + 1} \). In Figure 4, we show how collusion impacts the privacy budget compared to the baseline given by Theorem 6.1. In the baseline, represented by the dotted line, we set \( t = 2, t = 20 \), and choose \( r \) such that it minimizes \( \epsilon_L \). We plot Equation (9) as a function of \( c \) for \( t = 20 \) and choose \( r \) such that it minimizes \( \epsilon_L \).

**Outputting Exact Answers.** We note that our protocol could be used to output the exact frequency or mean. We would simply follow the selective MPC protocol without adding the final noise (the dummies have no effect on the accuracy). In this case, \( \epsilon_L \) is the only privacy cost incurred. This is similar to output-constrained DP [31] where the computation nodes do not learn more than a DP amount about the clients data, excluding what they learn from the final result. In general, publishing exact statistics should be avoided as it allows for reconstruction attacks.

### 7 SECURITY

We start by considering the semi-honest model and later extend to the malicious model. Our protocol consists of two main parts: the data collection and the multi-party computation protocols. For simplicity of the implementation, we are working under the assumption that there exist anonymous, secure channels between the clients and the computation nodes. The clients only see their own data and the final output of the differentially private computation. Our goal is that an adversary learns no more than the differentially private leakage from algorithm \( L \) and the final output of the differentially private computation.

#### 7.1 Proof of security

To analyze the security of the protocol, we rely on a definition of indistinguishable computationally differential privacy (IND-CDP) by Mironov et al. [39]. The following definition is for two-party computation. The security definitions which we present encompass the entire protocol, i.e. both the data collection and computation phases.

**Definition 7.1 (IND-CDP-2PC [31]).** A two-party protocol \( \Pi \) for computing function \( f \) satisfies \( (\epsilon_A(\lambda), \epsilon_B(\lambda)) \)-indistinguishable computationally differential privacy (IND-CDP-2PC) if \( \text{VIEW}_{\Pi}(D_A, D_B') \) satisfies \( \epsilon_B(\lambda) \)-IND-CDP, i.e., for any probabilistic polynomial time (in \( \lambda \)) adversary \( \mathcal{A} \), for any neighbouring databases \( (D_B, D_B') \),
\[ \Pr[\mathcal{A}(\text{VIEW}_{\Pi}(D_A, D_B)) = 1] \leq \exp(\epsilon_B) \cdot \Pr[\mathcal{A}(\text{VIEW}_{\Pi}(D_A, D_B')) = 1] + \text{negl}(\lambda). \]

Likewise for \( B \)'s view of any neighbours \( (D_A, D_A') \) and \( \epsilon_A \).

For our purposes, we extend the definition from two-party to multi-party computation and we let \( \epsilon = \epsilon_A = \epsilon_B \) for convenience. Intuitively, the following definition ensures that an adversary \( \mathcal{A} \) cannot distinguish neighbouring databases \( D, D' \) from their view, and the same applies to every other party.

**Definition 7.2 (IND-CDP-MPC).** A multi-party protocol \( \Pi \) for computing function \( f \) satisfies \( \epsilon(\lambda) \)-indistinguishable computationally differential privacy (IND-CDP-MPC) if for every probabilistic polynomial time (in \( \lambda \)) adversary \( \mathcal{A} \) with input \( D_A \), and for neighbouring
that are maliciously secure under the assumption that there exists malicious parties have been discussed in the literature [15, 34]. We require that the computation nodes act semi-honestly in the computation phase, we require that the computation nodes act semi-honestly. In the data collection phase, we require that the computation nodes act semi-honestly. Our security proof assumed that we work in the semi-honest model, but would not compromise the correctness of the output. In this paper, we have discussed how our protocol can account for semi-honest computation nodes and a minority of misbehaving computation nodes; however, we are not limited to these two settings. Since the multi-party computation phase is a simple adaption of previously known techniques in MPC, it is straightforward to substitute an MPC protocol with other guarantees. For example, in a scenario where one is concerned about the reconstruction of invalid or incorrect results, we can use an MPC protocol which ensures robustness. Similarly, we could use a protocol which is secure against Byzantine failures if we are concerned about computation nodes failing. The protocol which is ultimately chosen depends on the requirements of the system in practice, but is not limited in any way by our construction.

Next, we consider the possibility of malicious clients. We can prevent malicious clients from submitting invalid data by having the computation nodes perform validation checks on the shares they receive. Note that the validity check is conducted in MPC by the computation nodes that receive each piece of data, not a majority of misbehaving computation nodes; however, we are not limited to these two settings. Since the multi-party computation phase is a simple adaption of previously known techniques in MPC, it is straightforward to substitute an MPC protocol with other guarantees. For example, in a scenario where one is concerned about the reconstruction of invalid or incorrect results, we can use an MPC protocol which ensures robustness. Similarly, we could use a protocol which is secure against Byzantine failures if we are concerned about computation nodes failing. The protocol which is ultimately chosen depends on the requirements of the system in practice, but is not limited in any way by our construction.

7.2 Extending to the malicious model.

Our security proof assumed that we work in the semi-honest model, in which all parties follow the protocol. We now consider how our protocol may function if this assumption is relaxed. First, consider the possibility of malicious computation nodes. In the data collection phase, we require that the computation nodes act semi-honestly so that they are unable to arbitrarily add their own input. We can relax this assumption in the multi-party computation phase. Methods for extending MPC protocols to be secure in the presence of malicious parties have been discussed in the literature [15, 34]. We can apply the same strategies to our own work, using protocols that are maliciously secure under the assumption that there exists an honest majority [6, 15, 19]. This prevents nodes from adding arbitrary noise during the computation phase which may affect the results. In this scenario, we can adjust our bound \( b_N \) on the number of computation nodes that the adversary may control to be the minimum of \( t - 1 \) and \( (t - 1)/2 \). This ensures that the adversary cannot control a majority of participants in the multi-party computation. As with any other multi-party computation protocol which assumes an honest majority, if the assumption of an honest majority is not satisfied then there are no guarantees on the privacy or correctness of the output. In this paper, we have discussed how our protocol can account for semi-honest computation nodes and a minority of misbehaving computation nodes; however, we are not limited to these two settings. Since the multi-party computation phase is a simple adaption of previously known techniques in MPC, it is straightforward to substitute an MPC protocol with other guarantees. For example, in a scenario where one is concerned about the reconstruction of invalid or incorrect results, we can use an MPC protocol which ensures robustness. Similarly, we could use a protocol which is secure against Byzantine failures if we are concerned about computation nodes failing. The protocol which is ultimately chosen depends on the requirements of the system in practice, but is not limited in any way by our construction.

Finally, suppose that the third party generating dummies is malicious. On one hand, a malicious dummy generator may submit additional data points to attempt to influence the final result. This poses no more of a threat than the case where malicious clients submit false data points. This is a potential vulnerability to any system which collects data from clients and is not prevented by security against malicious adversaries. In such a scenario where the dummy generator is malicious, it is also possible that they submit no dummies or submit an incorrect number of dummies. This would compromise the differential privacy guarantees on the leakage of our protocol, but would not compromise the correctness of the output. It is difficult to ensure that dummy generation by a single party occurs correctly. However, recall that we have discussed alternatives to dummy generation which provide more distribution of trust in Section 5.3. Suppose we are using the method described...
in Section 5.3 where several clients are responsible for generating dummies. Then, we must ensure that the adversary cannot control all of the clients generating dummies. If we assume that the adversary can control at most \( b_C \) clients, then we can let \( b_C + 1 \) clients generate dummies, which ensures that we still achieve differential privacy. This does not affect the security proof because the adversary is unable to control at least one party generating dummies. Ultimately, distribution of trust allows us to consider the presence of malicious parties while achieving secure dummy generation.

## 8 Evaluation

This section provides a theoretical evaluation of our protocol in terms of accuracy, communication, and computation cost. We then compare this with other methods for computing statistics on key-value data. Finally, we present our experimental results to demonstrate the practicality of our protocol.

### 8.1 Our Protocol

For the purpose of comparison, we analyze our protocol in the case where \( t = 2 \). This only affects the communication cost during the data collection phase and is a reasonable choice in practice as it minimizes \( \epsilon_L \).

**Communication.** The communication conducted in our protocol can be decomposed into the communication from the clients (and dummy generator) to the computation nodes, i.e., data collection, and the communication between computation nodes. For comparison, we focus mainly on client-to-server communication. The communication between computation nodes is specific to solutions using MPC and depends largely on parameters of the MPC protocol, such as the precision of the noise generated. These are chosen by a system administrator depending on the context of the situation. We provide some experimental results for communication between computation nodes in Section 8.3. In this section, we elaborate on the communication between clients and servers. First, \( 2|S| \) elements are sent from the clients to the computation nodes because each client generates \( t = 2 \) shares for each of the key-value pairs it owns. The number of dummies added for each key follows a geometric distribution with an expected value of \( \frac{1}{2} \), which means \( \frac{1}{2} \) dummies are generated on average. For each dummy, \( t = 2 \) shares are generated. Therefore, in expectation, \( \theta(|S| + \frac{1}{2}) \) shares are sent from the clients (and dummy generator) to the computation nodes.

**Computation.** We define the computation cost of our protocol to be the number of multiplications in the MPC protocol. For each key, the noise generation (and division in the case of mean estimation) is run to derive the result. The complexity of these operations depends on the chosen protocol and parameters related to their implementation, such as precision. Hence, we consider the complexity to be constant with respect to each key. Therefore, the overall number of multiplications in the MPC protocol across all keys is \( O(n) \).

**Accuracy.** Finally, we consider the accuracy of the output of our protocol. Our protocol implements an algorithm with output identical to that of the central model, so it suffices to consider the accuracy of the Laplace mechanism, which is well known. We begin by evaluating the accuracy of our frequency estimation protocol denoted by \( F(S) \). Using a tail bound on the Laplace distribution along with the union bound as shown in Theorem 3.8 of Dwork and Roth [21], we obtain the following accuracy guarantee \( \forall \beta \in (0, 1] \).

\[
\Pr[||q(S) - F(S)||_{\infty} \geq \ln(n/\beta) \cdot (\lambda/\epsilon_F)] \leq \beta. \tag{10}
\]

Similarly, for the mean estimation protocol, with output denoted \( M(S) \), we obtain that \( \forall \beta \in (0, 1] \).

\[
\Pr[||\mu(S) - M(S)||_{\infty} \geq \ln(n/\beta) \cdot (2R\lambda/\epsilon_M)] \leq \beta. \tag{11}
\]

To allow for comparison of different methods, we simplify the above expression and obtain an approximation for the error of our protocols using the standard deviation. For a single key, we compute that the error is approximately \( O(\Lambda/\epsilon) \) where \( \Lambda \) is the sensitivity of the specific protocol. For simplicity, we will only consider the error for a single key’s frequency or mean and note the case of all keys can easily be obtained using a union bound. For our analysis, we assume all clients hold this key. Taking into account the privacy budget from the leakage function, we conclude that our solution achieves an approximate error bound of \( O(\Lambda/\epsilon - \epsilon_L) \).

### 8.2 Comparison with Other Work

We now compare the communication, computation, and accuracy of our protocol with a naive MPC solution that conceals and secret shares keys, existing local DP solutions (PrivKVM [52], PCKV [30]), and the central model. The results of our comparison are summarized in Table 2. Although there has been no prior work on key-value data in the shuffle model, we discuss potential solutions in this setting in Section 10.

|                | Client-to-Server Communication | MPC Complexity | Error |
|----------------|-------------------------------|----------------|-------|
| Our Protocol   | \( \theta(|S| + \frac{1}{2}) \) | \( O(n) \)  | \( O(\Lambda/\epsilon - \epsilon_L) \) |
| Central DP     | \( O(|S|) \)                  | \( O(\Lambda/\epsilon) \) |
| Naive MPC      | \( O(|S|t) \)                 | \( O(|S|n) \)  | \( O(\Lambda/\epsilon) \) |
| PrivKVM        | \( O(m \log n) \) *           | \( O(\sqrt{m\Lambda}/\epsilon) \) |
| PCKV           | \( O(m \log n) \)             | \( O(\sqrt{m\Lambda}/\epsilon) \) |

Table 2: Comparison of client-server communication, computation (number of multiplications), and error (approximate bound). *Includes a multiplicative factor equal to the number of iterations which we exclude.
$O(m \log n)$. The curator calculates the desired statistics (frequency and mean) by iterating over the data they received and calibrating the result. This computation cost is negligible in comparison to an MPC protocol. Finally, PrivKVM utilizes the Harmony protocol [41] and achieves an approximate error bound of $O(\sqrt{m}/\epsilon)$. We remark that the accuracy depends on the number of iterations $c$ carried out in the PrivKVM protocol. We follow the approach taken by the authors and treat $c$ as a constant in our analysis. Additionally, there is a sampling error due to the fact each client only sends one of their keys pairs which we omit from this analysis.

**PCKV.** This protocol improves upon PrivKVM. To estimate the mean, each client sends either a key-value pair or a vector equal to the size of the key domain. For our comparison, we will consider the former which has a lower overall communication complexity of $O(m \log n)$ across all users. The protocol does not require interaction, so there is no multiplicative factor for the number of rounds. In terms of accuracy, all of the improvements influence the error by a constant amount. While this showed significant improvements in experimental evaluations, it has no asymptotic advantage over PrivKVM.

### 8.3 Experimental Results

To evaluate the practicality of our protocol, we compute a series of benchmarks. Recall that our protocol consists of two phases: the data collection and multi-party computation phase. The bottleneck of the data collection phase is the communication of data from all clients. Collecting shares is as simple as sending a single message (to several nodes). This overhead is shared by all other related work to several nodes. To evaluate the practicality of our protocol, we compute a series of value pairs (we evaluate exactly how many in [32, Appendix D.4]). Moreover, the bulk of the overhead in the data collection phase consists of the time required for coordination among the clients, i.e., the time for all clients to send their data, which is not reflected in experiments. The computational overhead of our system in the data collection phase is the dummy generation which incurs a negligible overhead equivalent to sharing a few additional key-value pairs (we evaluate exactly how many in [32, Appendix D.4]). The most expensive part of our protocol is the MPC phase which requires tightly synchronized nodes. Thus, we implement this phase using the noise generation protocol from Wu et al. [51] and provide benchmarks for the run time and communication. We investigate the effects of the number of computation nodes, network latency, and the number of keys. Our implementation for the evaluation is publicly available. ²

**Setup.** We use the MP-SPDZ [34] framework to implement Protocol [51, §7] from Wu et al. for generating a Laplace random variate. We simulate the frequency or mean estimation protocol by sampling from the Laplace distribution and adding it to an arbitrary result (either a constant or a division of constants). We use the Shamir protocol, implemented in the MP-SPDZ framework. ³ Using this implementation, we observed that the underlying technique for generating Beaver triples switches when the number of computation nodes exceeds a threshold. The default threshold was set too high for our use case, resulting in sub-optimal run time results. We set the threshold to ten computation nodes to optimize run time results; however, there remains a noticeable change in the communication results.

**Number of Comp. Nodes.** First, we measure how our protocol scales with the number of computation nodes. Since our protocol can be parallelized over the number of keys, we only consider a single key for this benchmark. To create multiple parties, we use up to 30 Amazon EC2 T2 instances. Each machine has four cores, 16 GB of memory, and is located in the same Ohio data center.

In Figure 5, we plot the average run time over 100 runs with 95% confidence intervals as the number of nodes increases for frequency and mean estimation in the semi-honest model and mean estimation in the malicious model. For context, we also plot the corresponding $\epsilon_L$ for each $\ell$. We see roughly quadratic growth in the run time, which is consistent with the asymptotic complexity of MPC. In the mean estimation protocol, for 30 nodes (where $\epsilon_L = 0.512$) we observe 0.88 and 6.36 seconds for the semi-honest and maliciously secure protocol, respectively. Moreover, for 5 nodes (where $\epsilon_L = 0.758$) the protocol runs in 0.07 and 0.83 seconds in the semi-honest and malicious model, respectively. Figure 6 shows the corresponding total communication. The dip around ten nodes

---

²https://github.com/data61/MP-SPDZ
³https://git.uwaterloo.ca/r5akhava/selective-mpc
comes from the change in protocol mentioned in the previous section. In the mean estimation protocol, we observe 229.45 MB and 2976 MB for 30 nodes in the semi-honest and malicious model and 8.77 MB and 36.11 MB for five nodes in the semi-honest and malicious model. We conclude that our protocol scales acceptably to realistic numbers of computation nodes.

**Different Data Centers.** Having all computation nodes in the same data center is not always possible in practice. Thus, in this experiment, we consider the effects of network latency from having computation nodes in different locations. To do this, we consider three different scenarios: local, remote, and distant. In the local scenario, all nodes are in the Ohio data center. In the remote setting, half of the nodes are in the Ohio data center and half in Northern California. Finally, the distant scenario has half of the nodes in the Ohio data center and the other half in Frankfurt, Germany. We consider a single key and conduct this experiment for 3, 6, 10, and 20 nodes. Furthermore, we experiment with a semi-honest and malicious secure MPC protocol. The results are given for 100 runs in Table 3. We observe reasonable run times even for a high number of computation nodes. Despite increased latency in wide area networks, the total run time remains under six seconds.

**Varying Key Domain Sizes.** Furthermore, we investigate the effects of multiple keys. Recall that each key is treated separately in our protocol, so we parallelize the computation over the keys. To do this, we use the built-in threading functionality in MP-SPDZ. We use five computation nodes (which achieves \( \epsilon_L = 0.758 \)), each running an Amazon EC2 C5 instance with 64 cores and 128 GB of memory. We consider key domain sizes of 10, 100, 1000, and 10,000. As with the previous experiment, we use local, remote, and distant network setups. Results are given for 40 runs in Table 4. We observe reasonable run times that increase approximately linearly after the number of keys exceeds the number of cores.

**Input Validation.** We implement and examine the cost of validating the client’s input in [32, Appendix E]. Our experiments suggest that input validation does not impose an impractical burden on the overall protocol.

### Table 3: Run time (in seconds) for one key. Measurements are means with standard deviations in parentheses. For each configuration the first row and second row use a semi-honest and maliciously secure MPC, respectively.

| # Nodes | Local | Remote | Distant |
|---------|-------|--------|---------|
| 3       | 0.037 (0.002) | 2.032 (0.020) | 4.50 (0.02) |
| 6       | 0.077 (0.004) | 2.056 (0.015) | 4.80 (0.02) |
| 10      | 0.237 (0.047) | 0.15 (0.11) | 5.85 (0.27) |
| 20      | 0.573 (0.064) | 3.18 (0.14) | 5.73 (0.22) |

### Table 4: Run time (in seconds) for many keys with 5 computation nodes, which corresponds to \( \epsilon_L = 0.758 \). Measurements are means with standard deviations in parentheses. For each configuration the first row and second row use a semi-honest and maliciously secure MPC, respectively.

| # Keys | Local | Remote | Distant |
|--------|-------|--------|---------|
| 10     | 0.075 (0.003) | 2.58 (0.09) | 5.015 (0.08) |
| 100    | 0.58 (0.03) | 3.55 (0.11) | 6.38 (0.14) |
| 1000   | 2.38 (0.03) | 32.52 (0.87) | 63.6 (1.14) |
| 10000  | 26.64 (0.09) | 323.65 (0.23) | 627.3 (0.39) |

A related discussion on Theorem 6.1 would improve this bound. For three fixed values of \( \ell \) based on the result of Theorem 6.1. The lower bound on \( \epsilon_L \), which arises from the max function in the formula for \( \epsilon_L \), can be seen in the graphics. Specifically, for a given \( \ell \), the smallest achievable \( \epsilon_L \) can be derived as follows.

\[
\min_{\ell} \epsilon_L = \min_{\ell} \left( \ln \left( \max \left( \frac{1}{1 - r}, \frac{1}{1 + \frac{1 - r}{\ell}} \right) \right) \right)
\]

\[
= \ln \left( \frac{2(1 - 2/\ell)}{1 + 4(1 - 2/\ell)^2 - 1} \right)
\]

which is achieved at \( \ell = 2 \) and

\[
r = 1 - \sqrt{1 + 4(1 - 2/\ell)^2 - 1} \]

Figure 7b graphs the smallest achievable privacy of the leakage function based on Equation (13). The minimum of Equation (13) is obtained at \( \ell = \infty \), albeit impossible to achieve in practice, which gives a lower bound on \( \epsilon_L \). In particular, if \( \Delta = 1 \) for simplicity, then

\[
\epsilon_L > \ln \left( \frac{2(1 - 2/\ell)}{1 + 4(1 - 2/\ell)^2 - 1} \right) = \ln \left( \frac{2}{\sqrt{5} - 1} \right) = 0.4812.
\]

Such a lower bound may be unacceptable in certain scenarios. Tightening the analysis of Theorem 6.1 would improve this bound.

### 10 RELATED WORK

**Differentially Private Statistics over Key-Value Data.** Our work is most closely related to prior literature computing key-value analytics under the LDP model. The first work in this space was PrivKV
work, we take the approach of simulating the central trust model using MPC to avoid any dependence on the number of clients by introducing the shuffle model of DP. The shuffle model lies in between the local and central models [4, 11]. Thus in our work, we take the approach of simulating the central trust model using MPC to avoid any dependence on the number of clients by adding randomness in a central manner.

Figure 7: (a) Leakage privacy as a function of r for t = 2. (b) Lower bound on leakage privacy.

[52]. Ye et al. modify a randomized response based protocol [41] to better maintain the relationships between the keys and values. They apply their baseline technique in an iterative manner, called PrivKVM, where each iteration gets closer to the true mean. To reduce communication costs, PrivKVM requires each client only send a single randomly sampled key-value pair. This sampling comes at the cost of additional sampling error in the final result. Follow-up work from Gu et al. [30] introduced PCKV, which improves upon PrivKVM. They apply an advanced sampling procedure to enhance utility over the sampling done by PrivKVM. They also require only a single iteration and provide a tighter analysis of the privacy budget consumption. Sun et al. [46] investigate alternate perturbation techniques under the PrivKV framework and conditional analysis of key-value data.

While this line of work has made substantial advancements to the computation of key-value statistics in the local trust model, it suffers an inherent limitation. In any LDP solution, the randomness is introduced to each value before aggregation and thus the error compounds in the final result. As we show in our evaluation, this increases the error by a factor of $O(\sqrt{m})$. Bittau et al. [7] investigate reducing the effects of compounding noise by introducing the shuffle model of DP. The shuffle model lies in between the local and central models, with a weaker trust assumption than the central model but improved utility guarantees over the local model. In this model, a semi-trusted shuffler lies between the participants and the untrusted curator. The shuffler introduces anonymity by performing a permutation on the noisy data it receives from the participants before handing it to the curator. Since the initial work from Bittau et al. [7], there have been many interesting works in this space [4, 11, 24]. Although these works significantly reduce the dependency on the number of clients (for example Balle et al.’s work has a dependency of $O(\sqrt{m})$ [4]), they still involve adding noise locally, which in addition to making the error dependent on the number of clients, disrupts the correlation between keys and values. It has been shown that the utility of this approach is strictly between that of the local and central models [4, 11]. Thus in our work, we take the approach of simulating the central trust model using MPC to avoid any dependence on the number of clients by adding randomness in a central manner.

Combining Cryptography and Differential Privacy. The area of research dedicated to the intersection of cryptography and differential privacy has been termed DP-Cryptography [50]. One direction of this research uses MPC when computing DP statistics to eliminate a trusted central data curator [19, 22, 40]. Dwork et al. [19] provided protocols to generate Gaussian and exponentially distributed noise in a distributed setting. The combination of secure computation and DP was further explored in PrivaDA [22], a generic design that allowed for noise generation in a fully distributed setting. We note that these constructions cannot immediately be applied to key-value data without incurring high computation costs as a result of secret sharing the keys. Pettai and Laud [42] implemented the DP sample-and-aggregate method in an MPC framework, showing that the combination can still provide reasonable performance. Later work used homomorphic encryption for the secure computation protocols [2, 36, 43, 45]. Goryczka et al. [29] implemented and compared early models that combined MPC and DP.

Another area of DP-Cryptography uses DP to improve the performance of cryptographic primitives. Allowing for differentially private leakage can improve the computation or communication costs. Chen et al. [10] employ this technique in the context of searchable symmetric encryption. Toledo et al. [47] relax the information-theoretic requirements of private information retrieval to provide more efficient constructions which satisfy differentially private leakage. Similarly, Shrinkwrap [5] uses DP query processing to improve the performance of evaluating SQL queries. Mazloom and Gordon [38] extend the concept of differentially private leakage to secure multi-party computations of histograms, PageRank, and matrix factorization. DP leakage has also been used in the context of anonymous communication to reduce communication costs [35, 48, 49]. Our work is unique in that we simultaneously take both approaches to combining DP and cryptography: we allow differentially private leakage to improve performance and use secure computation to eliminate trust assumptions.

11 CONCLUSION

Although local differentially private solutions are becoming increasingly popular, they are also well known to provide poor utility. Unfortunately, a common way to circumvent this issue is to let $\epsilon$ be unreasonably large, offering almost meaningless protection to users’ privacy. By minimizing the costs that organizations incur when choosing privacy-aware solutions, we can encourage the development and use of systems that respect users’ privacy.

In line with this ideal, we provide an improved solution for calculating the frequency and mean of key-value data in the local trust model. Our work presents a novel approach to MPC, which circumvents traditional efficiency issues. Simultaneously, our approach enables us to achieve pure differential privacy guarantees on leakage without removing any data. By leveraging cryptographic primitives, we maintain the correlation between keys and values, resulting in a high accuracy algorithm. We believe that offering high efficiency, accuracy, privacy, and conceptual simplicity incentivizes the adoption of our approach in practice.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of NSERC for grants RGPIN-05849, IRC-537591, the NSERC Postgraduate Scholarship-Doctoral program, and the Royal Bank of Canada for funding this research.
