Seung Geol Choi*, Dana Dachman-soled, Mukul Kulkarni, and Arkady Yerukhimovich

Differentially-Private Multi-Party Sketching for Large-Scale Statistics

Abstract: We consider a scenario where multiple organizations holding large amounts of sensitive data from their users wish to compute aggregate statistics on this data while protecting the privacy of individual users. To support large-scale analytics we investigate how this privacy can be provided for the case of sketching algorithms running in time sub-linear of the input size.

We begin with the well-known LogLog sketch for computing the number of unique elements in a data stream. We show that this algorithm already achieves differential privacy (even without adding any noise) when computed using a private hash function by a trusted curator. Next, we show how to eliminate this requirement of a private hash function by injecting a small amount of noise, allowing us to instantiate an efficient LogLog protocol for the multi-party setting. To demonstrate the practicality of this approach, we run extensive experimentation on multiple data sets, including the publicly available IP address data set from University of Michigan's scans of internet IPv4 space, to determine the trade-offs among efficiency, privacy and accuracy of our implementation for varying numbers of parties and input sizes.

Finally, we generalize our approach for the LogLog sketch and obtain a general framework for constructing multi-party differentially private protocols for several other sketching algorithms.

Keywords: differential privacy, sketching algorithms, secure computation

DOI 10.2478/popets-2020-0047
Received 2019-11-30; revised 2020-03-15; accepted 2020-03-16.

***Corresponding Author: Seung Geol Choi:** United States Naval Academy. Email: choi@usna.edu
Dana Dachman-soled: University of Maryland, Colleage Park. Email: danadach@ece.umd.edu
Mukul Kulkarni: University of Massachusetts Amherst. Email: mukul@cs.umass.edu
Arkady Yerukhimovich: George Washington University. Email: arkady@gwu.edu

1 Introduction

Collecting and performing analytics on large amounts of personal data has become widespread and is intrinsic to the functionality of a rapidly growing number of apps and services. While desirable from a functionality perspective, this now popular computing paradigm raises unprecedented security and privacy concerns, and there is a growing and urgent need for technology solutions that balance these interests.

In this paper, we aim at achieving a privacy-preserving mechanism that handles the following scenario:

> *Multiple organizations (e.g., hospitals, banks, government agencies, or nodes in a network) each hold sensitive data from their users, and they wish to compute aggregate statistics while protecting the privacy of individual users.*

While our approach is quite general, for a concrete application we consider the problem of Tor measurement; specifically, counting the number of unique users across the Tor network [19]. In this scenario, the parties are the Tor relays that are tasked with routing Internet traffic for as many as (approximately) 2 million daily users. Tor guard relays, in particular, serve as users' entry points into the Tor network, and thus learn the IP addresses of users that route their traffic through them. In a day, each such guard relay may observe thousands if not millions of connections, and, moreover, since users are encouraged to regularly form new circuits, the same users are likely to route traffic through multiple guard relays in a given day. Thus, while the relays have a large number of unique users across them, there is also a good amount of overlap among the IP address lists at each relay.

Computing the number of unique users across a set of guard relays can measure the popularity of Tor in a particular part of the world, or detect fluctuations in the number of users indicating that traffic is being blocked. However, the identities (i.e., IP addresses) of Tor users are highly sensitive—after all, protecting their privacy is the raison d'être of Tor. Moreover, since some relays may be controlled by an adversary, it is necessary to protect the privacy of the input IP addresses both during the computation and after the output is released. A similar application enables exit relays to privately estimate the number of Tor hidden services visited.

Our proposed protocol enables the relays to collaboratively approximate these counts while maintaining both the privacy of the relays' inputs and internal state as well as the privacy of any individual user, even given the output of the computation.

Insufficiency of MPC. To address this scenario, one can apply the notion of *secure multiparty computation* (MPC). MPC allows parties to perform a distributed computing task—i.e. compute some joint functionality—while revealing nothing but the final output of the computation. The organizations would like to execute MPC protocols to allow individual users to keep their personal data private, while still benefiting from analytics performed across a large number of users' data.

While MPC will indeed be one aspect of the solution proposed in this work, MPC in and of itself does not solve the problem of ensuring privacy of personal data in large-scale, distributed computing settings.

1. *Privacy issue.* First, MPC *does not* actually provide privacy guarantees for an individual user's personal data! We note that MPC provides only a "relative" security guarantee, ensuring that parties do not learn *more* from running the MPC protocol, than what they *could have* learned given only the output of the functionality. It may be the case, however, that the output of the functionality itself reveals private information about an individual user, rendering the security guarantee provided by the MPC meaningless. In particular, in the Tor example, revealing the exact count on a daily basis may reveal when a new (targeted) IP uses Tor.
2. *Efficiency issue.* Moreover, computing statistics by having all user data as input to an MPC protocol will not be feasible in terms of efficiency. While MPC protocols are becoming more and more efficient, they still lose at least an order of magnitude in efficiency, compared to performing the computation in the clear. For big data tasks, where even linear-time functions stretch the limit of feasible computation, this approach will not work.

Our approach. In this work, we will use a combined approach to solve the above problems.

To address (1), we require that the functionality computed by the MPC satisfy certain privacy requirements. Specifically, we require that the functionality itself is *differentially private* (DP) [24, 26], which (roughly) guarantees that the output of the functionality does not change much if a single individual changes the data she contributes. In particular, this implies that it is impossible to recover an individual's data, given only the output of the functionality. The differential privacy notion has become the de facto standard for guaranteeing an individual's privacy and has been broadly adopted within the security community.

There are several well-known mechanisms for releasing the *approximate* output of a functionality, while satisfying differential privacy guarantee [26, 51, 57]. All DP mechanisms inherently suffer from a trade-off between accuracy and privacy. In the most commonly used mechanisms [26, 57], this trade-off is *explicit* since noise is deliberately added to the output of a functionality before releasing it, so as to ensure privacy.

This now leads us to addressing (2): Since the functionality is required to be DP, the output will necessarily only be approximate. Therefore, we may now consider functionalities that are only approximate *even before the differentially private noise is added.* This relaxation allows us to achieve far better efficiency and thus obtain practical solutions *even when the approximate functionality is computed via an MPC protocol.* The approximate functionalities that we utilize are based on well-known *sketching algorithms* from the literature [5, 22, 34, 40, 59]. Such algorithms employ a special data structure, known as a "sketch," which is compact, structure-preserving and efficiently updatable in the *streaming* setting.

1.1 Our Contribution

In this paper, we propose a framework that combines MPC, Sketching and Differentially Privacy. We begin by focusing on a specific type of big data task and a corresponding well-known sketch for this task. The task we consider is computing approximate *disjoint count* in the single party setting, or approximate *set union cardinality* in the multiparty setting. Both of these tasks can be accomplished via a well-known sketch known as the "LogLog" sketch [22]. This statistic has attracted significant attention and been used in many applications, e.g., in Tor measurement [69], traffic monitoring in networks [50], in-network query aggregation in wireless sensor network [55], and file significance evaluation in P2P systems [58].

We then extend our results by presenting a general framework for multiparty, differentially-private computation of various popular sketches. Our work consists of three main contributions.

DP of the LogLog sketch. First, we show that the popular "LogLog" algorithm is *itself* DP in the single party setting. Given a stream of sensitive data items each obtained from an individual user, the LogLog algorithm allows the organization to maintain a small-size sketch based on the hash value of each item. At the end, the organization outputs the approximate count of distinct elements from the sketch.

We show that the approximate count as computed by the LogLog sketch, *even without adding noise* is guaranteed to be DP, when the organization choose a hash function privately at random.

Extending to the multiparty setting. We achieve DP disjoint counting in the multiparty setting in the semi-honest model assuming an honest majority.

The LogLog sketch has the important property that it can be *merged*, which implies that multiple organizations can locally compute a sketch and these can be merged to obtain the (approximate) set union cardinality across all parties. However, the problem is that this merging can only be performed when *the same hash is used* by all parties; as described above, differential privacy of a local sketch from a single party holds only when the hash function is privately chosen at random. Therefore, when multiple local sketches are merged using the same public hash, the merged sketch does not guarantee differential privacy anymore. Indeed, it is easy to see that given the merged sketch, or the resulting count, an adversary can check (with noticeable probability) whether a particular IP is included in the sketch by observing whether adding it causes the count to change.

We overcome this challenge by securely adding a small amount of noise according to the Laplace mechanism. Since sampling noise inside an MPC protocol can be prohibitively expensive (requiring high precision computation of continuous, real-valued functions corresponding to the noise distribution), we introduce a novel way to sample the noise in a distributed fashion. We implement our MPC protocol that securely merges the sketches and securely adds a noise; we provide a concrete analysis of the resulting privacy and accuracy parameters.

A general framework. Finally, we generalize our protocol above for the LogLog sketch to obtain a general framework for efficiently combining a sketching algorithm that is *mergeable*, a DP mechanism and an MPC protocol. This framework can be applied to the LogLog sketch and other sketches such as AMS sketch [5] and the Johnson-Lindenstrauss Transform [42, 46].

1.2 Organization

We first give the background and related work in Section 2. Definitions and preliminaries are found in Section 3. In Section 4, we show that the LogLog algorithm itself is differentially private in the single party setting when the hash function is kept private. The protocol for the case of the public hash function is described in Section 5. In Section 6, we extend the single-party protocol to the multi-party setting and report the experiment results. Finally, we describe the general framework for distributed private sketching in Section 7 and conclude in Section 8.

2 Background and Related Work

2.1 DP and MPC

Differential privacy (DP) [24, 26] protects privacy of an individual by limiting the possible inferences that an adversary could draw about the individual from the output of the computation. That is, what can be learned about an individual from a differentially private computation is essentially limited to what could be learned about him from everyone else's data *without his own data being included in the computation.* Since its inception, a large body of research has been devoted to the design of differentially private algorithms (see [27] and references therein). Most works mainly considered the standard setting with a trusted curator who has access to all users' data and aims to respond in a differentially private manner.

Another active line of work has considered *local differential privacy* [30, 47] where parties locally add noise to their data to guarantee privacy before submitting it to an untrusted curator. Our model differs from this in that we use secure multi-party computation to, instead, have parties jointly simulate a trusted curator resulting in less noise being added.

Secure multi-party computation (MPC) [37, 70] is concerned with privacy of the users' input from a different perspective. Secure MPC protocols allow a set of parties $P_1, \dots P_n$ to compute some function of their inputs in a distributed fashion, while revealing nothing to a coalition of corrupted parties about any honest party's input beyond what is implied by the output. Recently, a large amount of active research has been conducted toward achieving efficient protocols for secure MPC (see [31] and the references therein).

DP in the multiparty setting. Note that secure MPC protocols do not protect against the leakage stemming from the actual output of the function. Therefore, it is desirable that the function should ensure that its output guarantees privacy of the private inputs even if secure MPC protocols are to be used.

Toward this goal, Dwork et al. [25] presented a multiparty protocol for sampling from a Laplace distribution. However, although their sampling protocol is secure in the malicious model, security of the entire protocol still requires the parties to choose their actual data inputs honestly. Rather than following their non-standard model, we adopt the standard semi-honest model for both noise sampling and the rest of the protocol. We show how semi-honest sampling of Laplace noise can be done at a fraction of the cost of their maliciously secure protocol. (They need a secure computation of $\Omega(n \log n)$ gates over $O(\log n)$ rounds to generate n random coins, while we just use non-interactive addition of locally generated noise.)

The formal security definition for differentially private multi-party protocols was first given by Beimel et al. [11] in the information theoretic setting. Their definition has been extended to the computational setting in [53, 63].

An alternative line of work [32, 39] instead studied secure approximation. However, their notion of security is fundamentally different from the DP security we aim for. In particular, they aim for a more semantic-security style flavor requiring that the approximate output should reveal no more than what the exact output reveals about the parties' private input.

2.2 Secure Statistics

Cardinality estimation. Although the cardinality of a multi-set (i.e., the number of unique elements) can be easily computed using space linear in said cardinality, the linear space complexity is often burdensome for many applications dealing with a very large amount of data. Therefore, approximation algorithms have been developed that need only sub-linear space, maintaining a small sketch—a data structure that allows aggregation of items—of size typically less than thousands of bytes. To name a few, these sketch-based algorithms include probablistic counting [35], MinCount [8], LogLog [22] and its variants HyperLogLog [34], and HyperLogLog++ [40].

The first work on privacy-preserving cardinality estimators is of Tschorsch and Scheuermann [67]; they proposed a cardinality estimation mechanism in a distributed setting. The mechanism adds noise to sketches and then merges noisy sketches in public. This approach incurs a larger amount of error compared to our approach, and they used their own privacy metric instead of the standard DP framework. Protocols based on MPC and Bloom filters have also been suggested [7, 28], but they did not consider differential privacy of individual users.

Dong and Loukides [20] gave secure two party protocol (which can be extended for multiple parties) for approximate set union and set intersection counting based on probabilistic counting sketches [35]. Their protocol for set union approximate counting outputs individual shares of the final output to each party, and runs in time $\log(N)$, where N is size of largest input set. In comparison, the runtime of our protocol only depends on the desired level of accuracy. Additionally, we also provide differential privacy of individual inputs.

Alaggan et al. [4] used Bloom-filter based techniques to achieve disjoint count estimates (and other statistics) over Wi-Fi data. They considered the centralized curator model only (i.e., a single-party setting) while we consider not only the single-party but also the multi-party setting. However, their security guarantees are stronger; they achieve pan-privacy against a malicious adversary. Due to the stronger security guarantees, their construction has significantly larger error than in our construction, i.e., 10% vs 2% for $\epsilon = 1$ (See Figure 4 in [4] and Figure 4 in this paper).

Recently, Desfontaines et al. [18] considered privacy of cardinality estimators from a different angle. In particular, they assumed an insider risk scenario where the adversary has access to the actual sketch (rather than the output statistic) and showed that no noiseless sketch can be differentially private. They also suggested hiding the hash key as a way of protecting such an attack; in this work, we affirm their suggestion by providing a rigorous proof (see Section 4).

Differentially private aggregation of statistics. A number of works have also considered aggregation of other statistics.

Erlingsson et al. [30] proposed Randomized Aggregatable Privacy-Preserving Ordinal Response (RAPPOR) for aggregating statistics (such as categories, frequencies, and histograms) generated via crowdsourcing. The system collects randomized responses via input perturbation, aiming to guarantee local differential privacy for individual reports. However, due to the larger noise necessary for local differential-privacy, it requires at least millions of users in order to obtain reasonably accurate approximate answers.

Other examples of private aggregation include [29, 33, 44, 52]. These works aggregate statistics computed by linear sketches with aggregation done using secret sharing [29, 44], accumulators based on public-key cryptography [33], and homomorphic encryption [52]. All these works are limited to computing linear sketches only and thus cannot support computations such as disjoint count.

Finally, the work of Wails et al. [69] also considers aggregate statistics for Tor measurement using MPC. However, their work focuses on optimizing MPC for a large number of parties and does not consider differential privacy.

Other related work. Recently, Bater et al. [9] considered performing SQL queries in a differentially private manner. Perhaps the most relevant aspect of their work to our work, is computing "Aspirin count" (counting distinct items matching some criteria from multiple tables). They report computing 2 join (similar to 2 party MPC) for data size at least $500K$ for each party.

Additionally, recent work, e.g. [49] has pointed out that differential privacy guarantees may be insufficient when data in the database is correlated as in the case when the same or closely related users may appear in multiple organizations' data sets. We note that for the case of disjoint counting our protocol provides security for correlated data since all instances of the same user are combined into a single record in the sketch, while related but unequal records are made independent by hashing. Understanding the impact of correlated inputs for other sketches is an interesting open problem.

2.3 Background on Sketching

Our framework requires sketches that are mergeable. Given such a sketch, our approach allows releasing statistics calculated from the merged sketch. The amount of noise to be added in the released statistics depends on the sensitivity of the statistic itself as well as the accuracy of the sketch. Note that our approach does not allow releasing the entire sketch itself.

A large body of work within the sketching literature is dedicated to *linear sketches*. These are sketches that are computed by applying a linear function (represented by a matrix $\mathbf{A}$) to the data X, i.e. a sketch S is computed as $S = \mathbf{A}X$. While non-linear sketches can still be mergeable—the LogLog sketch from above is *non-linear and mergeable*—linear sketches are inherently mergeable: If the merged data of N parties, each holding data X_i, is computed as $X = X_1 + \cdots + X_N$, then $\mathbf{A}(X) = \mathbf{A}(X_1 + \cdots + X_N) = \mathbf{A}X_1 + \cdots + \mathbf{A}X_N$. We describe several linear sketches from the literature and the types of low-sensitivity statistics that can be computed from these sketches. All of these sketches fall under our framework. We then provide some examples of sketches that do not fall under our framework.

Count-Sketch and Count-Min Sketch. Count-Sketch [15] and Count-Min Sketch [16] are sketches that are used to approximate the frequencies of elements in a data stream. Assume X is the exact frequency vector of dimension n. These sketching algorithms proceed by storing multiple independent copies of $\mathbf{A}X$, where $\mathbf{A}$ is an $m \times n$ dimension matrix (with $m \ll n$) drawn from a certain distribution. The stored sketches can be used to approximate statistics such as median, quantiles, or histograms.

AMS sketch, JL-transform and L_p-norm. By sampling $\mathbf{A}$ from a more complex distribution than above, it is possible to achieve the property that $||\mathbf{A}X||_p \approx ||X||_p$, where $|| \cdot ||_p$ denotes the L_p-norm of a vector. The AMS sketch [5] and the Johnson-Lindenstrauss transform [42, 46] provide distributions for $\mathbf{A}$ such that $||\mathbf{A}X||_2 \approx ||X||_2$. This was subsequently generalized to arbitrary L_p-norms [41, 43]. Note that the sensitivity of the L_p norm can be bounded, when the magnitude of each element of X is bounded.

L_p sampling. L_p sampling is the task of sampling from the distribution that places weight $X_i^p/||X||_p^p$ on the i-th element of the n-dimensional vector $X = X_1, \ldots, X_n$. A sequence of works showed that it is possible to compute a linear sketch of X that allows one to perform approximate L_p sampling [17]. Such sketches can be used to perform moment estimation [45, 54] and entropy estimation [13]. L_0 sampling is used for graph sketching, as discussed below.

Graph sketches. An L_0 sampling sketch for each node of the graph allows the sampling of a neighbor of each node uniformly at random. The above forms the basis of several graph sketches, including a sketch for minimum spanning tree (MST) cost. Since the L_0 sampling sketches are linear and can be merged, the resulting sketch is mergeable. Furthermore, approximate MST cost can be computed on top of the sketch [2]. We may consider the sensitivity of the MST cost when perturbing a single edge weight of the graph, but keeping the topology of the graph fixed. This is similar to the privacy model considered by Sealfon [61].

Sketches that do not fall under our framework. There are several such examples, including certain sketches for graph *sparsification*, where the goal is to construct a sparse graph that is similar in some way to the original graph. The sparsifiers of [10, 12, 64, 65] are not linear and do not appear to be mergeable. Importantly, we note that even if sketches are mergeable, if they compute statistics that have high sensitivity (e.g. connectivity of a graph), they would not be good candidates for our framework, since the amount of noise added would be prohibitively large.

3 Definitions and Preliminaries

Differential privacy in the single party setting. We first define differential privacy in the setting of a single trusted curator. Following the work of Dwork and Roth [27], we think of databases as being collections of records from a universe $\mathcal{X}$, and view databases using their histograms $X \in \mathbb{N}^{|\mathcal{X}|}$, where $\mathbb{N}$ is the set of all non-negative integers. In this definitional framework, each entry x_i represents *the number of elements in the database X of type $i \in \mathcal{X}$*. The ℓ_1 norm of database X is defined as $\|X\|_1 := \sum_{i=1}^{|\mathcal{X}|} |x_i|$, which is a measure of the size of the database. The ℓ_1 distance between two databases X and X' is $\|X - X'\|_1$, which is a measure of how many records differ between X and X'.

Definition 3.1. *A randomized algorithm M with domain $\mathbb{N}^{|\mathcal{X}|}$ is (ϵ, δ)-differentially private if for all $S \subseteq Range(M)$ and for all $X, X' \in \mathbb{N}^{|\mathcal{X}|}$ such that $\|X - X'\|_1 = 1$:*

$$\Pr[M(X) \in S] \le e^{\epsilon} \Pr[M(X') \in S] + \delta.$$

Below, we give more formal treatment following the work of Dwork and Roth [27].

Definition 3.2. *The global sensitivity of a function $f : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$ is:*

$$\Delta f = \max_{X,Y \in \mathbb{N}^{|\mathcal{X}|}, \|X-Y\|_1 = 1} \|f(X) - f(Y)\|_1$$

Definition 3.3. *The Laplace Distribution (centered at 0) with scale b is the distribution with probability density function:*

$$Lap(x|b) = \frac{1}{2b} e^{-x/b}.$$

We will write $Lap(b)$ to denote the Laplace distribution with scale b.

Given any function $f : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$, the Laplace mechanism that adds noise drawn from Laplace distribution; that is, given an input database X, the mechanism outputs

$$f(X) + (Y_1, \dots, Y_k),$$

where Y_i are i.i.d. random variables drawn from $Lap(\Delta f / \epsilon)$. It is known that the Laplace mechanism achieves $(\epsilon, 0)$-differential privacy [27, Theorem 3.6].

3.1 Distributed Differential Privacy

Our presentation here follows the similar definitions given in prior work [11, 63]. We assume that readers are familiar with security notions of standard cryptographic primitives [48] and formal definitions of a protocol securely realizing an ideal functionality (cf. [31]).

Notations and semi-honest adversary. Let λ denote the security parameter. A function g is said to be *negligible* if for every positive integer c, there is an integer n_c such that for all $n \ge n_c$ we have $g(n) \le 1/n^c$. Throughout the paper, we will usually use $\mathsf{negl}(\cdot)$ to denote a negligible function. We assume that the adversary is semi-honest.

C-neighboring inputs. Let $\mathcal{P} = \{P_1, \dots, P_N\}$ be the set of computing parties, with party P_i holding inputs X_i. Viewing X_i as a histogram (i.e., $X_i \in \mathbb{N}^{|\mathcal{X}|}$) as in the single-party definition above, we say that two input sets $(X_1, \dots, X_N)$ and $(X'_1, \dots, X'_N)$ are *neighboring* if there is a single index i such that $\|X_i - X'_i\|_1 = 1$ and for all $j \ne i$, $\||X_j - X'_j\|_1 = 0$. For a coalition of parties $C \subseteq \mathcal{P}$, we say that two input sets $(X_1, \dots, X_N)$ and $(X'_1, \dots, X'_N)$ are *C-neighboring* if the input sets are neighboring and for the index i at which they differ $P_i \notin C$.

Distributed differential privacy. We can now define computational distributed differential privacy against a coalition C. Roughly, this definition says that if a party outside C changes their input by a single value, the view of the coalition will not change too much.

More formally, for an N-party protocol Π and an input $(X_1, \dots, X_N)$, we let $\Pi(X_1, \dots, X_N)$ denote the execution of Π on this input. For a coalition C, we define the view of C in protocol Π, denoted $\Pi(X_1, \dots, X_N)|_C$, as the inputs x_i for $P_i \in C$, the random tapes of all parties in C, and all messages received by parties in C.

Definition 3.4 (cf. Def 2.1 in [63]). *Let $\epsilon > 0$ and $0 \leq \delta < 1$. A (randomized) protocol Π preserves* computational distributed (ϵ, δ)-Differential Privacy *against a coalition C, if for any polynomial-time adversary $\mathcal{A}$, for all C-neighboring inputs $(X_1, \ldots, X_N)$ and $(X'_1, \ldots, X'_N)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that,*

$$\Pr[\mathcal{A}(\Pi(X_1, \ldots, X_N)|_C, 1^\lambda) = 1] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(\Pi(X'_1, \ldots, X'_N)|_C, 1^\lambda) = 1] + \delta + \mathsf{negl}(\lambda)$$

One natural way to achieve differential distributed privacy is executing secure MPC protocol for a differential private functionality, as shown below:

Theorem 3.5 (Lemma 4.3 [63]). *Let f be (ϵ, δ)-differentially private, and let Π be protocol securely realizing f against a coalition C. Then, Π preserves computational distributed (ϵ, δ)-differential privacy against a coalition C.*

4 A Single-party Protocol

In this section, we first show that assuming *the hash key is kept secret from the adversary*, the LogLog algorithm (and its variants) actually achieves differential privacy without adding any noise.

Recently, a seemingly contradictory result has been published. In particular, Desfontaines et al. [18] showed that the LogLog sketch does not protect privacy from the inside attacker who has access to the sketch. Their result is not in conflict with our result in this section, since they assumed that the inside attacker also knows the hash key whereas our result assumes the private hash function. Indeed, Desfontaines et al. [18] suggested to hide the hash key as a way of protecting their attack; we affirm their suggestion by providing a rigorous proof. We prove differential privacy of the algorithm in the random oracle model.

We note that prior work [14, 62, 68] proved a similar result for the JL transform showing that some forms of the JL transform are differentially private when the JLT matrix is kept private.

```
Accuracy parameter: $K$.
Input: A sequence $D$ of elements, i.e., $D = (x_1, x_2, \ldots)$.
Output: An approximate count of distinct elements in $D$.

1. Initialize counters $c_1, \ldots, c_K$ each with 0.
2. Choose a random hash function $h$. Let $h$ output
   $\log K + m$ bits.
3. For each $x_i$ in $D$:
   (a) Compute $(j, y) := h(x_i)$, where $j$ is the first
       $\log K$ bits of $h(x_i)$ and $y$ is the rest. Abusing
       the notation, we will treat the $k$-bit string $j$ as
       a number $0 \leq j < K$.
   (b) Update $c_j := \max(c_j, \rho(y))$.
4. Compute the sum of the counter values, i.e., $u :=
   \sum_{j=1}^{K} c_j$
5. Output $\alpha \cdot K \cdot 2^{u/K}$.
   Here, $\alpha$ is a constant depending on $K$ only.[a]
```

[a] In particular, it holds $\alpha = \left(\Gamma(-1/K) \cdot \frac{1-2^{1/K}}{\log 2}\right)^{-K}$ with $\Gamma(s) = \frac{1}{s}\int_0^\infty e^{-t}t^s dt$.

Fig. 1. The LogLog algorithm.

4.1 DP Without Noise

Here, we describe the LogLog algorithm [22], which is the simplest and thereby best to explain how differential privacy is achieved without noise. Our analysis is easily carried over to other variants.

LogLog algorithm. Before describing the actual algorithm, we introduce a useful notation. For a binary string s, we define $\rho(s) := 1 + z(s)$ where $z(s)$ is the number of consecutive 0s in s counting from the left most position. For example, it holds $\rho(000011) = 5$, $\rho(010100) = 2$, $\rho(110000) = 1$, and $\rho(000000) = 7$. The LogLog algorithm is described in Figure 1.

Accuracy. The algorithm has the following accuracy.

Theorem 4.1 ([22]). *Consider the LogLog algorithm applied to D with n distinct elements, and let $\tilde{n}$ be the output of the algorithm. With $\sigma = 1.30/\sqrt{K}$, we have*

$$\Pr\left[\left|\frac{\tilde{n} - n}{n}\right| \geq 2\sigma\right] \leq 0.05.$$

and

$$\Pr\left[\left|\frac{\tilde{n} - n}{n}\right| \geq 3\sigma\right] \leq 0.01.$$

Differential privacy. Interestingly, this algorithm achieves differential privacy without adding noise when D is large.

Theorem 4.2. *Suppose D contains n distinct elements with $n \geq 8K\lambda \cdot \max(\frac{1}{\epsilon}, 1)$, where λ is the security* *parameter. Then, the LogLog algorithm applied to D achieves $(\epsilon, \mathsf{negl}(\lambda))$-differential privacy in the random oracle model.*

The proof is given in Appendix A. Here, we give intuition for why the theorem holds. In particular, we consider the following simple experiment $C_m(n)$. In relation to the LogLog algorithm, the output $C_m(n)$ corresponds to the counter value for a single bucket when n distinct items are considered in that bucket, assuming the random oracle model. Note differential privacy for this single-bucket scenario can be extended to the multi-bucket scenario as considered in the above theorem, since in the LogLog algorithm, difference of one item in neighboring databases will affect only a single bucket.

Experiment $C_m(n)$:
1. Choose $x_1, \ldots, x_n$ independently uniformly at random from $\{0,1\}^m$.
2. Output $\max\{\rho(x_1), \ldots, \rho(x_n)\}$.

First, for any n, any $m = \omega(\log n)$, and any $s \le m$, we have

$$\Pr[C_m(n) < s] = (1 - 2^{-(s-1)})^n.$$

To see why, given a binary string s, let $\mathsf{left}_k(s)$ be the leftmost k bits of s. For example, we have $\mathsf{left}_3(000101) = 000$. Now, for each i, with independent probability, we have that

$$\begin{aligned}\Pr[\rho(X_i) < s] &= 1 - \Pr[\rho(X_i) \ge s] \\ &= 1 - \Pr[\mathsf{left}_{s-1}(X_i) = 0^{s-1}] = 1 - 2^{-(s-1)}.\end{aligned}$$

Thus,

$$\begin{aligned}&\Pr[\max\{\rho(X_1), \ldots, \rho(X_n)\} < s] \\ &= \Pr\left[\bigwedge_{i=1}^{n} \rho(X_i) < s\right] = (1 - 2^{-(s-1)})^n.\end{aligned}$$

This implies that

$$\begin{aligned}\Pr[C_m(n) = s] &= \Pr[C_m(n) < s+1] - \Pr[C_m(n) < s] \\ &= \left(1 - \frac{1}{2^s}\right)^n - \left(1 - \frac{1}{2^{s-1}}\right)^n.\end{aligned}$$

The following two Lemmas show that the distributions $C_m(n)$ and $C_m(n+1)$ are close, which implies differential privacy in the single-bucket scenario.

Lemma 4.3. *For any n, any $m = \omega(\log n)$, and any $s \le m$, it holds*

$$\Pr[C_m(n+1) = s] \le (1 + 1/n) \cdot \Pr[C_m(n) = s].$$

Proof. Let $a = 1 - 2^{-s}$, $b = 1 - 2^{-s+1}$. Note $b < a < 1$.

$$\begin{aligned}(1 + \frac{1}{n}) \Pr[C_m(n) = s] &= (1 + \frac{1}{n})(a^n - b^n) \\ &= (a-b)(1 + \frac{1}{n}) \cdot \sum_{i=0}^{n-1} a^i \cdot b^{n-1-i} \\ &> (a-b)(\sum_{i=0}^{n-1} a^i \cdot b^{n-1-i} + \frac{1}{n}\sum_{i=0}^{n-1} b^{n-1}) \\ &> (a-b)(\sum_{i=0}^{n} a^i \cdot b^{n-i}) = \Pr[C(n+1, m) = s].\end{aligned}$$

□

Lemma 4.4. *For any n, any $m = \omega(\log n)$, and any $2 \le s \le m$, it holds*

$$\Pr[C_m(n) = s] \le (1 + 2^{-(s-2)}) \cdot \Pr[C_m(n+1) = s].$$

The proof is similar and it's given in Appendix A.

5 Achieving DP with Public Hash

Possible attacks and adding noise. When the hash key is revealed to the public, the LogLog algorithm described in Figure 1 fails to achieve differential privacy. Recall our analysis above started with choosing $x_1, \ldots, x_n$ uniformly at random where x_i corresponds to a hash output of an input. However, given that the hash key is revealed, x_i is not randomly distributed anymore; it's just some determined value. So, the entire analysis breaks down.

In fact, the adversary can distinguish two neighboring databases D and D' by hashing all the items in D and D' on its own and matching the result against the output of the LogLog algorithm. This attack works, since it's highly unlikely for the outputs from D and D' to be exactly the same. In order to guarantee differential privacy in this situation, we have to rely on added noise.

We note that more sophisticated attack was shown by Desfontaines et al. in [18] in the insider attack threat model, i.e., when the attacker has the sketch (i.e., the counter value for each bucket) in addition to hash function and the algorithm output. Interestingly, their [18] attack applies only to noiseless cardinality estimators as well.

Sensitivity of the LogLog algorithm and the Laplace mechanism. It is well known that a mechanism can be augmented to achieve differential privacy

```
Accuracy parameter: $K$.
Parameter for differential privacy : $\epsilon$
Parameter of the maximum possible number of items:
    $N_{max}$, i.e., $N_{max} \geq |D|$.
Public hash: $h : \{0,1\}^* \to \{0,1\}^{\log K} \times \{0,1\}^m$, where
    $m = 3 + \log \frac{N_{max}}{K}$.
Input: A sequence $D$ of elements, i.e., $D = (x_1, x_2, \dots)$.
Output: An approximate count of distinct elements in $D$.

1. Initialize counters $c_1, \dots, c_K$ each with 0.
2. For each $x_i$ in $D$:
    (a) Compute $(j, y) := h(x_i)$, where $j$ is $\log K$ bits
        long and $y$ is $m$ bits long. Abusing the notation,
        we will treat the $k$-bit string $j$ as a number $0 \leq$
        $j < K$.
    (b) Update $c_j := \max(c_j, \rho(y))$.
3. Compute the sum of the counter values, i.e., $u :=$
   $\sum_{j=1}^{K} c_j$
4. Choose a random number $r$ drawn from $Lap((1 +$
   $m)/\epsilon)$.
5. Output $\alpha \cdot K \cdot 2^{(u+r)/K}$.
```

Fig. 2. Differentially private LogLog with public hash.

by adding a reasonable amount of noise, *if the mechanism has low sensitivity*, i.e., when small change in the input leads to small change in the output.

To evaluate the sensitivity of the LogLog algorithm described in Figure 1, consider any two neighboring inputs D and D' and let u and u' be the sum of counters that the algorithm, on input D and D' respectively, computes at step 4. As before, we consider the differential privacy for this sum.

Note that only one item is different in D and D'. Each item will either positively contribute to the final sum by its ρ value (if the value is the maximum in the bucket) or be ignored. Let ρ_{max} denote the maximum possible ρ value. Then, it holds that $u = u_{same} + \alpha$ and $u' = u_{same} + \beta$ where $\alpha, \beta \in [0, \rho_{max}]$, and u_{same} is the sum of counters for the items that belong to both D and D'. This implies that we have $|u - u'| = |\alpha - \beta| \leq \rho_{max}$. Therefore, the global sensitivity of the algorithm is $\Delta f = \rho_{max}$ and we can achieve $(\epsilon, 0)$-differential privacy by adding noise drawn from $Lap(\rho_{max}/\epsilon)$ (see in Figure 2).

Engineering the parameter for better accuracy. Since the amount of noise is ρ_{max}, we should set the length of the hash digests as short as possible to reduce the noise and optimize accuracy of the overall mechanism. That is, letting $h : \{0,1\}^* \to \{0,1\}^{\log K + m}$, we would like to determine the minimum value for m that still guarantees the accuracy of the cardinality estimation algorithm. Note $\Delta f = \rho_{max} = m + 1$. The original LogLog algorithm [22] suggests $m = 3 + \log \frac{N_{max}}{K}$, where N_{max} is the maximum possible number of items.

Since the Laplace mechanism achieves $(\epsilon, 0)$-differential privacy [27, Theorem 3.6], we achieve the following:

Theorem 5.1. *The mechanism described in Figure 2 is* $(\epsilon, 0)$*-differentially private.*

Accuracy of the mechanism. Note that the modified algorithm has an additional $2^{r/K}$ multiplicative factor in the output, where $r \sim Lap((m + 1)/\epsilon)$. To assess accuracy concretely, set for example $N_{max} = 2^{40}$ and $K = 2^{12}$, and aim for $(1, 0)$-differential privacy. In this setting, we have $m = 31$ and $r \sim Lap(32)$.

Recall if $Y \sim Lap(b)$, it holds $\Pr[|Y| \geq tb] \leq e^{-t}$. Therefore, with probability $1 - e^{-3} \approx 95\%$, the value $|r|$ is at most $3 \cdot 32 = 96$. This implies that most of the time, the multiplicative factor $2^{r/K} = 2^{96/4096}$ is at most 1.016, i.e., incurring about 1.6% accuracy degradation. One can improve accuracy by taking $K = 2^{13}$ at the expense of the run time to be doubled. Then, the accuracy degradation becomes 0.8%.

6 Multi-party Protocol

Now that we can achieve differentially private cardinality estimation in the single party setting, in this section, we explore how to extend this to the multi-party setting. Let N be the number of parties. We assume the semi-honest model and honest majority.

To achieve a differentially private multi-party protocol, we first need to answer the following question:

> *How do we correctly merge the distinct counts produced by the MPC parties? Simple addition would not be the total* distinct *count.*

Sketching and merging paradigm. Fortunately, the LogLog algorithm is already a sketch-based algorithm that allows merging of small sketches efficiently. That is, each participant will maintain the counters $(c_1, \dots, c_K)$ as a sketch. Then, we can merge the sketches as follows:

```
Merge:
1. Denote party $P_i$'s input sketch by $(c_1^i, \dots, c_K^i)$.
2. Compute $c_1 = \max\{c_1^i : i \in [N]\}, c_2 = \max\{c_2^i : i \in$
   $[N]\}, \dots, c_K = \max\{c_K^i : i \in [N]\}$.
3. Output $u = \sum_{j=1}^{K} c_j$.
```

```
Public parameters: $K, \epsilon, N_{max}, \alpha$ (refer to Figure 2).
Public hash: $h : \{0,1\}^* \to \{0,1\}^{\log K} \times \{0,1\}^m$, where
    $m = 3 + \log \frac{N_{max}}{K}$.
Participants: There are $N$ parties $P_1, \ldots, P_N$.
Private input of $P_i$: A sequence $D_i$ of elements, i.e., $D_i =$
    $(x_1^i, x_2^i, \ldots)$.
Output: An approximate count of distinct elements in $D =$
    $\bigcup_i D_i$.
1. Each $P_i$ locally computes the following:
    (a) Initialize counters $c_1^i, \ldots, c_K^i$ each with 0.
    (b) For each item $x$ in $D_i$,
            Compute $(j, y) := h(x)$ and update $c_j^i :=$
            $\max(c_j^i, \rho(y))$.
    (c) Sample $a_i, b_i \sim \Gamma(2/N, 1)$ and set $e_i = \frac{m+1}{\epsilon} \cdot$
        $(a_i - b_i)$.
2. Parties execute a secure MPC protocol to compute
   the following:
    – Input: $\{(c_1^i, \ldots, c_K^i), e_i\}_{i=1}^N$
    – Output $u := \sum_{j=1}^K (\max\{c_j^i : i \in [N]\}) +$
      $\sum_{i=1}^N e_i$
3. Each $P_i$ locally outputs the following:
        Given $u$ from MPC, output $\alpha \cdot K \cdot 2^{u/K}$.
```

Fig. 3. Differentially private multi-party LogLog algorithm.

Merge can be easily implemented as a circuit using comparison and addition gates and therefore using MPC one can securely compute u, the final sum.

Protocol using private hash. Recall that a LogLog sketch produced using a private hash function (e.g., run by a trusted curator) in Figure 1 is $(\epsilon, \mathsf{negl}(\lambda))$-differentially private. To achieve a differentially private multi-party protocol, we could have each party run an oblivious pseudorandom function (OPRF) [36] on each item to evaluate the private hash value and then construct the local sketch. Then, execute an MPC protocol computing Merge on the local sketches. However, this requires huge communication costs that is at least linear in the input size for the OPRF evaluations. This approach doesn't provide an efficient solution.

Protocol using public hash and generating private noise. We choose to employ the $(\epsilon, 0)$-DP LogLog algorithm with public hash described in Figure 2. An MPC protocol can be used to compute Merge on the local sketches. Therefore, we only need to generate the noise securely and add to the output of the Merge circuit.

Recall that we assume honest majority in the semi-honest model. The problem is that simply adding random variables from the Laplace distribution doesn't follow the Laplace distribution that we want. Still, we can sample a Laplace random variable in a distributed manner.

Fact 6.1. *$\forall k \in \mathbb{R}$, if $X \sim Lap(1)$, then $kX \sim Lap(k)$.*

Therefore, we only need to sample from $Lap(1)$ in order to sample $Lap(m+1)$.

Fact 6.2. *If $X, Y \sim \mathsf{Exponential}(\lambda)$ (i.e., exponential distribution with pdf $f(x) = \lambda e^{-\lambda x}$), then $X - Y \sim Lap(1/\lambda)$.*

Therefore, we only need to sample from $\mathsf{Exponential}(1)$ twice. Note that Dwork et al. [25] showed how to sample from an exponential distribution in the malicious model. Since we are in the semi-honest model, we can achieve our goal much more efficiently, as described below.

Fact 6.3. *The distribution $\mathsf{Exponential}(1)$ is equivalent to $\Gamma(1,1)$, where $\Gamma(\alpha, \beta)$ is the gamma distribution with pdf $f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$.*

Therefore, we reduce our task to sampling from $\Gamma(1,1)$ in the multi-party setting, which can be done by using the following fact:

Fact 6.4. *If $X_i \sim \Gamma(\alpha_i, \beta)$, then $\sum_i X_i \sim \Gamma(\sum_i \alpha_i, \beta)$.*

Based on the above facts, we generate the Laplace noise as follows:

Noise: Distributed sampling from $Lap(\ell)$.

1. Each party P_i prepares its input e_i as follows:
   – Choose a_i and b_i from *gamma distribution* $\Gamma(2/N, 1)$.
   – Set $e_i = \ell(a_i - b_i)$.
2. Execute an MPC protocol that computes $\sum_i e_i$.

Note that for any set $\mathcal{S}$ of size $N/2$, we have $\sum_{i \in \mathcal{S}} a_i \sim \mathsf{Exponential}(1)$ and $\sum_{i \in \mathcal{S}} b_i \sim \mathsf{Exponential}(1)$, thereby $\sum_{i \in \mathcal{S}} e_i \sim Lap(1)$. This suffices to show that the added noise achieves the differential privacy of the overall mechanism described in Figure 3.

Scalability of our protocol. Assuming that we are using an arithmetic circuit, the circuit used in the MPC protocol shown in Figure 3 has the following features:

– The size of input: Each P_i has $K+1$ numbers. Therefore, the total input size is $(K+1) \cdot N$.
– The number of operations: We need $N + K + 1$ additions and K operations of max over N numbers.

Based on the above, we conclude that the size of circuit is $O(N \cdot K)$ Therefore, the MPC computation will be *independent* of the actual input size $|D|$. This means that our protocol would *scale very well*; only the amount of local computation will increase as the size of the input increases.

Security of the protocol. It is easy to see the protocol is differentially private. The proof of the following Theorem is found in the Appendix B.

Theorem 6.5. *The protocol described in Figure 3 preserves $(\epsilon, 0)$ computational differential privacy against a semi-honest adversary that can corrupt at most t parties where $t < N/2$.*

6.1 Implementation

To demonstrate the efficiency of our protocol, we implemented the protocol described in Figure 3, using Python 3.7.4 and open-source secure multi-party computation framework MPyC [60]. In this section, we overview our implementation and in Section 6.2, we provide a summary of the performance measurements.

Choosing a hash function. We stress that *distributional differential privacy of our mechanism does not depend on the security of the hash function.* We achieve differential privacy based only on *security of the MPC protocol and the Laplace mechanism for the ideal functionality.* In particular, differential privacy holds even when the hash functions are public with seeds revealed to everyone. It is the accuracy guarantee, however, that requires a good hash function; our accuracy analysis assumes that each input is mapped to a random bit string.

A natural candidate would be a cryptographic hash function such as SHA-2. However, we choose AES to improve efficiency of computing the local sketch. AES is a pseudorandom function that maps an item into a pseudorandom string given a random key, which is just what we want. Of course, AES is not a hash function and its domain has a fixed length; if we need to have the hash function take arbitrarily long strings as input, AES would not work. Fortunately, our experiments deal with only small-length data such as IP addresses, so this is not a problem. The practice of using AES as a hash function has become increasingly popular, e.g., in the area of secure computation [38].

We have not performed further optimizations such as parallelizing the sketch computation.

Circuit optimizations. We use a 128-bit prime field for secret sharing the intermediate values (sketches) while implementing the MPC protocol given in step 2 of Figure 3. A huge portion of the circuit is dedicated to computing maximums, which is reduced to comparing two numbers. The common technique for comparison, also used in MPyC, involves converting numbers into binary strings [66]. In order to optimize the performance of this conversion procedure, we observed that each ρ value needs at most 5 bits.

Moreover, in order to avoid costly fixed-point operations the parties first compute the scaled noise value $\hat{e}_i = \lfloor 2^{60} \cdot e_i \rfloor$ and execute a circuit that takes only integers as input, i.e.,

$$\{(c_1^i, \dots, c_K^i), \hat{e}_i\}_{i=1}^N$$

The MPC circuit computes

$$\hat{u} = 2^{60} \cdot \sum_{j=1}^{K} (\max\{c_j^i : i \in [N]\}) + \sum_{i=1}^{N} \hat{e}_i \tag{1}$$

and each party locally computes $u = \hat{u}/2^{60}$.

Size of a local sketch and input to the circuit. Note that the local sketch that P_i holds is $(c_1^i, \dots, c_K^i)$, which is just K numbers independent of the data items that P_i collects. Moreover, as shown above, each party P_i feeds $(c_1^i, \dots, c_K^i, \hat{e}_i)$ to the circuit, which simply amounts to $K + 1$ numbers, where K is the number of buckets, which we usually set to 4096 in our experiment. In other words, the asymptotic complexity of our MPC protocol (circuit) is $\Theta(NK)$ where N is the number of parties. Note that this is independent of the size of individual parties' input data. Therefore, whether the actual number of items is 10K or 100M, the input to the MPC circuit *always consists of $K+1$ numbers* per party. This property allows optimal scalability for our mechanism. On the other hand, the circuit size increases as the number N of participating organizations grows. Our experiments show below that the MPyC framework provides good efficiency for our protocol up to about ten organizations.

6.2 Performance Results

In this section we present the results of several experiments to capture the accuracy and performance tradeoffs of our protocol. For all the results reported below, we run the experiment with the specified settings 10 times and report the average statistics of these 10 runs.

Data sets. We use two different data sets in our experiments. Our first data consists of the publicly available University of Michigan internet scan data [23]. This data

Input size	10K	100K	1M	10M	100M
#unique	29989	299036	2000000	21640641	42261769

Table 1. IPv4 addresses when running experiments with 3 parties. Each party has the same input data size. For example, for data size 10K, *each* party holds 10K IP addresses, and together they contain 29989 *unique* IP addresses.

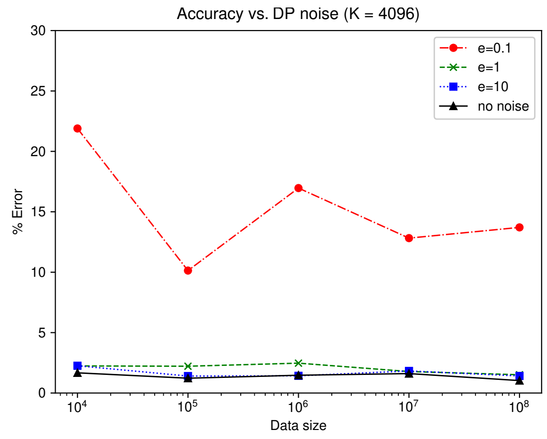

Fig. 4. Accuracy vs. DP noise. The protocol improves as we increase the differential privacy budget. This is expected since we add lower amount of noise as ϵ grows. We run this experiment for different input data sizes from 10K to 100M with K = 4096.

set contains the IPv4 addresses scanned by University of Michigan project during December 2013 and January 2014. The input files contain specified numbers of IPv4 addresses. Raw text data files containing 10K, 100K, 1 million, 10M, and 100M IPv4 addresses were created from randomly sampling downloaded IP addresses. See Table 1.

The second data set we use is the 1998 FIFA world cup (soccer) web site daily access logs data set [4, 6]. This data set consists of all the requests made to the 1998 World Cup Web site between April 30, 1998 and July 26, 1998. We use the tool provided by the data owner to extract the client ID for arbitrarily chosen access logs. We further split the extracted data in different files of sizes 10K, 100K and 1M which are used as inputs of the parties running the MPC protocol.

Privacy loss parameter ϵ vs. accuracy. Our first experiment aimed to analyze the trade-off between the differential privacy budget ϵ and the accuracy achieved by our protocol. We used the data set of IPv4 addresses. For this experiment, we held the number of MPC parties fixed at three. Figure 4 shows the relationship between the input data size, the differential privacy parameter ϵ and the accuracy of approximate count. It can be seen that for all input sizes the accuracy improves (error reduces) as the privacy loss parameter ϵ increases. We measure the % error with respect to the actual count of distinct elements in parties input. For example, if the

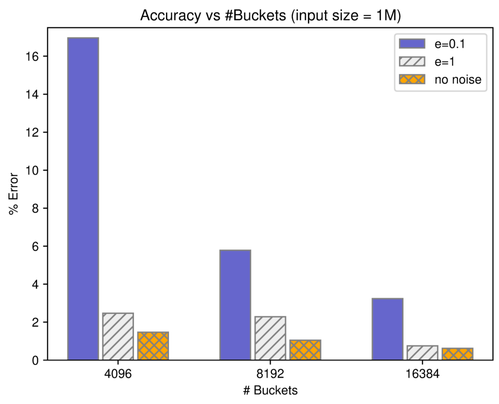

Fig. 5. Accuracy vs. number of buckets. The accuracy of the protocol improves with increase in the number of buckets. For input data size 1M and $\epsilon = 0.1$ when K increases from 4096 to 8192, the average error drops from 17% to 5%.

estimate is 105 and the actual count is 100, we would report the error as 5%. We fix the number of buckets $K = 4096$ for this experiment.

We note that, Figure 4 shows that the algorithm does not guarantee monotonically better accuracy according to the number of items. One can verify this for every privacy loss parameter (even for the case without noise, i.e., the black line with triangles). The case with a larger amount of noise (i.e., the red line with circles) shows this trend more clearly. This is due to the fact that Theorem 4.1 only claims that the accuracy error converges to about $2/\sqrt{K} \approx 3\%$ with high probability and may vary before it does so. Our experiment shows that the algorithm has slightly better accuracy in practice (2% or less) for larger ϵ.

Accuracy vs. number of buckets. Our next experiment studies the trade-off between the number of buckets (i.e. the parameter K) and the achieved accuracy. Figure 5 shows the trade-off between accuracy of the protocol and the number of buckets for various values of ϵ and for different input data sizes. It is easy to see that the error accuracy improves significantly as the number of buckets grows. Since the MPC circuit grows linearly in the bucket size, the run time increases as the bucket size grows.

Run time vs. input data size. We conducted additional experiments with the data set of IPv4 addresses where parties have different input sizes and measured the run time. We ran the protocol between 3 parties where the input data size for each party is chosen from $\{10K, 100K, 1M\}$, we kept the parameter $\epsilon = 1$ and $K = 4096$ fixed. We ran these experiments on a Macbook Pro with 2.6 GHz Intel i5 processor with 4 cores and 16GB DDR RAM.

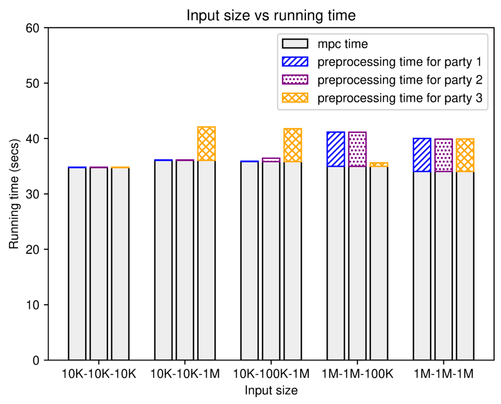

Fig. 6. Run time vs. input data size. The run time of the MPC protocol is independent of the input data size of each party. The local pre-processing time for individual party scales linearly with input data size. Here, on x-axis, 10K-100K-1M represents the experiment with $N = 3$ parties where parties P_1, P_2, and P_3 holds input data of sizes $10K, 100K$, and $1M$ respectively.

As shown in Figure 6, the run time of the MPC part is independent of the individual parties' input sizes. This is due to the fact that the low space complexity of the LogLog sketch allows us to use the same sketch size, and thus MPC input size, for all input sizes we consider. On the other hand, the local pre-processing time, although much shorter than the MPC time, scales linearly with individual input data size since each party needs to hash its own data to compute the local sketches.

We note that while we do not experimentally evaluate the cost of a straightforward MPC calculation of unique count without sketching, it is clear to see that it will be many times slower than the sketch-based approach. A direct implementation of unique count requires sorting or at least $\Omega(n \log n)$ secure comparisons for n inputs, so for $1M$ inputs per party ($3M$ total) this will require 60 million secure comparisons as compared to $KN \approx 12,000$ comparisons for our protocol.

Run time overhead for noise sampling. Recall that the noise sampling in our protocol is performed by simply computing $\sum_{i=1}^{N} e_i$ in the MPC circuit. To measure the overhead of this part, we compared the run time of the protocol with noise sampling and the one without, using the configurations in the above experiment (i.e., run time vs. input data size). In all configurations, the cost of noise sampling was very small taking at most 1.36 seconds.

Number of buckets vs. MPC run time. In order to experimentally verify that the MPC run time grows linearly with number of buckets K as shown in Equation (1), we conducted experiments for $\epsilon = 1$, and input data sizes in $\{10K, 100K, 1M\}$ for 3 parties with different values of $K \in \{4096, 8192, 16384\}$ and measured the run time of the MPC protocol. Table 2 shows the resulting MPC runtimes for these experiments. As expected, the average MPC runtime scales linearly with the bucket size. We used the world cup data set for conducting these experiments and the same Macbook Pro machine used above.

K	●-●-●	●-●-◇	●-⋆-◇	◇-◇-⋆	◇-◇-◇
4096	33.91	34.57	34.50	33.39	33.68
8192	72.10	70.99	71.27	70.57	70.96
16384	147.79	144.73	144.59	145.52	147.70

Table 2. Comparison of runtime (in seconds) of MPC protocol with DP parameter $\epsilon = 1$. We vary the input data size and bucket size K. All the reported values are average run time of MPC protocol in seconds for $N = 3$ parties. Columns 2-6 show different input data sizes, $\bullet = 10K$, $\star = 100K$, and $\diamond = 1M$.

No. of parties	3	5	7	9
Runtime (in sec)	17	48	92	183

Table 3. Comparison of runtime of MPC protocol. Input data size = 1 Million, DP parameter $\epsilon = 1$.

Number of parties vs. MPC run time. Finally, we analyze the run time of our protocol as a function of the number of MPC parties. Table 3 shows the runtime of MPC protocols for different numbers of parties each holding input of size 1 million and $\epsilon = 1$. We ran the protocol on cluster of 64 Intel® Xeon® E5-2697A v4 CPUS each with 16 cores operating at 2.6 GHz and 380GB memory.

Increasing the number of parties causes a slow down for two reasons: First, the runtime of executing the MPC for the Merge circuit depends on the number of local sketches (and hence the number of MPC parties each of which supplies a sketch), which increases linearly with the number of parties. Second, as the number of parties grows, the overhead of the underlying MPC protocol also grows (even when circuit size is fixed).

7 Framework for Distributed Private Sketching

We now show how we can generalize our results to give a construction for private distributed sketching for a large class of sketch-based approximations. Specifically, we show (roughly) that for any sketch with a good accuracy guarantee there is a private distributed mechanism that requires the addition of only a small amount of noise that depends on the accuracy of the sketch and the global sensitivity of the function being approximated.

We first describe key technical insights. We first note that accuracy and sensitivity of a sketch go together. That is, if a sketching algorithm approximates a functionality f that has low sensitivity (the output of the functionality differs by a bounded amount on any two neighboring inputs), then we can use the *accuracy* of the sketching algorithm to bound the sensitivity of the output of the sketch. Once we bound the sensitivity, we can then apply known DP mechanisms to obtain differential privacy.

For most sketches in the literature, however, the accuracy achieved by the sketch is *multiplicative*, meaning that, on any input, the output of the sketch is guaranteed to be within a $(1 \pm \tilde{\epsilon})$ multiplicative factor of the correct answer with high probability (over randomness of the sketch). On the other hand, the Laplace mechanism [26] is easier to analyze and achieve better concrete parameters when the sensitivity is an *additive* constant. We resolve this discrepancy by having our MPC protocol add noise *in the exponent.* That is, instead of adding some noise e to the output of the sketch, the MPC protocol will multiply the output of the sketch by $2^{e'}$—where the distribution of e' depends on the accuracy of the sketch and the sensitivity of the functionality—effectively adding e' noise to the *logarithm* of the output of the sketch.

Now, we begin with some terminology.

Definition 7.1. *A* sketch *is a pair of algorithms* (Sketch, Evaluate).
- Sketch *takes an input sequence X and outputs a data structure $S(X)$.*
- Evaluate *takes $S(X)$ and outputs a value $y \in \mathbb{R}^+$*

We will often abuse notation, using S to refer to both the sketch, and the data structure output by Sketch.

Definition 7.2. *We say that a sketch $S(X)$ is an $(\tilde{\epsilon}, \tilde{\delta})$-*accurate *approximation of a function f if, letting $y =$* Evaluate$(S(X))$*, for all sequences of inputs X, we have that* $\Pr[(1-\tilde{\epsilon})f(X) < y < (1+\tilde{\epsilon})f(X)] > 1-\tilde{\delta}$.

Definition 7.3. *We say that a sketch S is* mergeable *if there exists a merge procedure* Merge *that takes N sketches and outputs a single sketch. Formally, for any input sequence X partitioned into N subsequences $X := X_1, \ldots, X_N$, $S(X) =$* Merge$(S(X_1), \ldots, S(X_N))$.

We now bound the global sensitivity of the logarithm of a sketch evaluation. Looking forward, the reason that we bound the logarithm of the sketch evaluation, rather than the evaluation itself is that our algorithm will add noise in the exponent. This is done to account for the multiplicative error of the sketch; the traditional differential privacy setting usually considers only additive error.

```
Participants: There are $N$ parties $P_1, \ldots, P_N$.
Each party $P_i$ for $i \in [N]$ inputs a sequence $X_i$ of elements,
    i.e., $X_i = (x_i^1, x_i^2, \ldots)$. Also, a budget $\epsilon$ for differential
    privacy is specified.
Output: An approximation for some function $f(X)$.

1. Each party chooses a uniform random seed $\mathsf{seed}_i \leftarrow$
   $\{0,1\}^\lambda$. The parties then compute $\mathsf{seed} = \oplus_{i=1}^N \mathsf{seed}_i$.
2. Each $P_i$ locally computes the following:
   (a) Computes $S_i = \mathsf{Sketch}(X_i)$ algorithm using
       $H(\mathsf{seed}||\cdot)$ as a random oracle and $H(\mathsf{seed}||0||\cdot)$
       for computing any additional randomness necessary for constructing the sketch.
   (b) Generate noise $e'_i := (\Delta S/\epsilon) \cdot (a_i - b_i)$, where $a_i$
       and $b_i$ are i.i.d. random variables drawn from the
       Gamma distribution $\Gamma(2/N, 1)$, using randomness $r_i$ (from $P_i$'s random tape) for sampling.
       Let $e_i = 2^{e'_i}$.
3. Parties execute a secure MPC protocol to compute
   the following
   – Input: $(S_1, \ldots, S_N), (e_1, \ldots, e_N)$
   – Compute $y' = \mathsf{Evaluate}(\mathsf{Merge}(S_1, \ldots, S_N))$
   – Output: $y = y' \cdot \left(\Pi_{i=1}^n e_i\right)$.
```

Fig. 7. $\Pi_{PrivateSketch}$: Distributed sketch framework.

Lemma 7.4. *Assume that a sketch S is an $(\tilde{\epsilon}, \tilde{\delta})$-accurate approximation of a function $f(X)$ outputting a positive number. Define $Y(X) =$* Evaluate$(S(X))$ *and let $g(X) = \log_2(Y(X))$ be the random variable corresponding to the logarithm of the evaluation of sketch S on X. Then, for any two neighboring inputs X_1, X_2,*

$$\Pr\left[|g(X_1) - g(X_2)| < \log(1 + \frac{2\tilde{\epsilon}}{1-\tilde{\epsilon}}) \cdot (1 + \frac{\Delta f}{\min(f)})\right] \geq 1 - 2\tilde{\delta}$$

where Δf is the global sensitivity of f, and $\min(f)$ is the minimum value attained by f on any input X.

Proof. By the accuracy of the sketch, we have that for any neighboring inputs X_1 and X_2 with probability $1 - 2\tilde{\delta}$, we have:

$$\frac{Y(X_1)}{Y(X_2)} \leq \frac{(1+\tilde{\epsilon})f(X_1)}{(1-\tilde{\epsilon})f(X_2)} \leq \frac{(1+\tilde{\epsilon})(\Delta f + f(X_2))}{(1-\tilde{\epsilon})f(X_2)} \leq (1 + \frac{2\tilde{\epsilon}}{1-\tilde{\epsilon}}) \cdot (1 + \frac{\Delta f}{f(X_2)})$$

Taking the logarithm of this equation, we conclude that the lemma holds. □

Definition 7.5. *For a sketch S as in Lemma 7.4, we define its global sensitivity,*

$$\Delta S = \log(1 + 2\widetilde{\epsilon}/(1-\widetilde{\epsilon})) + \log(1 + \frac{\Delta f}{\min(f)}).$$

We now use this lemma to construct a distributed DP sketch described in Figure 7. But, before doing so, we need an $(\widetilde{\epsilon}, \widetilde{\delta})$-approximation algorithm with $\widetilde{\delta} = \mathsf{negl}(\lambda)$. This can be achieved through the median technique [8].

Lemma 7.6. *Suppose there is a sketching algorithm S that is a $(\widetilde{\epsilon}, \widetilde{\delta})$-accurate approximation of a function f with $\widetilde{\delta} \le 1/4$. Then, there is a sketching algorithm S' approximating f with $(\widetilde{\epsilon}, \mathsf{negl}(\lambda))$-accuracy.*

The proof of the lemma is provided in Appendix C.

We can now state our main result in this section. But, before doing so, we define some parameters:

– $(\widetilde{\epsilon}, \widetilde{\delta})$: The accuracy parameters of the underlying *mergeable* sketch. We require $\widetilde{\delta}$ to be negligible in λ.
– (ϵ, δ): The DP parameters for the protocol. We require δ to be negligible in λ.
– $(\overline{\epsilon}, \overline{\delta})$: The accuracy of the output of the protocol, relative to the output of the sketch (i.e., the additional accuracy error on top of the original error of the sketch).

We further require the following relationship among the parameters:

$$\widetilde{\epsilon}/(1-\widetilde{\epsilon}) \le \frac{(\overline{\epsilon} - \overline{\epsilon}^2/2)\cdot\epsilon}{4\ln(2/\overline{\delta})} - \frac{\Delta f}{2\min(f)},$$

We next present the main theorem of this section, which states the accuracy, privacy, and efficiency parameters of our protocol, in terms of the above parameters.

Theorem 7.7. *For parameters $(\widetilde{\epsilon}, \widetilde{\delta})$, (ϵ, δ), $(\overline{\epsilon}, \overline{\delta})$ satisfying the above requirements, the protocol $\Pi_{PrivateSketch}$ in Figure 7 is a distributed private sketch secure against a semi-honest adversary controlling any coalition C, with $|C| < N/2$, with the following properties:*

1. *Accuracy: $\Pi_{PrivateSketch}$ is a $(\overline{\epsilon} + \widetilde{\epsilon} + \overline{\epsilon}\cdot\widetilde{\epsilon}, \overline{\delta} + \widetilde{\delta})$-approximation of f.*
2. *Privacy: $\Pi_{PrivateSketch}$ achieves (ϵ, δ)-DP.*
3. *Run time: The run time depends only on the size of C_{Merge} and $C_{\mathsf{EvaluateMech}}$ and is independent of the size of the input.*

The proof is given in Appendix D.

Example of a concrete instantiation. We briefly describe the resulting parameter settings when instantiating our framework with the LogLog sketch. Note that the concrete parameters in the previous section beat the parameters described here and we present these parameters settings for illustrative purposes only. Assuming that our underlying LogLog sketch implementation achieves $\widetilde{\epsilon} := 0.01$ and negligible $\widetilde{\delta}$ by applying Lemma 7.6 (i.e., the size of sketch will blow up by $O(\lambda)$ times). We set $\overline{\epsilon} := 0.09$ and $\overline{\delta} = 0.1$, which means that our final accuracy of the MPC output is within about 10% accuracy with 90% probability. We also assume that $\min(f) \ge 1000$, since we assume that at least one of the parties inputs a set of size at least 1000 (otherwise we are not in the big-data setting). Moreover, $\Delta f = 1$. For these settings, we can achieve (ϵ, δ)-DP, where δ is negligible as long as ϵ satisfies:

$$0.01/(1-0.01) \le \frac{(0.09 - 0.09^2/2)\cdot\epsilon}{4\ln(2/0.1)} - \frac{1}{2000}.$$

Thus, we may set $\epsilon \approx 1.5$.

7.1 Example Instantiations

To justify the usefulness of our general framework we show that several well-known sketching algorithms satisfy its requirements.

LogLog sketch. The first sketching algorithm we consider is the LogLog sketch studied in the previous sections (See Figure 1). We observe that the Sketch algorithm can be computed locally after the seed for H is chosen, so each party P_i can build a sketch S_i of his own data. Next, the Merge algorithm (given in Section 6) computes the max value (among N values) for each of its K counters. This can be done in linear time in the number of counters, K, and the number of parties N. Importantly, this is far less than the total number of input values. Finally, the $\mathsf{Evaluate}$ method computes $\alpha \cdot K \cdot 2^{u/K}$, which has a cost independent of the input size. Thus, the LogLog sketch can be plugged into our framework to achieve a distributed private variant. We also note that the global sensitivity of the unique count function is 1. So, if we require that all possible inputs have above a certain number of minimum elements, the noise added $(\Delta S/\min(f))$ is quite small.

We note however, that the protocol resulting from this general framework has worse parameters than the optimized protocol presented earlier (See Section 6). Furthermore, the optimized variant allows a hash function to be chosen once and for all, allowing for repeated input phases, whereas the general framework requires that inputs be fixed before the hash function is fixed to prevent the adversary finding a “bad” hash input.

AMS sketch. We next consider the well-known Alon-Matias-Szegedy (AMS) sketch [5] for approximating the L_2-norm of the frequency vector of a stream of data items. Briefly, for a universe of size m, the AMS sketch chooses k vectors $(r_1^j, \ldots, r_m^j)$, $j \in [k]$ independently with $\Pr[r_i^j = 1] = \Pr[r_i^j = -1] = .5$ (these can be chosen using the hash function H and do not need to be stored). Next upon observing streaming values x_i, Sketch computes $Z^j = \sum r_i^j x_i$, $j \in [k]$. Note that this sketch is linear and thus mergeable. Finally, the Evaluate procedure simply computes $\sum_{j\in[k]}(Z^j)^2$.

JL Transform. The Johnson-Lindenstrauss Transform (JLT) is an important tool in the sketching literature [46]. It is a linear sketching algorithm that can be used to embed a high dimensional space into a low dimensional space, while preserving pairwise distances. It has many applications (cf. [3, 21, 59]), including providing another method for approximating the L_2 norm of a frequency vector of a stream of data items. Since the sketch is *linear*, it is trivially mergeable.

Let X be a high dimensional (dimension n) vector corresponding to the frequency vector of a stream of data items. To compute the transform on a vector X, one samples a matrix $\Pi \in \mathbb{R}^{m\times n}$, where $m \ll n$ and outputs ΠX. The product ΠX can be computed in a streaming fashion, since the columns of Π can be sampled on-the-fly. The dimension m of the resulting sketch depends on the desired $(\tilde{\epsilon}, \tilde{\delta})$-accuracy, but does not depend on the dimension n of the original data. There are various ways to sample Π, we present one such method:

Definition 7.8 (JL Transform ($(\widetilde{\epsilon}, \widetilde{\delta}, N)$-JLT)). *Let $X \in \mathbb{R}^n$ be a finite set with $|X| = N$. A random matrix $\mathbf{\Pi} \in \mathbb{R}^{m\times n}$ is called* Johnson-Lindenstrauss Transform ($(\widetilde{\epsilon}, \widetilde{\delta}, N)$-JLT)*, if for all unit norm vectors $\mathbf{x}, \mathbf{x}' \in X$ following holds with probability at least $1 - \widetilde{\delta}$:*

$$\|\mathbf{\Pi x}\|^2 = (1\pm\varepsilon) \text{ and, } \|\mathbf{\Pi}(\mathbf{x}+\mathbf{x}')\|^2 = (1 \pm \widetilde{\epsilon})\|(\mathbf{x} + \mathbf{x}')\|^2.$$

Theorem 7.9 (Originally in [42]). *Let $\mathbf{\Pi}$ be the scaled matrix (multiply by $\frac{1}{\sqrt{m}}$) with i.i.d. standard Gaussian random variables and $\widetilde{\epsilon}, \widetilde{\delta} \in (0, 1)$. If $m = \Omega\left(\left(\frac{1}{\widetilde{\epsilon}^2}\right)\left(\log\frac{N}{\widetilde{\delta}}\right)\right)$, then $\mathbf{\Pi}$ is $(\widetilde{\epsilon}, \widetilde{\delta}, N)$-JLT.*

There have been results showing that various forms of the JL transform are differentially private, without addition of noise to the output [14, 62, 68]. It may therefore seem that the techniques provided by our framework are redundant for the above application. However, we emphasize that those results hold only *when the JLT matrix* Π *is private.* Differential privacy *does not hold* if the matrix Π is publicly released. In our distributed setting, all parties must use the same JLT matrix Π in order for the sketches to be mergeable. Therefore, we must assume that the JLT matrix Π is *public* and those prior results do not hold. This is why addition of noise via an MPC protocol is necessary in our setting.

8 Conclusion and Future Work

In summary, the contributions of this work consist of: (1) We show that the LogLog algorithm is *itself* differentially private (DP) in the single party setting, *even without adding noise*, when the hash function is private and random. (2) We achieve DP disjoint counting in the multiparty setting in the semi-honest model assuming an honest majority, by leveraging the *mergeability* of the the LogLog sketch. We implement our MPC protocol and analyze the resulting privacy and accuracy parameters. (3) We propose a general framework for efficiently combining a sketching algorithm that is *mergeable*, a DP mechanism and an MPC protocol.

In future work, we plan to optimize our MPC protocol for settings in which there are a large number of parties. We will also explore new application domains for our framework, that go beyond computing statistics. For example, an interesting question is whether by using suitable mergeable sketches and efficient noise sampling techniques in conjunction with machine learning algorithms, we can obtain efficient distributed and differentially private machine learning algorithms.

9 Acknowledgments

Seung Geol Choi was supported in part by NSF #CNS-1618269, ONR #N0001419WX00568, and ONR #N0001419WX01032. Dana Dachman-soled was supported in part by NSF grants #CNS-1933033, #CNS-1840893, #CNS-1453045 (CAREER), by a research partnership award from Cisco and by Financial assistance award 70NANB15H328 from the U.S. Department of Commerce, National Institute of Standards and Technology. Work of Mukul Kulkarni was done in part while he was a student at the University of Maryland. Arkady Yerukhimovich was supported in part by NSA #H98230-19-1-0320.

References

[1] Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors. *ACM CCS 14*. ACM Press, November 2014.

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Yuval Rabani, editor, *23rd SODA*, pages 459–467. ACM-SIAM, January 2012.

[3] Nir Ailon and Bernard Chazelle. The fast johnson–lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on computing*, 39(1):302–322, 2009.

[4] Mohammad Alaggan, Mathieu Cunche, and Sébastien Gambs. Privacy-preserving wi-fi analytics. *Proceedings on Privacy Enhancing Technologies*, 2018(2):4–26, 2018.

[5] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[6] M. Arlitt and T. Jin. 1998 world cup web site access logs, August 1998. Available at http://www.acm.org/sigcomm/ITA/.

[7] Vikas G. Ashok and Ravi Mukkamala. A scalable and efficient privacy preserving global itemset support approximation using bloom filters. In *Data and Applications Security and Privacy XXVIII - 28th Annual IFIP WG 11.3 Working Conference, DBSec 2014, Vienna, Austria, July 14-16, 2014. Proceedings*, pages 382–389, 2014.

[8] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques, 6th International Workshop, RANDOM 2002, Cambridge, MA, USA, September 13-15, 2002, Proceedings*, pages 1–10, 2002.

[9] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. Shrinkwrap: efficient sql query processing in differentially private data federations. *Proceedings of the VLDB Endowment*, 12(3):307–320, 2018.

[10] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Review*, 56(2):315–334, 2014.

[11] Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 451–468. Springer, Heidelberg, August 2008.

[12] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *28th ACM STOC*, pages 47–55. ACM Press, May 1996.

[13] Lakshminath Bhuvanagiri and Sumit Ganguly. Estimating entropy over data streams. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 148–159, 2006.

[14] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. The johnson-lindenstrauss transform itself preserves differential privacy. In *53rd FOCS*, pages 410–419. IEEE Computer Society Press, October 2012.

[15] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[16] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[17] Michael S. Crouch and Andrew McGregor. Periodicity and cyclic shifts via linear sketches. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 158–170, 2011.

[18] Damien Desfontaines, Andreas Lochbihler, and David A. Basin. Cardinality estimators do not preserve privacy. *PoPETs*, 2019(2):26–46, 2019.

[19] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320, 2004.

[20] C. Dong and G. Loukides. Approximating private set union/intersection cardinality with logarithmic complexity. *IEEE Transactions on Information Forensics and Security*, 12(11):2792–2806, Nov 2017.

[21] Petros Drineas, Michael W Mahoney, S Muthukrishnan, and Tamás Sarlós. Faster least squares approximation. *Numerische mathematik*, 117(2):219–249, 2011.

[22] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, pages 605–617, 2003.

[23] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 13th Internet Measurement Conference*, October 2013.

[24] Cynthia Dwork. Differential privacy (invited paper). In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 1–12. Springer, Heidelberg, July 2006.

[25] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 486–503. Springer, Heidelberg, May / June 2006.

[26] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 265–284. Springer, Heidelberg, March 2006.

[27] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[28] Rolf Egert, Marc Fischlin, David Gens, Sven Jacob, Matthias Senker, and Jörn Tillmanns. Privately computing set-union and set-intersection cardinality via bloom filters. In *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*, pages 413–430, 2015.

[29] Tariq Elahi, George Danezis, and Ian Goldberg. PrivEx: Private collection of traffic statistics for anonymous communication networks. In Ahn et al. [1], pages 1068–1079.

[30] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In Ahn et al. [1], pages 1054–1067.

[31] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends in Privacy and Security*, 2(2-3):70–246, 2018.

[32] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. Secure multi-party computation of approximations. *ACM Trans. Algorithms*, 2(3):435–472, 2006.

[33] Ellis Fenske, Akshaya Mani, Aaron Johnson, and Micah Sherr. Distributed measurement with private set-union cardinality. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 2295–2312. ACM Press, October / November 2017.

[34] Philippe Flajolet, Eric Fusy, Olivier Gandouet, and Frederic Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *IN AOFA 2007: PROCEEDINGS OF THE 2007 INTERNATIONAL CONFERENCE ON ANALYSIS OF ALGORITHMS*, 2007.

[35] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[36] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Heidelberg, February 2005.

[37] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[38] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. *To appear in IEEE S&P*, 2020.

[39] Shai Halevi, Robert Krauthgamer, Eyal Kushilevitz, and Kobbi Nissim. Private approximation of NP-hard functions. In *33rd ACM STOC*, pages 550–559. ACM Press, July 2001.

[40] Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 683–692, 2013.

[41] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[42] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *30th ACM STOC*, pages 604–613. ACM Press, May 1998.

[43] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 202–208. ACM Press, May 2005.

[44] Rob Jansen and Aaron Johnson. Safely measuring tor. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1553–1567. ACM Press, October 2016.

[45] Rajesh Jayaram and David P. Woodruff. Perfect lp sampling in a data stream. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 544–555, 2018.

[46] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space, 1984.

[47] Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? In *49th FOCS*, pages 531–540. IEEE Computer Society Press, October 2008.

[48] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[49] Changchang Liu, Prateek Mittal, and Supriyo Chakraborty. Dependence makes you vulnberable: Differential privacy under dependent tuples. In NDSS 2016 [56].

[50] Yang Liu, Wenji Chen, and Yong Guan. Identifying high-cardinality hosts from network-wide traffic measurements. *IEEE Trans. Dependable Sec. Comput.*, 13(5):547–558, 2016.

[51] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *48th FOCS*, pages 94–103. IEEE Computer Society Press, October 2007.

[52] Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics with succinct sketches. In NDSS 2016 [56].

[53] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil P. Vadhan. Computational differential privacy. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 126–142. Springer, Heidelberg, August 2009.

[54] Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error l_p-sampling with applications. In Moses Charika, editor, *21st SODA*, pages 1143–1160. ACM-SIAM, January 2010.

[55] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. *TOSN*, 4(2):7:1–7:40, 2008.

[56] *NDSS 2016*. The Internet Society, February 2016.

[57] Aleksandar Nikolov, Kunal Talwar, and Li Zhang. The geometry of differential privacy: the sparse and approximate cases. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 351–360. ACM Press, June 2013.

[58] Nikos Ntarmos, Peter Triantafillou, and Gerhard Weikum. Counting at large: Efficient cardinality estimation in internet-scale data networks. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 40, 2006.

[59] Tamás Sarlós. Improved approximation algorithms for large matrices via random projections. In *47th FOCS*, pages 143–152. IEEE Computer Society Press, October 2006.

[60] Berry Schoenmakers. Mpyc - secure multiparty computation in python. *GitHub*, 2018. https://github.com/lschoe/mpyc.

[61] Adam Sealfon. Shortest paths and distances with differential privacy. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 29–41, 2016.

[62] Or Sheffet. Differentially private ordinary least squares. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 3105–3114, 2017.

[63] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, and Dawn Song. Distributed private data analysis: Lower bounds and practical constructions. *ACM Trans. Algorithms*, 13(4):50:1–50:38, 2017.

[64] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In László Babai, editor, *36th ACM STOC*, pages 81–90. ACM Press, June 2004.
[65] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011.
[66] Thomas Toft. *Primitives and applications for multi-party computation*. PhD thesis, Aarhus Universitet, Denmark, 2007.
[67] Florian Tschorsch and Björn Scheuermann. An algorithm for privacy-preserving distributed user statistics. *Computer Networks*, 57(14):2775–2787, 2013.
[68] Jalaj Upadhyay. Differentially private linear algebra in the streaming model. *CoRR*, abs/1409.5414, 2014.
[69] Ryan Wails, Aaron Johnson, Daniel Starin, Arkady Yerukhimovich, and S. Dov Gordon. Stormy: Statistics in tor by measuring securely. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 615–632, 2019.
[70] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Proof of Theorem 4.2

The proof uses the following lemma.

Lemma A.1. *Let $M : \{0,1\}^* \to \{0,1\}^\ell$ be a mechanism. Let $\mathsf{Bd}_{M,x}$ be a (bad) event that occurs when mechanism M is run on input x. Suppose that for all neighboring data sets D and D', and for all possible outputs s from M, it holds*

$$\Pr[M(D) = s \wedge \overline{\mathsf{Bd}_{M,D}}] \leq (1+\epsilon)\Pr[M(D') = s \wedge \overline{\mathsf{Bd}_{M,D'}}].$$

Then, for all neighboring data sets D and D', and for all $\mathcal{S} \subseteq \{0,1\}^\ell$, it holds that

$$\Pr[M(D) \in \mathcal{S}] \leq e^\varepsilon \Pr[M(D') \in \mathcal{S}] + \Pr[\mathsf{Bd}_{M,D}].$$

Proof. First note that if for all neighboring D, D' and all $s \in \{0,1\}^\ell$

$$\Pr[M(D) = s \wedge \overline{\mathsf{Bd}_{M,D}}] \leq (1+\epsilon)\Pr[M(D') = s \wedge \overline{\mathsf{Bd}_{M,D'}}]$$

then for all neighboring D, D' and all $\mathcal{S} \subseteq \{0,1\}^\ell$

$$\Pr[M(D) \in \mathcal{S} \wedge \overline{\mathsf{Bd}_{M,D}}] \leq (1+\epsilon)\Pr[M(D') \in \mathcal{S} \wedge \overline{\mathsf{Bd}_{M,D'}}].$$

Therefore, the lemma follows from the following:

$$\begin{aligned}
&\Pr[M(D) \in \mathcal{S}] \\
&= \Pr[M(D) \in \mathcal{S} \wedge \overline{\mathsf{Bd}_{M,D}}] + \Pr[M(D) \in \mathcal{S} \wedge \mathsf{Bd}_{M,D}] \\
&\leq (1+\epsilon)\Pr[M(D') \in \mathcal{S} \wedge \overline{\mathsf{Bd}_{M,D'}}] + \Pr[\mathsf{Bd}_{M,D}] \\
&\leq (1+\epsilon)\Pr[M(D') \in \mathcal{S}] + \Pr[\mathsf{Bd}_{M,D}] \\
&\leq e^\varepsilon \Pr[M(D') \in \mathcal{S}] + \Pr[\mathsf{Bd}_{M,D}].
\end{aligned}$$

The first inequality holds from the assumption. For the second inequality, we used the fact that all $x \in \mathbb{R}$ we have $(1+x) \leq e^x$. □

A.1 Proof of Lemma 4.4

$$\begin{aligned}
&\left(1 - \frac{1}{2^{s-1}}\right)^{-1} \cdot \Pr[C(n+1,m) = s] \\
&= \left(1 - \frac{1}{2^{s-1}}\right)^{-1} \cdot \left(\left(1 - \frac{1}{2^s}\right)^{n+1} - \left(1 - \frac{1}{2^{s-1}}\right)^{n+1}\right) \\
&> \left(1 - \frac{1}{2^s}\right)^{-1} \cdot \left(1 - \frac{1}{2^s}\right)^{n+1} - \left(1 - \frac{1}{2^{s-1}}\right)^{n} \\
&= \Pr[C_m(n) = s].
\end{aligned}$$

Therefore, we have $\Pr[C_m(n) = s] \leq \left(1 - \frac{1}{2^{s-1}}\right)^{-1} \cdot \Pr[C(n+1,m) = s]$. The lemma holds from the following:

$$\frac{1}{1 - \frac{1}{2^{s-1}}} = \frac{2^{s-1}}{2^{s-1} - 1} = 1 + \frac{1}{2^{s-1} - 1} \leq 1 + \frac{2}{2^{s-1}} \quad .$$

A.2 Proof of Theorem 4.2

The distribution of the output of the LogLog algorithm. Assuming the random oracle, if we strip out the post-processing, the output of the LogLog algorithm applied to D with n distinct elements is identically distributed to the following experiment:

Experiment $Z_{K,m}(n)$:
1. Consider n balls and K bins. For each ball, choose a random bin in which to put the ball. Let B_j is the number of balls in the jth bin.
2. Output $C_m(B_0), \ldots, C_m(B_{K-1})$.

Since differential privacy of $Z_{K,m}$ implies differential privacy of the LogLog algorithm, it suffices to show

$$\begin{aligned}
\Pr[Z_{K,m}(n) \in \mathcal{S}] &\leq e^\epsilon \Pr[Z_{K,m}(n+1) \in \mathcal{S}] + \mathsf{negl}(\lambda), \\
\Pr[Z_{K,m}(n+1) \in \mathcal{S}] &\leq e^\epsilon \Pr[Z_{K,m}(n) \in \mathcal{S}] + \mathsf{negl}(\lambda).
\end{aligned}$$

We first show the first inequality. We will follow the strategy described in Lemma A.1. That is, we define an event Bd and show for any $\mathbf{s}$, it holds $\Pr[\mathsf{Bd}] = \mathsf{negl}(\lambda)$ and

$$\Pr[Z_{K,m}(n) = \mathbf{s} \wedge \overline{\mathsf{Bd}}] \le (1+\epsilon)\Pr[Z_{K,m}(n+1) = \mathbf{s} \wedge \overline{\mathsf{Bd}}] \tag{2}$$

Event Bd. Recall $n \ge \max\{\frac{8K\lambda}{\epsilon}, 8K\lambda\}$. Let Bd be an event in which the output $\mathbf{s}$ has some s_j less than $s^* = \log(\frac{4}{\epsilon})$. We show $\Pr[\mathsf{Bd}]$ is negligible in λ. Let Enough be an event such that $B_0 \ge 2\lambda/\epsilon$. Then, we have

$$\begin{aligned}\Pr[\mathsf{Bd}] &\le \sum_{j\in[K]} \Pr[s_j < s^*]\\ &= \sum_{j\in[K]} \Pr[C_m(B_j) < s^*]\\ &= K\cdot \Pr[C_m(B_0) < s^*]\\ &\le K\cdot(\Pr[C_m(B_0) < s^* | \mathsf{Enough}] + \Pr[\neg\mathsf{Enough}])\end{aligned}$$

Now, we have

$$\begin{aligned}&\Pr[C_m(B_0) < s^* | \mathsf{Enough}]\\ &\le \left(1 - \frac{1}{2^{s^*-1}}\right)^{B_0} \le \left(e^{-\frac{1}{2^{s^*-1}}}\right)^{B_0} \le e^{-\lambda}\end{aligned}$$

To bound $\Pr[\neg\mathsf{Enough}]$, we use the Chernoff bound. Since $\mu = \mathbf{Exp}[B_0] = n/K$, we have

$$\Pr[\neg\mathsf{Enough}] \le \Pr[B_0 < \mu/2] \le e^{-\mu/8} \le e^{-\lambda}.$$

This concludes that $\Pr[\mathsf{Bd}] = \mathsf{negl}(\lambda)$.

Showing (2). In order to show (2), we consider the following:

– In $Z_{K,m}(n)$ and $Z_{K,m}(n+1)$, let R denote random coins that choose which bins the (first) n balls should be placed in.
– Note that R doesn't include the choice of the $(n+1)^{\text{st}}$ ball in $Z_{K,m}(n+1)$. Also, R doesn't determine $C_m(B_j)$ either.

In order to prove Equation (2), it suffices to prove for every R:

$$\begin{aligned}&\Pr[Z_{K,m}(n) = \mathbf{s} \mid R \wedge \overline{\mathsf{Bd}}]\\ &\quad\le (1+\epsilon)\Pr[Z_{K,m}(n+1) = \mathbf{s} \mid R \wedge \overline{\mathsf{Bd}}]\end{aligned}$$

For $j = 0, \ldots, K-1$, let I_j be an indicator variable for the event in which the last ball would be placed in in the jth bin in experiment $Z_{K,m}(n+1)$. Fix R and let β_j be the number of balls in bin j by throwing n balls according to the random coin R. Then, we have

$$\begin{aligned}&\Pr[Z_{K,m+1}(n) = \mathbf{s} \mid R \wedge \overline{\mathsf{Bd}}]\\ &= \sum_{j=0}^{K-1} \Pr[I_j]\cdot \Pr[Z_{K,m+1}(n) = \mathbf{s} \mid I_j \wedge R \wedge \overline{\mathsf{Bd}}]\\ &= \sum_{j=0}^{K-1}\left(\frac{1}{K}\Pr[C_m(\beta_j+1) = s_j]\cdot \prod_{\substack{i\in[K]\\ i\neq j}} \Pr[C_m(\beta_i) = s_i]\right)\\ &\ge \sum_{j=0}^{K-1}\left(\frac{1}{K}\cdot\frac{1}{1+2^{-(s_j-2)}}\cdot \Pr[C_m(\beta_j) = s_j]\cdot\right.\\ &\qquad\qquad\left.\prod_{i\in[K], i\neq j} \Pr[C_m(\beta_i) = s_i]\right)\\ &\ge \sum_{j=0}^{K-1}\left(\frac{1}{K}\cdot\frac{1}{e^\epsilon}\cdot\prod_i \Pr[C_m(\beta_i) = s_i]\right)\\ &= \frac{1}{e^\epsilon}\cdot \Pr[Z_{K,m}(n) = \mathbf{s} \mid R \wedge \overline{\mathsf{Bd}}].\end{aligned}$$

The first inequality holds from Lemma 4.4. The second equality holds since we conditioned on $\overline{\mathsf{Bd}}$ (i.e., $s_j \ge \log(4/\epsilon)$); we have

$$1 + 2^{-s_j+2} \le 1+\epsilon \le e^\epsilon.$$

The other direction. We are left to show:

$$\Pr[Z_{K,m}(n+1) \in \mathcal{S}] \le e^\epsilon \Pr[Z_{K,m}(n) \in \mathcal{S}] + \mathsf{negl}(\lambda),$$

which can be similarly shown by using Lemma 4.3. □

B Proof of Theorem 6.5

Assuming semi-honest security model, for any set $\mathcal{S}$ of size $N/2$ consisting of only honest parties, the value $\sum_{i\in\mathcal{S}} e_i$ would be distributed according to $Lap((m+1)/\epsilon)$ to the adversary. (Note that to achieve this we had to add roughly twice this amount of noise to account for the fact that the adversary sees his own contribution to the noise.) This implies that the MPC protocol inside of Figure 3 realizes a functionality that is $(\epsilon, 0)$-differentially private, which is essentially the same as a single-party protocol with public hash described in Figure 2. Therefore, applying Theorem 3.5, we conclude that our protocol is distributed differentially private.

C Proof of Lemma 7.6

The median technique augments S into a mechanism S':

$S'(X)$:

1. Execute k instances of S on input X with independent randomness.
2. Output the median of the outputs from the instances.

Note that for the median technique to fail to achieve good accuracy, at least $k/2$ instances should have bad estimates. Let T_i be an indicator variable such that T_i is 1 if the i instance gives a bad estimate or 0 otherwise. Let $T = \sum_{i=1}^{k} T_i$. Note $E[T_i] < \widetilde{\delta}$ and $\mu = E[T] < k\widetilde{\delta}$. Therefore, letting $z = \frac{1}{2\widetilde{\delta}} - 1$, we have

$$\begin{aligned}\Pr[S' \text{ outputs bad estimates}] &= \Pr[T > k/2] \\ &\le \Pr[T > \mu(1+z)] \\ &\le e^{-\frac{z^2}{z+2}\mu} \\ &\le e^{-k/3}.\end{aligned}$$

The second inequality is from the Chernoff bound, and the last inequality holds since we have $\frac{z^2}{z+2}\mu \ge k/3$ for any $\widetilde{\delta} < 1/4$. Setting $k = \Theta(\lambda)$ will satisfy the lemma.

D Proof of Theorem 7.7

We prove Theorem 7.7 following the proof outline given in Theorem 3.5. First, we define an ideal functionality in Figure 8 and show that this functionality achieves ϵ-differential privacy. Then, we show that our protocol securely realizes this functionality against any coalition C of size at most $t < N/2$.

We consider an intermediate hybrid model, where the MPC protocol in Figure 7, that takes as input $(S_1, \dots, S_N), (e_1, \dots, e_N)$, computes $y' = \mathsf{Evaluate}(\mathsf{Merge}(S_1, \dots, S_N))$ and outputs $y = y' \cdot (\Pi_{i=1}^{n} e_i)^{\Delta S/\epsilon}$ to each party is replaced with an ideal functionality $\mathcal{F}_{\mathsf{Merge}}$.

Differential privacy of the ideal world. We first show that the ideal functionality described in Figure 8. The differential privacy of the ideal follows from the global sensitivity and differential privacy of Laplace mechanism.

We assume that the adversary corrupts a coalition C s.t. $|C| = N/2$ to simplify our notation. Differential privacy in the case that there are fewer corruptions follows immediately. Recall that the view of the adversary in the Ideal world consists of the output y, as well as the randomness $[r_i]_{P_i \in C}$ and the seed seed. We therefore analyze differential privacy, conditioned on a fixed

Input: Each party P_i for $i \in [N]$ inputs a sequence X_i of elements, i.e., $X_i = (x_i^1, x_i^2, \dots)$. Also, a budget ϵ for differential privacy is specified.

Output: An approximation for some N-input function $f(X_1, \dots, X_N)$.

1. Choose $\mathsf{seed} \leftarrow \{0,1\}^\lambda$
2. Computes $S_i = \mathsf{Sketch}(X_i)$ algorithm using $H(\mathsf{seed}||\cdot)$ as a random oracle (the random oracle is also used to generate any additional randomness necessary for constructing the sketch).
3. Generate noise $e'_i := (\Delta S/\epsilon) \cdot (a_i - b_i)$, where a_i and b_i are i.i.d. random variables drawn from the *Gamma distribution* $\Gamma(2/N, 1)$, using randomness r_i for sampling. Let $e_i = 2^{e'_i}$.
4. Compute $y' = \mathsf{Evaluate}(\mathsf{Merge}(S_1, \dots, S_N))$.
5. To each party $i \in [N]$, output $y = y' \cdot \left(\Pi_{i=1}^{n} e_i\right)$ as well as their randomness r_i.

Fig. 8. Ideal Functionality $\mathcal{F}$ for Distributed Sketch Framework.

seed and fixed values of $[r_i]_{P_i \in C}$, from which the values of $[e_i]_{P_i \in C}$ and $[e'_i]_{P_i \in C}$ (where $e'_i = \log(e_i)$) can be derived.

Consider two C-neighboring inputs $X = (X_1, \dots, X_N)$, $X' = (X'_1, \dots, X'_N)$ leading to two sets of sketches $(S_1, \dots, S_N)$, $(S'_1, \dots, S'_N)$. Let

$$\begin{aligned}g(X) &:= \mathsf{Evaluate}(\mathsf{Merge}(S_1, \dots, S_N)) \\ g(X') &:= \mathsf{Evaluate}(\mathsf{Merge}(S'_1, \dots, S'_N)).\end{aligned}$$

Since $S(X_1, \dots, X_N) = \mathsf{Merge}(S_1, \dots, S_N)$ and $S(X'_1, \dots, X'_N) = \mathsf{Merge}(S'_1, \dots, S'_N)$, by definition of ΔS, we have that with all but negligible probability (over the randomness of the sketch)

$$|\log g(X) - \log g(X')| \le \Delta S. \qquad (3)$$

Let y be some outcome of the functionality $\mathcal{F}$. Note that $\log(y)$ is fully determined given y and vice versa. We therefore consider the probability of obtaining a given value of $\log(y)$, conditioned on fixed seed and $[e_i]_{i \in \overline{\mathcal{S}}}$, when the input is X versus X'. Let $\log'(y) = \log(y) - \sum_{i \notin \mathcal{S}} e'_i$.

If the input is X, then the probability of obtaining outcome $\log'(y)$ is the probability that $\sum_{i \in \mathcal{S}} e'_i = \log'(y) - \log g(X)$. Since $\sum_{i \in \mathcal{S}} e'_i$ is distributed as $\mathsf{Laplace}(0, \Delta S/\epsilon)$, this is equivalent to the probability of $\log'(y) - \log g(X)$ under the corresponding PDF:

$$\begin{aligned}p_X &:= Lap\left(\log'(y) - \log g(X) \middle| \frac{\Delta S}{\epsilon}\right) \\ &= \frac{\epsilon}{2\Delta S} \cdot e^{-\epsilon \cdot \frac{\log'(y) - \log g(X)}{\Delta S}}.\end{aligned}$$

Similarly, if the input is X', then the probability of obtaining outcome $\log'(y)$ is

$$\begin{aligned} p_{X'} &:= Lap\left(\log'(y) - \log g(X') \middle| \frac{\Delta S}{\epsilon}\right) \\ &= \frac{\epsilon}{2\Delta S} \cdot e^{-\epsilon \cdot \frac{\log'(y) - \log g(X')}{\Delta S}}. \end{aligned}$$

Using (3) we can upperbound the ratio $\frac{p_X}{p_{X'}}$ by

$$\begin{aligned} &exp\left(-\epsilon \cdot \frac{\log'(y) - \log g(X)}{\Delta S} + \epsilon \cdot \frac{\log'(y) + \log g(X')}{\Delta S}\right) \\ &= exp\left(\epsilon \cdot \frac{\log g(X) - \log g(X')}{\Delta S}\right) \\ &\le exp(\epsilon). \end{aligned}$$

Thus, we obtain the desired result that the ideal functionality achieves $(\epsilon, \delta = \mathsf{negl}(\lambda))$-differential privacy.

Protocol realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{Merge}}$-hybrid model. We now show that, once the MPC subprotocol in Figure 7 is replaced with the Ideal functionality $\mathcal{F}_{\mathsf{Merge}}$, we securely realize functionality $\mathcal{F}$, given in Figure 8.

We present a simulator Sim for the semi-honest case with less than $N/2$ corruptions. Let $\mathcal{S} \subseteq [N]$ be the set of uncorrupted parties and $\overline{\mathcal{S}} \subseteq [N]$ be the set of corrupted parties. For each of the corrupted parties, Sim receives their input stream X_i, $i \in \overline{\mathcal{S}}$. Sim invokes $\mathcal{F}$ with inputs $[X_i]_{i \in \overline{\mathcal{S}}}$, receiving back (y, seed, r_i). Sim sets the random tape of each corrupted party P_i to contain the randomness r_i returned by the ideal functionality, and chooses the rest of the tape uniformly at random. Sim uses the internal states of the corrupted parties to compute $\mathsf{seed}_i, i \in \overline{\mathcal{S}}$. Sim chooses random $\mathsf{seed}_i, i \in \mathcal{S}$ (i.e. the values for the uncorrupted parties) such that $\mathsf{seed} = \oplus_{i=1}^{n} \mathsf{seed}_i$. Sim instantiates the corrupted parties and plays the part of the uncorrupted parties in the protocol by sending $\mathsf{seed}_i, i \in \mathcal{S}$ to the corrupted parties. If Sim receives a random oracle query from a corrupted party, Sim forwards the query to the random oracle, and returns the response to the corrupted party.[1] When Sim receives $[(S_i, e_i)]_{i \in \overline{\mathcal{S}}}$ from the corrupted parties as input to $\mathcal{F}_{\mathsf{Merge}}$, Sim returns y to the corrupted parties. Finally, Sim outputs the view of the corrupted parties.

It is straightforward to verify that the joint distribution over the output of Sim and the output of the honest parties in the ideal $\mathcal{F}$-model is identical to the joint distribution over the view of the corrupted parties and the output of the honest parties in the $\mathcal{F}_{\mathsf{Merge}}$-hybrid model.

1 Note that Sim does not program the random oracle in our security proof.

Accuracy of the Protocol. To achieve $(\widetilde{\epsilon} + \overline{\epsilon} + \widetilde{\epsilon}\overline{\epsilon}, \widetilde{\delta} + \overline{\delta})$-accuracy, we need to ensure $(\overline{\epsilon}, \overline{\delta})$-accuracy of the outcome of the added noise. Therefore, letting $\alpha := \frac{GS(S)}{\epsilon}$, we must ensure that

$$\Pr\left[\left|\alpha \cdot \sum_{i \in [N]} e_i\right| > \log(1 + \overline{\epsilon})\right] \le \overline{\delta}.$$

Note the above is upperbounded by the following probability:

$$\Pr\left[\left|\alpha \sum_{i \in \mathcal{S}} e_i\right| > \frac{\log(1+\overline{\epsilon})}{2} \text{ or } \left|\alpha \sum_{i \in \overline{\mathcal{S}}} e_i\right| > \frac{\log(1+\overline{\epsilon})}{2}\right].$$

Therefore, it is sufficient to ensure that the probability that a Laplacian random variable $\mathsf{Laplace}(0, \frac{\Delta S}{\epsilon})$ has magnitude greater than $\log(1 + \overline{\epsilon})/2$ is bounded by $\overline{\delta}/2$. Using the CDF for the Laplacian, we thus require that

$$e^{-\frac{\log(1+\overline{\epsilon}) \cdot \epsilon}{2\Delta S}} \le \overline{\delta}/2 \tag{4}$$

Equivalently,

$$\frac{\log(1+\overline{\epsilon}) \cdot \epsilon}{2\Delta S} \ge \ln(2/\overline{\delta}). \tag{5}$$

We next bound the left side of (5):

$$\begin{aligned} &\frac{\log(1+\overline{\epsilon}) \cdot \epsilon}{2\log(1 + 2\widetilde{\epsilon}/(1-\widetilde{\epsilon})) + 2\log(1 + \Delta S/\min(f))} \\ &= \frac{\ln(1+\overline{\epsilon}) \cdot \epsilon}{2\ln(1 + 2\widetilde{\epsilon}/(1-\widetilde{\epsilon})) + 2\ln(1 + \Delta S/\min(f))} \\ &\ge \frac{(\overline{\epsilon} - \overline{\epsilon}^2/2) \cdot \epsilon}{4\widetilde{\epsilon}/(1-\widetilde{\epsilon}) + 2\Delta S/\min(f)}. \end{aligned}$$

We used two inequalities, i.e., $\ln(1+x) \ge x - x^2/2$ and $(1+x) \le e^x$ in the above. Thus, setting

$$\widetilde{\epsilon}/(1-\widetilde{\epsilon}) \le \frac{(\overline{\epsilon} - \overline{\epsilon}^2/2) \cdot \epsilon}{4\ln(2/\overline{\delta})} - \frac{\Delta S}{2\min(f)}, \tag{6}$$

we obtain the desired accuracy.