Differential Privacy on Finite Computers*†

Victor Balcer[1] and Salil Vadhan[2]

1 **Center for Research on Computation & Society, School of Engineering & Applied Sciences, Harvard University, 33 Oxford Street, Cambridge, MA 02138, USA**
vbalcer@g.harvard.edu

2 **Center for Research on Computation & Society, School of Engineering & Applied Sciences, Harvard University, 33 Oxford Street, Cambridge, MA 02138, USA**
salil_vadhan@harvard.edu

Abstract

We consider the problem of designing and analyzing differentially private algorithms that can be implemented on *discrete* models of computation in *strict* polynomial time, motivated by known attacks on floating point implementations of real-arithmetic differentially private algorithms (Mironov, CCS 2012) and the potential for timing attacks on expected polynomial-time algorithms. We use a case study: the basic problem of approximating the histogram of a categorical dataset over a possibly large data universe $\mathcal{X}$. The classic Laplace Mechanism (Dwork, McSherry, Nissim, Smith, TCC 2006 and J. Privacy & Confidentiality 2017) does not satisfy our requirements, as it is based on real arithmetic, and natural discrete analogues, such as the Geometric Mechanism (Ghosh, Roughgarden, Sundarajan, STOC 2009 and SICOMP 2012), take time at least linear in $|\mathcal{X}|$, which can be exponential in the bit length of the input.

In this paper, we provide strict polynomial-time discrete algorithms for approximate histograms whose simultaneous accuracy (the maximum error over all bins) matches that of the Laplace Mechanism up to constant factors, while retaining the same (pure) differential privacy guarantee. One of our algorithms produces a sparse histogram as output. Its "per-bin accuracy" (the error on individual bins) is worse than that of the Laplace Mechanism by a factor of $\log |\mathcal{X}|$, but we prove a lower bound showing that this is necessary for any algorithm that produces a sparse histogram. A second algorithm avoids this lower bound, and matches the per-bin accuracy of the Laplace Mechanism, by producing a compact and efficiently computable representation of a dense histogram; it is based on an $(n+1)$-wise independent implementation of an appropriately clamped version of the Discrete Geometric Mechanism.

1998 ACM Subject Classification F.2.m Analysis of Algorithms and Problem Complexity, Miscellaneous

Keywords and phrases Algorithms, Differential Privacy, Discrete Computation, Histograms

Digital Object Identifier 10.4230/LIPIcs.ITCS.2018.43

* This work was supported by NSF grant CNS-1237235 and CNS-1565387, a Simons Investigator Award, and a grant from the Sloan Foundation.

† A full version of the paper is available at [1], http://arxiv.org/abs/1709.05396.

1 Introduction

Differential Privacy [11] is by now a well-established framework for privacy-protective statistical analysis of sensitive datasets. Much work on differential privacy involves an interplay between statistics and computer science. Statistics provides many of the (non-private) analyses that we wish to approximate with differentially private algorithms, as well as probabilistic tools that are useful in analyzing such algorithms, which are necessarily randomized. From computer science, differential privacy draws upon a tradition of adversarial modeling and strong security definitions, techniques for designing and analyzing randomized algorithms, and considerations of algorithmic resource constraints (such as time and memory).

Because of its connection to statistics, it is very natural that much of the literature on differential privacy considers the estimation of real-valued functions on real-valued data (e.g. the sample mean) and introduces noise from continuous probability distributions (e.g. the Laplace distribution) to obtain privacy. However, these choices are incompatible with standard computer science models for algorithms (like the Turing machine or RAM model) as well as implementation on physical computers (which use only finite approximations to real arithmetic, e.g. via floating point numbers). This discrepancy is not just a theoretical concern; Mironov [21] strikingly demonstrated that common floating-point implementations of the most basic differentially private algorithm (the Laplace Mechanism) are vulnerable to real attacks. Mironov shows how to prevent his attack with a simple modification to the implementation, but this solution is specific to a single differentially private mechanism and particular floating-point arithmetic standard. His solution increases the error by a constant factor and is most likely more efficient in practice than the algorithm we will use to replace the Laplace Mechanism. However, he provides no bounds on asymptotic running time. Gazeau, Miller and Palamidessi [13] provide more general conditions for which an implementation of real numbers and a mechanism that perturbs the correct answer with noise maintains differential privacy. However, they do not provide an explicit construction with bounds on accuracy and running time.

From a theoretical point of view, a more appealing approach to resolving these issues is to avoid real or floating-point arithmetic entirely and only consider differentially private computations that involve discrete inputs and outputs, and rational probabilities, as first done in [10]. Such algorithms are realizable in standard discrete models of computation. However, some such algorithms have running times that are only bounded in expectation (e.g. due to sampling from an exponential distribution supported on the natural numbers), and this raises a potential vulnerability to timing attacks. If an adversary can observe the running time of the algorithm, it learns something about the algorithm's coin tosses, which are assumed to be secret in the definition of differential privacy. (Even if the time cannot be directly observed, in practice an adversary can determine an upper bound on the running time, which again is information that is implicitly assumed to be secret in the privacy definition.)

Because of these considerations, we advocate the following principle:

> ***Differential Privacy for Finite Computers:***
> *We should describe how to implement differentially private algorithms on* ***discrete*** *models of computation with* ***strict*** *bounds on running time (ideally polynomial in the bit length of the input) and* ***analyze*** *the effects of those constraints on both privacy and accuracy.*

Note that a strict *bound* on running time does not in itself prevent timing attacks, but once we have such a bound, we can pad all executions to take the same amount of time.

Also, while standard discrete models of computation (e.g. randomized Turing machines) are defined in terms of countable rather than finite resources (e.g. the infinite tape), if we have a strict bound on running time, then once we fix an upper bound on input length, they can indeed be implemented on a truly finite computer (e.g. like a randomized Boolean circuit).

In many cases, the above goal can be achieved by appropriate discretizations and truncations applied to a standard, real-arithmetic differentially private algorithm. However, such modifications can have a nontrivial price in accuracy or privacy, and thus we also call for a rigorous analysis of these effects.

In this paper, we carry out a case study of achieving "differential privacy for finite computers" for one of the first tasks studied in differential privacy, namely approximating a histogram of a categorical dataset. Even this basic problem turns out to require some nontrivial effort, particularly to maintain strict polynomial time, optimal accuracy and pure differential privacy when the data universe is large.

We recall the definition of differential privacy.

▶ **Definition 1.1** ([11])**.** Let $\mathcal{M} : \mathcal{X}^n \to \mathcal{R}$ be a randomized algorithm. We say $\mathcal{M}$ is **(ε, δ)-differentially private** if for every pair of neighboring datasets D and D' (datasets differing on one row) and every subset $S \subseteq \mathcal{R}$

$$\Pr[\mathcal{M}(D) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathcal{M}(D') \in S] + \delta$$

We say an (ε, δ)-differentially private algorithm satisfies **pure differential privacy** when $\delta = 0$ and say it satisfies **approximate differential privacy** when $\delta > 0$.

In this paper, we study the problem of estimating the *histogram* of a dataset $D \in \mathcal{X}^n$, which is the vector $c = c(D) \in \mathbb{N}^{\mathcal{X}}$, where c_x is the number of rows in D that have value x. Histograms can be approximated while satisfying differential privacy using the *Laplace Mechanism*, introduced in the original paper of Dwork, McSherry, Nissim and Smith [11]. Specifically, to obtain $(\varepsilon, 0)$-differential privacy, we can add independent noise distributed according to a Laplace distribution, specifically $\mathrm{Lap}(2/\varepsilon)$, to each component of c and output the resulting vector $\tilde{c}$. Here $\mathrm{Lap}(2/\varepsilon)$ is the *continuous*, real-valued random variable with probability density function $f(z)$ that is proportional to $\exp(-\varepsilon \cdot |z|/2)$. The Laplace Mechanism also achieves very high accuracy in two respects:

Per-Query Error: For each bin $x \in \mathcal{X}$, with high probability we have $|\tilde{c}_x - c_x| \leq O(1/\varepsilon)$.
Simultaneous Error: With high probability, we have $\max_x |\tilde{c}_x - c_x| \leq O(\log(|\mathcal{X}|)/\varepsilon)$.

Note that both of the bounds are independent of the number n of rows in the dataset, and so the fractional error vanishes linearly as n grows.

Simultaneous error is the more well-studied notion in the differential privacy literature, but we consider per-query error to be an equally natural concept: if we think of the approximate histogram $\tilde{c}$ as containing approximate answers to the $|\mathcal{X}|$ different counting queries corresponding to the bins of $\mathcal{X}$, then per-query error captures the error as experienced by an analyst who may be only interested in one or a few of the bins of $\tilde{c}$. The advantage of considering per-query error is that it can be significantly smaller than the simultaneous error, as is the case in the Laplace Mechanism when the data universe $\mathcal{X}$ is very large. It is known that both of the error bounds achieved by the Laplace Mechanism are optimal up to constant factors; no $(\varepsilon, 0)$-differentially private algorithm for histograms can achieve smaller per-query error or simultaneous error [17, 2].

Unfortunately, the Laplace Mechanism uses real arithmetic and thus cannot be implemented on a finite computer. To avoid real arithmetic, we could use the Geometric Mechanism

[14], which adds noise to each component of c according to the 2-sided geometric distribution, Geo($2/\varepsilon$), which is supported on the integers and has probability mass function $f(z) \propto \exp(-\varepsilon \cdot |z|/2)$. However, this mechanism uses integers of unbounded size and thus cannot be implemented on a finite computer. Indeed, while the algorithm can be implemented with a running time that is bounded in expectation (after reducing ε so that $e^{\varepsilon/2}$ and hence all the probabilities are rational numbers), truncating long executions or allowing an adversary to observe the actual running time can lead to a violation of differential privacy. Thus, as first described by Dwork, Kenthapadi, McSherry, Mironov and Naor [10], it is better to restrict the output of the mechanism to a binary representation of fixed length in order to avoid small tail probabilities. Similarly, we work with the Truncated Geometric Mechanism of Ghosh, Roughgarden and Sundararajan [14], where we clamp each noisy count $\tilde{c}_x$ to the interval $[0, n]$. We observe that the resulting probability distribution of $\tilde{c}_x$, supported on $\{0, 1, \ldots, n\}$, can be described explicitly in terms of c_x, ε and n, and it can be sampled in polynomial time using only integer arithmetic (after ensuring $e^{\varepsilon/2}$ is rational). Thus, we obtain:

▶ **Theorem 1.2** (Bounded Geometric Mechanism, informal statement of Thm. 5.1)**.** *For every finite $\mathcal{X}$, n and $\varepsilon \in (0, 1]$, there is an $(\varepsilon, 0)$-differentially private algorithm $\mathcal{M} : \mathcal{X}^n \to \{0, 1, \ldots, n\}^{\mathcal{X}}$ for histograms achieving:*

- *Per-query error $O(1/\varepsilon)$.*
- *Simultaneous error $O(\log(|\mathcal{X}|)/\varepsilon)$.*
- *Strict running time $\tilde{O}\left((|\mathcal{X}|/\varepsilon) \cdot \log^2 n\right) + O(n \log n \cdot \log |\mathcal{X}|)$*

We note that while we only consider our particular definition of per-query accuracy, namely that with high probability $|\tilde{c}_x - c_x| \leq O(1/\varepsilon)$, Ghosh et al. [14] proved that the output of the Bounded Geometric Mechanism can be used (with post-processing) to get optimal expected loss with respect to an extremely general class of loss functions and arbitrary priors. The same result applies to each individual noisy count $\tilde{c}_x$ output by our mechanism, since each bin is distributed according to the Bounded Geometric Mechanism (up to a modification of ε to ensure rational probabilities).

The Bounded Geometric Mechanism is not polynomial time for large data universes $\mathcal{X}$. Indeed, its running time (and output length) is linear in $|\mathcal{X}|$, rather than polynomial in the bit length of data elements, which is $\log |\mathcal{X}|$. To achieve truly polynomial time, we can similarly discretize and truncate a variant of the Stability-Based Histogram that was introduced by Korolova, Kenthapadi, Mishra and Ntoulas [19], and explicitly described by Bun, Nissim and Stemmer [4]. This mechanism only adds Lap($2/\varepsilon$) noise to the *nonzero* components of c_x and then retains only the noisy values $\tilde{c}_x$ that are larger than a threshold $t = \Theta(\log(1/\delta)/\varepsilon)$. Thus, the algorithm only outputs a partial histogram, i.e. counts $\tilde{c}_x$ for a subset of the bins x, with the rest of the counts being treated as zero. By replacing the use of the Laplace Mechanism with the (rational) Bounded Geometric Mechanism as above, we can implement this algorithm in strict polynomial time:

▶ **Theorem 1.3** (Stability-Based Histogram, formal statement omitted from this version)**.** *For every finite $\mathcal{X}$, n, $\varepsilon \in (0, 1]$ and $\delta \in (0, 1/n)$, there is an (ε, δ)-differentially private algorithm $\mathcal{M} : \mathcal{X}^n \to \{0, 1, \ldots, n\}^{\subseteq \mathcal{X}}$ for histograms achieving:*

- *Per-query error $O(1/\varepsilon)$ on bins with true count at least $O(\log(1/\delta)/\varepsilon)$.*
- *Simultaneous error $O(\log(1/\delta)/\varepsilon)$.*
- *Strict running time $\tilde{O}((n/\varepsilon) \cdot \log(1/\delta)) + O(n \log n \cdot \log |\mathcal{X}|)$.*

Notice that the simultaneous error bound of $O(\log(1/\delta)/\varepsilon)$ is better than what is achieved by the Laplace Mechanism when $\delta > 1/|\mathcal{X}|$, and is known to be optimal up to constant

factors in this range of parameters. The fact that this error bound is independent of the data universe size $|\mathcal{X}|$ makes it tempting to apply even for infinite data domains $\mathcal{X}$. However, we note that when $\mathcal{X}$ is infinite, it is impossible for the algorithm to have a strict bound on running time (as it needs time to read arbitrarily long data elements) and thus is vulnerable to timing attacks and is not implementable on a finite computer.

Note also that the per-query error bound only holds on bins with large enough true count (namely, those larger than our threshold t); we will discuss this point further below.

A disadvantage of the Stability-based Histogram is that it sacrifices pure differential privacy. It is natural to ask whether we can achieve polynomial running time while retaining pure differential privacy. A step in this direction was made by Cormode, Procopiuc, Srivastava and Tran [9]. They observe that for an appropriate threshold $t = \Theta(\log(|\mathcal{X}|)/\varepsilon)$, if we run the Bounded Geometric Mechanism and only retain the noisy counts $\tilde{c}_x$ that are larger than t, then the expected number of bins that remain is less than $n + 1$. Indeed, the expected number of bins we retain whose true count is zero ("empty bins") is less than 1. They describe a method to directly sample the distribution of the empty bins that are retained, without actually adding noise to all $|\mathcal{X}|$ bins. This yields an algorithm whose output length is polynomial in expectation. However, the output length is not strictly polynomial, as there is a nonzero probability of outputting all $|\mathcal{X}|$ bins. And it is not clear how to implement the algorithm even in expected polynomial time, because even after making the probabilities rational, they have denominators of bit length linear in $|\mathcal{X}|$.

To address these issues, we consider a slightly different algorithm. Instead of trying to retain all noisy counts $\tilde{c}_x$ that are larger than some fixed threshold t, we retain the n largest noisy counts (since there are at most n nonzero true counts). This results in a mechanism whose output length is always polynomial, rather than only in expectation. However, the probabilities still have denominators of bit length linear in $|\mathcal{X}|$. Thus, we show how to approximately sample from this distribution, to within an arbitrarily small statistical distance δ, at the price of a $\mathrm{poly}(\log(1/\delta))$ increase in running time. Naively, this would result only in $(\varepsilon, O(\delta))$-differential privacy. However, when δ is significantly smaller than $1/|\mathcal{R}|$, where $\mathcal{R}$ is the range of the mechanism, we can convert an (ε, δ)-differentially private mechanism to an $(\varepsilon, 0)$-differentially private mechanism by simply outputting a uniformly random element of $\mathcal{R}$ with small probability. (A similar idea for the case that $|\mathcal{R}| = 2$ has been used in [18, 5].) Since our range is of at most exponential size (indeed at most polynomial in bit length), the cost in our runtime for taking $\delta \ll 1/|\mathcal{R}|$ is at most polynomial. With these ideas we obtain:

▶ **Theorem 1.4** (Pure DP Histogram in Polynomial Time, informal statement of Thm. 6.6)**.** *For every finite $\mathcal{X}$, n and $\varepsilon \in (0, 1]$, there is an $(\varepsilon, 0)$-differentially private algorithm $\mathcal{M} : \mathcal{X}^n \to \{0, 1, \ldots, n\}^{\subseteq \mathcal{X}}$ for histograms achieving:*

- *Per-query error $O(1/\varepsilon)$ on bins with true count at least $O(\log(|\mathcal{X}|)/\varepsilon)$.*
- *Simultaneous error $O(\log(|\mathcal{X}|)/\varepsilon)$.*
- *Strict running time $\tilde{O}(n^2 \cdot \log^2 |\mathcal{X}| + (n/\varepsilon) \cdot \log^2 |\mathcal{X}|)$.*

It is an open problem as to whether or not we can improve the nearly quadratic dependence in running time on n to nearly linear while maintaining the sparsity, privacy and accuracy guarantees achieved in Theorem 1.4.

Both Theorems 1.3 and 1.4 only retain per-query error $O(1/\varepsilon)$ on bins with a large enough true count. In the full version of the paper [1], we also prove a lower bound showing that this limitation is inherent in any algorithm that outputs a sparse histogram (as both of these algorithms do).

▶ **Theorem 1.5** (Lower Bound on Per-Query Error for Sparse Histograms)**.** *Suppose that there is an (ε, δ)-differentially private algorithm $\mathcal{M} : \mathcal{X}^n \to \{0, 1, \ldots, n\}^{\mathcal{X}}$ for histograms that always outputs histograms with at most n' nonempty bins and has per-query error at most E on all bins. Then*

$$E \geq \Omega\left(\frac{\min\{\log|\mathcal{X}|, \log(1/\delta)\}}{\varepsilon}\right)$$

provided that $\varepsilon > 0$, $\varepsilon^2 > \delta > 0$ and $|\mathcal{X}| \geq (n')^2$.

This lower bound is similar in spirit to a lower bound of [2], which shows that no $(\varepsilon, 0)$-differentially private PAC learner for "point functions" (functions that are 1 on exactly one element of the domain) can produce sparse functions as hypotheses.

To bypass this lower bound, we can consider algorithms that produce succinct descriptions of dense histograms. That is, the algorithm can output a polynomial-length description of a function $\tilde{c} : \mathcal{X} \to [0, n]$ that can be evaluated in polynomial time, even though $\mathcal{X}$ may be of exponential size.

We show that this relaxation allows us to regain per-query error $O(1/\varepsilon)$.

▶ **Theorem 1.6** (Polynomial-Time DP Histograms with Optimal Per-Query Accuracy, informal statement of Thm. 7.2)**.** *For every finite $\mathcal{X}$, n and $\varepsilon \in (0, 1]$, there is an $(\varepsilon, 0)$-differentially private algorithm $\mathcal{M} : \mathcal{X}^n \to \mathcal{H}$ for histograms (where $\mathcal{H}$ is an appropriate class of succinct descriptions of histograms) achieving:*

- *Per-query error $O(1/\varepsilon)$.*
- *Simultaneous error $O(\log(|\mathcal{X}|)/\varepsilon)$.*
- *Strict running time $O(n) \cdot \tilde{O}((1/\varepsilon^2) \cdot (\log^2 n + \log^2 |\mathcal{X}|))$*
- *Evaluation time $O(n) \cdot \tilde{O}((1/\varepsilon) \cdot (\log n + \log |\mathcal{X}|))$.*

The algorithm is essentially an $(n + 1)$-wise independent instantiation of the Bounded Geometric Mechanism. Specifically, we release a function $h : \mathcal{X} \to \{0, 1\}^r$ selected from an $(n + 1)$-wise independent family of hash functions, and for each $x \in \mathcal{X}$, we view $h(x)$ as coin tosses specifying a sample from the Bounded Geometric Distribution. That is, we let $S : \{0, 1\}^r \to [0, n]$ be an efficient sampling algorithm for the Bounded Geometric Distribution, and then $\tilde{c}_x = S(h(x))$ is our noisy count for x. The hash function is chosen randomly from the family conditioned on values $\tilde{c}_x$ for the nonempty bins x, which we obtain by running the actual Bounded Geometric Mechanism on those bins. The $(n + 1)$-wise independence ensures that the behavior on any two neighboring datasets (which together involve at most $n + 1$ distinct elements of $\mathcal{X}$) are indistinguishable in the same way as in the ordinary Bounded Geometric Mechanism. The per-query accuracy comes from the fact that the marginal distributions of each of the noisy counts are the same as in the Bounded Geometric Mechanism. (Actually, we incur a small approximation error in matching the domain of the sampling procedure to the range of a family of hash functions.)

As far as we know, the only other use of limited independence in constructing differentially private algorithms is a use of pairwise independence by [2] in differentially private PAC learning algorithms for the class of point functions. Although that problem is related to the one we consider (releasing a histogram amounts to doing "query release" for the class of point functions, as discussed below), the design and analysis of our algorithm appears quite different. (In particular, our analysis seems to rely on $(n + 1)$-wise independence in an essential way.)

Another potential interest in our technique is as another method for bypassing limitations of *synthetic data* for *query release*. Here, we have a large family of predicates $\mathcal{Q} = \{q : \mathcal{X} \to \{0,1\}\}$, and are interested in differentially private algorithms that, given a dataset $D = (x_1, \ldots, x_n) \in \mathcal{X}^n$, output a "summary" $\mathcal{M}(D)$ that allows one to approximate the answers to all of the *counting queries* $q(D) = \sum_i q(x_i)$ associated with predicates $q \in \mathcal{Q}$. For example, if $\mathcal{Q}$ is the family of *point functions* consisting of all predicates that evaluate to 1 on exactly one point in the data universe $\mathcal{X}$, then this query release problem amounts to approximating the histogram of D. The fundamental result of Blum, Ligett, and Roth [3] and successors show that this is possible even for families $\mathcal{Q}$ and data universes $\mathcal{X}$ that are of size exponential in n. Moreover, the summaries produced by these algorithms has the form of a synthetic dataset — a dataset $\hat{D} \in \mathcal{X}^{\hat{n}}$ such that for every query $q \in \mathcal{Q}$, we have $q(\hat{D}) \approx q(D)$. Unfortunately, it was shown in [24] that even for very simple families $\mathcal{Q}$ of queries, such correlations between pairs of binary attributes, constructing such a differentially private synthetic dataset requires time exponential in the bit length $\log |\mathcal{X}|$ of data universe elements. Thus, it is important to find other ways of representing approximate answers to natural families $\mathcal{Q}$ of counting queries, which can bypass the inherent limitations of synthetic data, and progress along these lines was made in a variety of works [15, 7, 16, 23, 6, 12]. Our algorithm, and its use of $(n+1)$-wise independence, can be seen as yet another representation that bypasses a limitation of synthetic data (albeit a statistical rather than computational one). Indeed, a sparse histogram is simply a synthetic dataset that approximates answers to all point functions, and by Theorem 1.5, our algorithm achieves provably better per-query accuracy than is possible with synthetic datasets. This raises the question of whether similar ideas can also be useful in bypassing the computational limitations of synthetic data for more complex families of counting queries.

2 Preliminaries

Throughout this paper, let $\mathbb{N}$ be the set $\{0, 1, \ldots\}$, $\mathbb{N}_+$ be the set $\{1, 2, \ldots\}$ and $\mathbb{N}^{-1}$ be the set $\{1/n : n \in \mathbb{N}_+\}$. For $n \in \mathbb{N}_+$, let $[n]$ denote the set $\{0, \ldots, n\}$ and $[n]_+$ denote the set $\{1, \ldots, n\}$. (Notice that $|[n]| = n + 1$ while $|[n]_+| = n$.) Given a set A and finite set B, we define A^B to be the set of length $|B|$ vectors over A indexed by the elements of B.

2.1 Histograms

For $x \in \mathcal{X}$, the **point function** $c_x : \mathcal{X}^n \to \mathbb{N}$ is defined to count the number of occurrences of x in a given dataset, i.e. $c_x(D) = |\{i \in [n]_+ : D_i = x\}|$. In this paper we focus on algorithms for privately releasing approximations to the values of all point functions, also known as a **histogram**. A histogram is a collection of **bins**, one for each element x in the data universe, with the x^{th} bin consisting of its **label** x and a **count** $c_x \in \mathbb{N}$.

2.1.1 Representations

The input to our algorithms is always a dataset (i.e. an element $D \in \mathcal{X}^n$) and the outputs represent approximate histograms. We consider the following histogram representations as our algorithms' outputs:

- A vector in $\mathbb{N}^{\mathcal{X}}$. We use $\{\tilde{c}_x\}_{x \in \mathcal{X}}$ to denote a histogram where $\tilde{c}_x \in \mathbb{N}$ is the approximate count for the element x.
- A partial vector $h \in (\mathcal{X} \times \mathbb{N})^*$ such that each element $x \in \mathcal{X}$ appears at most once in h with each pair $(x, \tilde{c}_x) \in \mathcal{X} \times \mathbb{N}$ interpreted as element x having approximate count $\tilde{c}_x$.

Elements x not listed in the partial vector are assumed to have count $\tilde{c}_x = 0$. Implicitly, an algorithm can return a partial vector by releasing bins for a subset of $\mathcal{X}$.[1]

- A data structure, encoded as a string, which defines a function $h : \mathcal{X} \to \mathbb{N}$ where $h(x)$, denoted h_x, is the approximate count for $x \in \mathcal{X}$ and h_x is efficiently computable given this data structure (e.g. time polynomial in the length of the data structure). In Section 7, this data structure consists of the coefficients of a polynomial, along with some parameters.

Each representation is able to express any histogram over $\mathcal{X}$. The difference between them is the memory used and the efficiency of computing a count. For example, computing the approximate count for $x \in \mathcal{X}$, when using the data structure representation is bounded by the time it takes to compute the associated function. But when using partial vectors, one only needs to iterate through the vector to determine the approximate count.

We define the following class of histograms. Let $\mathcal{H}_{n,n'}(\mathcal{X}) \subseteq \mathbb{N}^{\mathcal{X}}$ be the set of all histograms over $\mathcal{X}$ with integer counts in $[0, n]$ (or $\mathbb{N}$ when $n = \infty$) and at most n' of them nonzero. By using partial vectors each element of $\mathcal{H}_{n,n'}(\mathcal{X})$ can be stored in $O(n' \cdot (\log n + \log |\mathcal{X}|))$ bits, which is shorter than the vector representation when $n' = o(|\mathcal{X}|/\log |\mathcal{X}|)$.

2.1.2 Accuracy

In order to preserve privacy, our algorithms return histograms with noise added to the counts. Therefore, it is crucial to understand their accuracy guarantees. So given a dataset $D \in \mathcal{X}^n$ we compare the **noisy count** $\tilde{c}_x = \mathcal{M}(D)_x$ of $x \in \mathcal{X}$ (the count released by algorithm $\mathcal{M}$) to its **true count**, $c_x(D)$. We focus on the following two metrics:

▶ **Definition 2.1.** A histogram algorithm $\mathcal{M} : \mathcal{X}^n \to \mathbb{N}^{\mathcal{X}}$ has **(a, β)-per-query accuracy** if

$$\forall D \in \mathcal{X}^n \quad \forall x \in \mathcal{X} \qquad \Pr[|\mathcal{M}(D)_x - c_x(D)| \leq a] \geq 1 - \beta$$

▶ **Definition 2.2.** A histogram algorithm $\mathcal{M} : \mathcal{X}^n \to \mathbb{N}^{\mathcal{X}}$ has **(a, β)-simultaneous accuracy** if

$$\forall D \in \mathcal{X}^n \qquad \Pr[\forall x \in \mathcal{X} \quad |\mathcal{M}(D)_x - c_x(D)| \leq a] \geq 1 - \beta$$

Respectively, these metrics capture the maximum error for any one bin and the maximum error simultaneously over all bins. However, we may not always be able to achieve as good per-query accuracy as we want. So we will also use the following relaxation which bounds the error only on bins with large enough true count.

▶ **Definition 2.3.** A histogram algorithm $\mathcal{M} : \mathcal{X}^n \to \mathbb{N}^{\mathcal{X}}$ has **(a, β)-per-query accuracy on counts larger than t** if

$$\forall D \in \mathcal{X}^n \quad \forall x \in \mathcal{X} \text{ s.t. } c_x(D) > t \qquad \Pr[|\mathcal{M}(D)_x - c_x(D)| \leq a] \geq 1 - \beta$$

2.2 Probability Terminology

▶ **Definition 2.4.** Let Z be an integer-valued random variable. The **probability mass function of Z**, denoted f_Z, is the function $f_Z(z) = \Pr[Z = z]$ for all $z \in \mathbb{Z}$. The **cumulative distribution function of Z**, denoted F_Z, is the function $F_Z(z) = \Pr[Z \leq z]$ for all $z \in \mathbb{Z}$. The **support** of Z, denoted $\text{supp}(Z)$, is the set of elements for which $f(z) \neq 0$.

[1] Note that the order in which bins are released can result in a breach of privacy (e.g. releasing the bins of elements in the dataset before the bins of elements not in the dataset). As a result, our algorithms always sort the released bins according to a predefined ordering based only on $\mathcal{X}$.

▶ **Definition 2.5.** Let Y and Z be random variables taking values in discrete range $\mathcal{R}$. The **total variation distance between Y and Z** is defined as

$$\Delta(Y,Z) = \max_{A \subseteq \mathcal{R}} \left| \Pr[Y \in A] - \Pr[Z \in A] \right| = \frac{1}{2} \cdot \sum_{a \in \mathcal{R}} \left| \Pr[Z = a] - \Pr[Y = a] \right|$$

2.2.1 Sampling

Because we are interested in the computational efficiency of our algorithms we need to consider the efficiency of sampling from various distributions.

A standard method for sampling a random variable is via **inverse transform sampling**. Let $\mathrm{Unif}(A)$ denote the uniform distribution over the set A.

▶ **Lemma 2.6.** *Let $U \sim \mathrm{Unif}((0,1])$. Then for any integer-valued random variable Z we have $F_Z^{-1}(U) \sim Z$ where $F_Z^{-1}(u)$ is defined as $\min\{z \in \mathrm{supp}(Z) : F_Z(z) \geq u\}$.*

If Z, the random variable we wish to sample, has finite support we can compute the inverse cumulative distribution by performing binary search on $\mathrm{supp}(Z)$ to find the minimum. This method removes the need to compute the inverse function of the cumulative distribution function. If in addition, the cumulative distribution function of Z can be represented by rational numbers, then we only need to sample from a discrete distribution instead of $(0,1]$.

▶ **Lemma 2.7.** *Let Z be an integer-valued random variable with finite support and has all probabilities of its cumulative distribution function expressible as rational numbers with denominator d. Then $F_Z^{-1}(U) \sim Z$ where $U \sim (1/d) \cdot \mathrm{Unif}([d]_+)$ and $F_Z^{-1}(u)$ is defined as $\min\{z \in \mathrm{supp}(Z) : F_Z(z) \geq u\}$.*

2.2.2 Order Statistics

▶ **Definition 2.8.** Let $Z_1, \ldots, Z_\ell$ be integer-valued random variables. The **i-th order statistic of $Z_1, \ldots, Z_\ell$** denoted $Z_{(i)}$ is the i-th smallest value among $Z_1, \ldots, Z_\ell$.

▶ **Lemma 2.9.** *Let $Z_1, \ldots, Z_\ell$ be i.i.d. integer-valued random variables with cumulative distribution function F. Then $F_{Z_{(\ell)}}(z) = (F(z))^\ell$ and*

$$F_{Z_{(i)} | Z_{(i+1)} = v_{i+1}, \ldots Z_{(\ell)} = v_\ell}(z) = F_{Z_{(i)} | Z_{(i+1)} = v_{i+1}}(z) = \begin{cases} 1 & \text{if } z > v_{i+1} \\ \left(F(z)/F(v_{i+1})\right)^i & \text{otherwise} \end{cases}$$

for all $1 \leq i < \ell$ and $v_{i+1} \leq v_{i+2} \leq \ldots \leq v_\ell$ all in the support of Z_1.

From this lemma, we can iteratively sample random variables distributed identically to $Z_{(\ell)}, Z_{(\ell-1)}, \ldots, Z_{(i)}$ without having to sample all ℓ of the original random variables.

2.3 Model of Computation

We analyze the running time of our algorithms with respect to the **w-bit word RAM model** taking w logarithmic in our input length, namely $w = O(\log n + \log\log|\mathcal{X}|)$. In this model, memory accesses and basic operations (arithmetic, comparisons and logical) on w-bit words are constant time. In addition, we assume the data universe $\mathcal{X} = [m]$ for some $m \in \mathbb{N}$. Some parameters to our algorithms are rational. We represent rationals by pairs of integers.

Some of our algorithms will use numbers that span many words. And one of our algorithms operates on finite fields $\mathbb{F}_d$ where $d = 2^{2 \cdot 3^\ell}$ for some $\ell \in \mathbb{N}$. We will use the fact

that $\mathbb{F}_d \simeq \mathbb{F}_2[x]/(x^{2\cdot 3^\ell} + x^{3^\ell} + 1)$ [20] to provide an explicit representation of $\mathbb{F}_d$. For ease of notation, we make the following assumptions on running time: (i) multiplying two x-bit numbers is $\tilde{O}(x)$ [25], (ii) multiplying two elements of $\mathbb{F}_d$ is $\tilde{O}(\log d)$ [22], (iii) multiplying two degree q polynomials over $\mathbb{F}_d$ is $\tilde{O}(q \cdot \log d)$ [22], (iv) evaluating a degree q polynomial over $\mathbb{F}_d$ is $O(q) \cdot \tilde{O}(\log d)$ [25] and (v) polynomial interpolation of q distinct points over $\mathbb{F}_d$ is $\tilde{O}(q \cdot \log^2 d)$ [25].

Our algorithms require randomness so we assume that they have access to an oracle that when given a number $d \in \mathbb{N}_+$ returns a uniformly random integer between 1 and d inclusive.

Finally, for representing histograms as partial vectors, we will assume internally to the algorithms that they are stored as red-black trees. This will allow us to insert and search for elements in $O(\log n \cdot \log |\mathcal{X}|)$ time [8]. However, when releasing a partial vector we return a list of bins using an in-order traversal of tree as the tree's structure could violate privacy.

3 A General Framework for Implementing Differential Privacy

In this section, we outline a basic framework for implementing a pure differentially private algorithm $\mathcal{M}$ on a finite computer with only a small loss in privacy and possibly a small loss in accuracy. It can be broken down into the following steps:

1. Start by discretizing the input and output of $\mathcal{M}$ so that they can only take on a finite number of values (e.g. rounding a real-valued number to the nearest integer in some finite set). Depending on how utility is measured, the loss in accuracy by discretizing may be acceptable.
2. Then find an algorithm $\mathcal{M}'$ that runs on a finite computer and approximates the output distribution of the discretized version of $\mathcal{M}$ to within "small" statistical distance. Notice that $\mathcal{M}'$ is only guaranteed to satisfy approximate differential privacy and may not satisfy pure differentially privacy. (This step may requires a non-trivial amount of work. For one example, see Theorem 6.5.)
3. Finally, provided that the statistical distance of the previous step is small enough, by mixing $\mathcal{M}'$ with uniformly random output (from the discretized and finite output space), the resulting algorithm satisfies pure differentially privacy.

We will use this framework several times in designing our algorithms. Here we start by formalizing Step 3. That is, for algorithms whose output distribution is close in total variation distance to that of a pure differentially private algorithm, we construct an algorithm satisfying pure differentially privacy by mixing it with random output inspired by similar techniques in [18, 5].

Algorithm 1 $\mathcal{M}^*_{\mathcal{M}',\mathcal{D},\gamma}(D)$ for $D \in \mathcal{X}^n$ where $\mathcal{R}$ is discrete and finite, an algorithm $\mathcal{M}' : \mathcal{X}^n \to \mathcal{R}$, a distribution $\mathcal{D}$ over $\mathcal{R}$ and $\gamma \in \mathbb{N}^{-1}$

1. With probability $1 - \gamma$ release $\mathcal{M}'(D)$.
2. Otherwise release an element sampled from the distribution $\mathcal{D}$.

▶ **Lemma 3.1.** *Suppose that there is an $(\varepsilon, 0)$-differentially private algorithm $\mathcal{M} : \mathcal{X}^n \to \mathcal{R}$ such that $\Delta(\mathcal{M}(D), \mathcal{M}'(D)) \leq \delta$ for all input datasets $D \in \mathcal{X}^n$ with parameter $\delta \in [0, 1)$. Then the algorithm $\mathcal{M}^*_{\mathcal{M}',\mathcal{D},\gamma} : \mathcal{X}^n \to \mathcal{R}$ has the following properties:*

- *$(\varepsilon, 0)$-differential privacy whenever*

$$\delta \leq \frac{e^\varepsilon - 1}{e^\varepsilon + 1} \cdot \frac{\gamma}{1 - \gamma} \cdot \min_{r \in \mathcal{R}} \left\{ \Pr_{Z \sim \mathcal{D}}[Z = r] \right\} \tag{1}$$

- *Running time* $O(\log(1/\gamma)) + \text{Time}(\mathcal{M}') + \text{Time}(\mathcal{D})$ *where* $\text{Time}(\mathcal{D})$ *is the time to sample from the distribution* $\mathcal{D}$.

By taking γ and δ small enough and satisfying (1), the algorithm $\mathcal{M}^*_{\mathcal{M}',\mathcal{D},\gamma}$ satisfies pure differential privacy and has nearly the same utility as $\mathcal{M}$ (due to having a statistical distance at most $\gamma + \delta$ from $\mathcal{M}$) while allowing for a possibly more efficient implementation since we only need to approximately sample from the output distribution of $\mathcal{M}$.

To maximize the minimum in (1), one can take $\mathcal{D} \sim \text{Unif}(\mathcal{R})$. However, it may the case that sampling this distribution exactly is inefficient and we are willing to trade needing a smaller δ to maintain pure differentially privacy for a faster sampling algorithm.

4 Counting Queries

Before discussing algorithms for privately releasing histograms, we show how to privately answer a single counting query using only integers of bounded length. While there exist known algorithms for this problem [10, 21], our algorithms have additional properties that will be used to construct histogram algorithms in later sections. In general, counting queries have as input the dataset $D \in \mathcal{X}^n$ and the bin x to query. However, we will take the true count, $c_x(D)$, as the input to our counting query algorithms. When constructing histogram algorithms in later sections, this will allow us to improve the running time as we will only need to iterate through the dataset once to determine all true counts prior to answering any counting query. In addition, we would like to keep track of the randomness used by our algorithms so we write that as an explicit second input. As a result, we have the following definitions:

▶ **Definition 4.1.** Let $n, d \in \mathbb{N}_+$. We say an algorithm $\mathcal{M} : [n] \times [d]_+ \to [n]$ is **(ε, δ)-differentially private for counting queries** if the algorithm $\mathcal{M} : \{0,1\}^n \to [n]$ defined as $\mathcal{M}(D) = \mathcal{M}(\sum_{i=1}^n D_i, U)$ where $U \sim \text{Unif}([d]_+)$ is (ε, δ)-differentially private.

▶ **Definition 4.2.** Let $n, d \in \mathbb{N}_+$. We say $\mathcal{M} : [n] \times [d]_+ \to [n]$ has **(α, β)-accuracy** if for all $c \in [n]$, $\Pr[|\mathcal{M}(c, U) - c| \leq a] \geq 1 - \beta$ where $U \sim \text{Unif}([d]_+)$.

▶ **Definition 4.3.** Let $n, d \in \mathbb{N}_+$ and $\mathcal{M} : [n] \times [d]_+ \to [n]$ be deterministic. Let the **scaled cumulative distribution function of $\mathcal{M}$ at 0** denoted $F_{\mathcal{M}}$ be the function $F_{\mathcal{M}} : [n] \to [d]_+$ defined as $F_{\mathcal{M}}(z) = d \cdot F_{\mathcal{M}(0,U)}(z)$ where $U \sim \text{Unif}([d]_+)$ for all $z \in [n]$.

4.1 The Geometric Mechanism

As shown by Dwork, McSherry, Nissim and Smith [11], we can privately release a counting query by adding appropriately scaled Laplace noise to the count. Because our algorithm's outputs are counts, we do not need to use continuous noise and instead use a discrete analogue, as in [10, 14].

We say an integer-valued random variable Z follows a **two-sided geometric distribution with scale parameter s centered at $c \in \mathbb{Z}$** (denoted $Z \sim c + \text{Geo}(s)$) if its probability mass function $f_Z(z)$ is proportional to $e^{-|z-c|/s}$. It can be verified that f_Z and its cumulative distribution function F_Z are

$$f_Z(z) = \left(\frac{e^{1/s} - 1}{e^{1/s} + 1}\right) \cdot e^{-|z-c|/s} \qquad F_Z(z) = \begin{cases} \frac{e^{1/s}}{e^{1/s}+1} \cdot e^{-(c-z)/s} & \text{if } z \leq c \\ 1 - \frac{1}{e^{1/s}+1} \cdot e^{-(z-c)/s} & \text{otherwise} \end{cases}$$

for all $z \in \mathbb{Z}$. When c is not specified, it is assumed to be 0.

Now, we state the counting query algorithm using discrete noise, formally studied in [14]. We will not keep track of the randomness used by this algorithm, but to match our syntax for counting query algorithms we use a dummy parameter.

Algorithm 2 $\texttt{GeometricMechanism}_{n,\varepsilon}(c, 1)$ for $c \in [n]$ where $n \in \mathbb{N}_+$ and $\varepsilon > 0$

1. Return $\tilde{c}$ set to $c + \text{Geo}(2/\varepsilon)$ clamped to the interval $[0, n]$. i.e.

$$\tilde{c} = \begin{cases} 0 & \text{if } Z \leq 0 \\ n & \text{if } Z \geq n \\ Z & \text{otherwise} \end{cases} \qquad \text{where } Z = c + \text{Geo}(2/\varepsilon).$$

▶ **Theorem 4.4.** *Let $n \in \mathbb{N}_+$ and $\varepsilon > 0$. Then $\texttt{GeometricMechanism}_{n,\varepsilon} : [n] \times [1]_+ \to [n]$ has the following properties:*

- $\texttt{GeometricMechanism}_{n,\varepsilon}$ *is $(\varepsilon/2, 0)$-differentially private for counting queries [14].*
- $\texttt{GeometricMechanism}_{n,\varepsilon}$ *has (a, β)-accuracy for $\beta \in (0, 1)$ and $a = \lceil (2/\varepsilon) \cdot \ln(1/\beta) \rceil$.*

As presented above, this algorithm needs to store integers of unbounded size since $\text{Geo}(2/\varepsilon)$ is unbounded in magnitude. As noted in [14], by restricting the generated noise to a fixed range we can avoid this problem. However, even when the generated noise is restricted to a fixed range, generating this noise via inverse transform sampling may require infinite precision. By appropriately choosing ε, the probabilities of this noise's cumulative distribution function can be represented with finite precision, and therefore generating this noise via inverse transform sampling only requires finite precision.

▶ **Theorem 4.5.** *Let $n \in \mathbb{N}_+$, $\varepsilon \in \mathbb{N}^{-1}$ and $\tilde{\varepsilon} = 2 \cdot \ln\left(1 + 2^{-\lceil \log(2/\varepsilon) \rceil}\right) \in (4/9 \cdot \varepsilon, \varepsilon]$. Then there is a deterministic algorithm $\texttt{GeoSample}_{n,\varepsilon} : [n] \times [d]_+ \to [n]$ where $\log d = O(n \cdot \log(1/\varepsilon))$ with the following properties:*

- $\texttt{GeoSample}_{n,\varepsilon}(c, U) \sim \texttt{GeometricMechanism}_{n,\tilde{\varepsilon}}(c, 1)$ *where $U \sim \text{Unif}([d]_+)$ for all $c \in [n]$. Thus, $\texttt{GeoSample}_{n,\varepsilon}$ is $(\tilde{\varepsilon}/2, 0)$-differentially private for counting queries and has (a, β)-accuracy for $\beta \in (0, 1)$ and $a = \lceil (2/\tilde{\varepsilon}) \cdot \ln(1/\beta) \rceil$.*
- $\texttt{GeoSample}_{n,\varepsilon}$ *has running time $\tilde{O}(n \cdot \log(1/\varepsilon))$.*
- *For all $z \in [n]$, $F_{\texttt{GeoSample}_{n,\varepsilon}}(z)$ can be computed in time $\tilde{O}(n \cdot \log(1/\varepsilon))$.*

4.2 Approximating Geometric Noise to Release Counting Queries Faster

Notice that $\texttt{GeoSample}_{n,\varepsilon}$ has running time at least linear in n. This is due to evaluating a (scaled) cumulative distribution function operating on integers with bit length $\Omega(n)$. We can improve the running time by approximately sampling from a two-sided geometric distribution. Small tail probabilities are dropped to reduce the number of required bits to represent probabilities to logarithmic in n. And then to recover pure differential privacy, following Lemma 3.1, we mix with uniformly random output.

▶ **Theorem 4.6.** *Let $n \in \mathbb{N}+$, $\varepsilon, \gamma \in \mathbb{N}^{-1}$ and $\tilde{\varepsilon} = 2 \cdot \ln(1 + 2^{-\lceil \log(2/\varepsilon) \rceil}) \in (4/9 \cdot \varepsilon, \varepsilon]$. Then there is a deterministic algorithm $\texttt{FastSample}_{n,\varepsilon,\gamma} : [n] \times [d]_+ \to [n]$ where $\log d = \tilde{O}(1/\varepsilon) \cdot \log(n/\gamma)$ with the following properties:*

- $\texttt{FastSample}_{n,\varepsilon,\gamma}$ *is $(\varepsilon/2, 0)$-differentially private for counting queries.*
- *For every $\beta > \gamma$, $\texttt{FastSample}_{n,\varepsilon,\gamma}$ has (a, β)-accuracy for $a = \lceil (2/\tilde{\varepsilon}) \cdot \ln(1/(\beta - \gamma)) \rceil$.*

- $\texttt{FastSample}_{n,\varepsilon,\gamma}$ *has running time* $\tilde{O}((1/\varepsilon)\cdot\log^2 n + (1/\varepsilon)\cdot\log n\cdot\log(1/\gamma))$.
- *For all* $z\in[n]$, $F_{\texttt{FastSample}_{n,\varepsilon,\gamma}}(z)$ *can be computed in time* $\tilde{O}((1/\varepsilon)\cdot\log(n/\gamma))$.

5 Generalization of the Laplace Mechanism

As shown by Dwork, McSherry, Nissim and Smith [11], we can privately release a histogram by adding independent and appropriately scaled Laplace noise to each bin. Below we state a generalization guaranteeing privacy provided the counting query algorithm used is private and the released counts are independent.

Algorithm 3 $\texttt{BasicHistogram}_{\mathcal{M},A}(D)$ for $D\in\mathcal{X}^n$, $\mathcal{M}:[n]\times[d]_+\to[n]$ and $A\subseteq\mathcal{X}$

1. Compute $c_x(D)$ for all $x\in A$.
2. For each $x\in A$, do the following:
   a. Sample u_x uniformly at random from $[d]_+$.
   b. Let $\tilde{c}_x=\mathcal{M}(c_x(D),u_x)$.
   c. Release $(x,\tilde{c}_x)$.

The output of this algorithm is a collection of bins $(x,\tilde{c}_x)$ representing a partial vector.

▶ **Theorem 5.1.** *Let* $A\subseteq\mathcal{X}$ *and* $\mathcal{M}:[n]\times[d]_+\to[n]$ *be* $(\varepsilon/2,0)$*-differentially private for counting queries and have* (a,β)*-accuracy. Then* $\texttt{BasicHistogram}_{\mathcal{M},A}:\mathcal{X}^n\to\mathbb{N}^A$ *has the following properties:*

- $\texttt{BasicHistogram}_{\mathcal{M},A}$ *is* $(\varepsilon,0)$*-differentially private.*
- *For all* $D\in\mathcal{X}^n$, *we have*

$$\forall x\in A\quad \Pr[|(\texttt{BasicHistogram}_{\mathcal{M},A}(D))_x - c_x(D)|\le a]\ge 1-\beta$$

  In particular, $\texttt{BasicHistogram}_{\mathcal{M},\mathcal{X}}(D)$ *has* (a,β)*-per-query accuracy.*
- *For all* $D\in\mathcal{X}^n$, *we have*

$$\Pr[\forall x\in A\quad |(\texttt{BasicHistogram}_{\mathcal{M},A}(D))_x - c_x(D)|\le a]\ge 1-|A|\cdot\beta$$

  In particular, $\texttt{BasicHistogram}_{\mathcal{M},\mathcal{X}}$ *has* $(a,|\mathcal{X}|\cdot\beta)$*-simultaneous accuracy.*
- *Running time* $O(n\log n\cdot\log|\mathcal{X}| + |A|\cdot(\log n\cdot\log|\mathcal{X}| + \log d + \mathrm{Time}(\mathcal{M})))$.

It is important to note that the privacy guarantee only holds when A is fixed and does not depend on the dataset D. The choice of parameterizing by A will be convenient in defining more complex histogram algorithms later.

$\mathcal{M}$	Running Time	(a,β)-Per-Query	(a,β)-Simul.
`GeometricMechanism`	n/a	$\left\lceil\frac{2}{\varepsilon}\ln\frac{1}{\beta}\right\rceil$	$\left\lceil\frac{2}{\varepsilon}\ln\frac{\lvert\mathcal{X}\rvert}{\beta}\right\rceil$
`GeoSample`	$\tilde{O}(\lvert\mathcal{X}\rvert\cdot n\cdot\log(1/\varepsilon))$	$\left\lceil\frac{9}{2\varepsilon}\ln\frac{1}{\beta}\right\rceil$	$\left\lceil\frac{9}{2\varepsilon}\ln\frac{\lvert\mathcal{X}\rvert}{\beta}\right\rceil$
`FastSample`	$\tilde{O}\left((\lvert\mathcal{X}\rvert/\varepsilon)\cdot\log^2 n\right)+\tilde{O}(n)\cdot\log\lvert\mathcal{X}\rvert$	$\left\lceil\frac{9}{2\varepsilon}\ln\frac{2}{\beta}\right\rceil$	$\left\lceil\frac{9}{2\varepsilon}\ln\frac{2\lvert\mathcal{X}\rvert}{\beta}\right\rceil$

Figure 1 The running time and errors of $\texttt{BasicHistogram}_{\mathcal{M},\mathcal{X}}$ for the counting query algorithms of Section 4. Values shown are for a $(\varepsilon,0)$-differentially private release. For `FastSample`, we take $\gamma=\beta/(2|\mathcal{X}|)$ and assume $\beta\ge 1/n^{O(1)}$.

By taking $\mathcal{M} = \texttt{GeometricMechanism}$, $\texttt{BasicHistogram}_{\mathcal{M},\mathcal{X}}$ is identically distributed to the Truncated Geometric Mechanism of Ghosh, Roughgarden and Sundararajan [14] which achieves per-query and simultaneous accuracy with error up to constant factors matching known lower bounds for releasing a private histogram [17, 2].

6 Improving the Running Time

In this section, we present an algorithm whose running time depends only poly-logarithmically on the universe size while maintaining pure differential privacy based on the observation that most counts are 0 when $n \ll |\mathcal{X}|$; this is the same observation made by Cormode, Procopiuc, Srivastava and Tran [9] to release private histograms that are sparse in expectation.

6.1 Sparse Histograms

We start by reducing the output length of $\texttt{BasicHistogram}_{\mathcal{M},\mathcal{X}}$ to release only the bins with the heaviest (or largest) counts (interpreted as a partial vector).

Algorithm 4 $\texttt{KeepHeavy}_{\mathcal{M}}(D)$ for $D \in \mathcal{X}^n$ where $\mathcal{M} : [n] \times [d]_+ \to [n]$

1. Let $\{(x, \tilde{c}_x)\}_{x\in\mathcal{X}} = \texttt{BasicHistogram}_{\mathcal{M},\mathcal{X}}(D)$.
2. Let $x_1, \ldots, x_{n+1}$ be the elements of $\mathcal{X}$ with the largest counts in sorted order, i.e.

$$\tilde{c}_{x_1} \geq \tilde{c}_{x_2} \geq \ldots \geq \tilde{c}_{x_{n+1}} \geq \max_{x\in\mathcal{X}\setminus\{x_1,\ldots,x_{n+1}\}} \tilde{c}_x$$

3. Release $h = \{(x, \tilde{c}_x) : x \in \mathcal{X} \text{ and } \tilde{c}_x > \tilde{c}_{x_{n+1}}\} \in \mathcal{H}_{n,n}(\mathcal{X})$.

Observe that the output length has been improved to $O(n \cdot (\log|\mathcal{X}| + \log n))$ bits compared to the $O(|\mathcal{X}| \cdot (\log|\mathcal{X}| + \log n))$ bits needed to represent the outputs of $\texttt{BasicHistogram}_{\mathcal{M},\mathcal{X}}$.

▶ **Theorem 6.1.** *Let $\mathcal{M} : [n] \times [d]_+ \to [n]$ be $(\varepsilon/2, 0)$-differentially private for counting queries such that $\texttt{BasicHistogram}_{\mathcal{M},\mathcal{X}}$ has (a_1, β)-per-query accuracy and (a_2, β)-simultaneous accuracy with $a_1 \leq a_2$. Then $\texttt{KeepHeavy}_{\mathcal{M}} : \mathcal{X}^n \to \mathcal{H}_{n,n}(\mathcal{X})$ has the following properties:*

- *$(\varepsilon, 0)$-differential privacy.*
- *$(a_1, 2\beta)$-per-query accuracy on counts larger than $2a_2$.*
- *$(2a_2, \beta)$-simultaneous accuracy.*

Unlike $\texttt{BasicHistogram}_{\mathcal{M},\mathcal{X}}$, by taking $\mathcal{M} = \texttt{GeoSample}$, the algorithm $\texttt{KeepHeavy}_{\mathcal{M}}$ achieves $(O(\log(1/\beta)/\varepsilon), \beta)$-per-query accuracy only on counts larger than $O(\log(|\mathcal{X}|/\beta)/\varepsilon)$. This loss is necessary for any algorithm that outputs a sparse histogram by Theorem 1.5.

However, as described KeepHeavy still requires adding noise to the count of every bin. The following algorithm $\texttt{KH}' : \mathcal{X}^n \to \mathcal{H}_{n,n}(\mathcal{X})$ simulates KeepHeavy by generating a candidate set of heavy bins from which only the heaviest are released. This candidate set is constructed from all bins with nonzero true count and a sample representing the bins with a true count of 0 that have the heaviest noisy counts.

Algorithm 5 $\texttt{KH}'_{\mathcal{M}}(D)$ for $D \in \mathcal{X}^n$ where $\mathcal{M} : [n] \times [d]_+ \to [n]$ and $|\mathcal{X}| \geq 2n + 1$[2]

1. Let $A = \{x \in \mathcal{X} : c_x(D) > 0\}$ and $m = |\mathcal{X} \setminus A|$.
2. Let $\{(x, \tilde{c}_x)\}_{x \in A} = \texttt{BasicHistogram}_{\mathcal{M},A}(D)$.
3. Pick a uniformly random sequence $(q_0, \ldots, q_n)$ of distinct elements from $\mathcal{X} \setminus A$.
4. Sample $(\tilde{c}_{q_0}, \ldots, \tilde{c}_{q_n})$ from the joint distribution of the order statistics $(Z_{(m)}, \ldots, Z_{(m-n)})$ where $Z_1, \ldots, Z_m$ are i.i.d. $\mathcal{M}(0, U)$ random variables with $U \sim \text{Unif}([d]_+)$.
5. Sort the elements of $A \cup \{q_0, \ldots, q_n\}$ as $x_1, \ldots, x_{|A|+n+1}$ such that $\tilde{c}_{x_1} \geq \ldots \geq \tilde{c}_{x_{|A|+n+1}}$.
6. Release $h = \{(x, \tilde{c}_x) : x \in \{x_1, \ldots, x_n\} \text{ and } \tilde{c}_x > \tilde{c}_{x_{n+1}}\} \in \mathcal{H}_{n,n}(\mathcal{X})$.[3]

▶ **Proposition 6.2.** $\texttt{KH}'_{\mathcal{M}}(D)$ *is identically distributed to* $\texttt{KeepHeavy}_{\mathcal{M}}(D)$.

In order to sample from the order statistics used by $\texttt{KH}'_{\mathcal{M}}$ we construct an algorithm using inverse transform sampling similar to the counting query algorithms of Section 4.

▶ **Proposition 6.3.** *Let $n, d \in \mathbb{N}_+$ and $F : [n] \to [d]_+$ such that F is non-decreasing and $F(n) = d$. Let $m \in \mathbb{N}_+$ such that $m \geq n + 1$. Let $Z_1, \ldots, Z_m$ be i.i.d. random variables over $[n]$ with cumulative distribution function $F(z)/d$ for all $z \in [n]$. Then the following algorithm $\texttt{OrdSample}_F(m)$ is identically distributed to the top $n+1$ order statistics $(Z_{(m)}, \ldots, Z_{(m-n)})$.*

Also, $\texttt{OrdSample}_F(m)$ has running time $O(n \log n) \cdot \big(\tilde{O}(m \cdot \log d) + \text{Time}(F)\big)$.

Algorithm 6 $\texttt{OrdSample}_F(m)$ for $m \in \mathbb{N}_+$ such that $m \geq n + 1$ where $F : [n] \to [d]_+$

1. Let $v_{-1} = n$.
2. For $i \in [n]$, do the following:
   a. Sample u_i uniformly at random from $[F(v)^{m-i}]_+$.
   b. Using binary search find the smallest $z \in [v_{i-1}]$ such that $F(z)^{m-i} \geq u_i$. Call it v_i.
3. Return $(v_0, \ldots v_n)$.

Now from $\texttt{KH}'$ we replace sampling from the joint distribution of the order statistics with the explicit sampling algorithm $\texttt{OrdSample}$ to get the following algorithm.

[2] $|\mathcal{X}| \geq 2n + 1$ ensures that $|\mathcal{X} \setminus A| \geq n + 1$. One can use $\texttt{BasicHistogram}_{\mathcal{X}}(D, \mathcal{M})$ when $|\mathcal{X}| \leq 2n$.

[3] If instead $\mathcal{M}$ had real-valued range this last step is equivalent to releasing the n heaviest bins. However, in the discrete case, where ties can occur, from the set $A \cup \{x_1, \ldots, x_n\}$ we cannot determine all bins with a count tied for the n-th heaviest as there may be many other noisy counts tied with $\tilde{c}_{x_n}$. As a result, we only output the bins with a strictly heavier count than $\tilde{c}_{x_{n+1}}$.

Algorithm 7 $\texttt{KH}''_{\mathcal{M},\beta_1}(D)$ for $D \in \mathcal{X}^n$, $\mathcal{M} : [n] \times [d]_+ \to [n]$, $\beta_1 \in \mathbb{N}^{-1}$ and $|\mathcal{X}| \geq 4n$

1. Construct a sequence Q of $\lceil (4n+2)/\beta_1 \rceil$ elements sampled uniformly at random from $\mathcal{X}$. If Q has less than $2n+1$ distinct elements release an empty histogram and stop.[4]
2. Let $A = \{x \in \mathcal{X} : c_x(D) > 0\}$ and $m = |\mathcal{X} \setminus A|$.
3. Let $\{(x, \tilde{c}_x)\}_{x \in A} = \texttt{BasicHistogram}_{\mathcal{M},A}(D)$.
4. Let $(q_0, \ldots, q_n)$ be the first $n+1$ distinct elements of Q not in A.
5. Let $(\tilde{c}_{q_0}, \ldots, \tilde{c}_{q_n}) = \texttt{OrdSample}_{F_{\mathcal{M}}}(m)$.
6. Sort the elements of $A \cup \{q_0, \ldots, q_n\}$ as $x_1, \ldots, x_{|A|+n+1}$ such that $\tilde{c}_{x_1} \geq \ldots \geq \tilde{c}_{x_{|A|+n+1}}$.
7. Release $h = \{(x, \tilde{c}_x) : x \in \{x_1, \ldots, x_n\} \text{ and } \tilde{c}_x > \tilde{c}_{x_{n+1}}\} \in \mathcal{H}_{n,n}(\mathcal{X})$.

▶ **Theorem 6.4.** *Let deterministic $\mathcal{M} : [n] \times [d]_+ \to [n]$ be $(\varepsilon/2, 0)$-differentially private for counting queries such that $\texttt{BasicHistogram}_{\mathcal{X}}(D, \mathcal{M})$ has (a_1, β_2)-per-query accuracy and (a_2, β_2)-simultaneous accuracy with $a_1 \leq a_2$. Then $\texttt{KH}''_{\mathcal{M},\beta_1} : \mathcal{X}^n \to \mathcal{H}_{n,n}(\mathcal{X})$ has the following properties:*

- *$(\varepsilon, 0)$-differential privacy.*
- *$(a_1, \beta_1 + 2\beta_2)$-per-query accuracy on counts larger than $2a_2$.*
- *$(2a_2, \beta_1 + \beta_2)$-simultaneous accuracy.*

This algorithm only has an output of length $O(n \cdot (\log |\mathcal{X}| + \log n))$. However, its running time depends polynomially on $|\mathcal{X}|$ since sampling the m^{th} order statistic, $\tilde{c}_{q_0}$, using $\texttt{OrdSample}$ takes time at least linear in $m \geq |\mathcal{X}| - n$. Indeed, this is necessary since the distribution of the order statistic $Z_{(m)}$ has probabilities that are exponentially small in m.

6.2 An Efficient Approximation

To remedy the inefficiency of $\texttt{KH}''$ we consider an efficient algorithm that approximates the output distribution of $\texttt{KH}''$.

▶ **Theorem 6.5.** *Let $\beta_1, \delta \in \mathbb{N}^{-1}$ and $\mathcal{M} : [n] \times [d]_+ \to [n]$ be $(\varepsilon/2, 0)$-differentially private for counting queries. Then there exists an algorithm $\texttt{SparseHistogram}_{\mathcal{M},\beta_1,\delta} : \mathcal{X}^n \to \mathcal{H}_{n,n}(\mathcal{X})$ with the following properties:*

- *$\Delta\big(\texttt{KH}''_{\mathcal{M},\beta_1}(D), \texttt{SparseHistogram}_{\mathcal{M},\beta_1,\delta}(D)\big) \leq \delta$ for all $D \in \mathcal{X}^n$.*
- *$\texttt{SparseHistogram}_{\mathcal{M},\beta_1,\delta}$ is $(\varepsilon, (e^{\varepsilon}+1)\cdot\delta)$-differentially private.*
- *Moreover, for $\beta_1 \geq 1/\log^{O(1)} n$ the running time of $\texttt{SparseHistogram}_{\mathcal{M},\beta_1,\delta}$ is*

$$\tilde{O}(n) \cdot \big(\log |\mathcal{X}| \cdot \tilde{O}(\log d + \log(1/\delta) + \log |\mathcal{X}|)\big) + O(n \log n) \cdot (\text{Time}(\mathcal{M}) + \text{Time}(F_{\mathcal{M}}))$$

Note that this algorithm only achieves $(\varepsilon, O(\delta))$-differential privacy. By reducing δ, the algorithm better approximates $\texttt{KH}''$, at the cost of increasing running time (polynomial in the bit length of δ). Notice that $\texttt{KH}''$ passes an argument to $\texttt{OrdSample}$ that results in $\texttt{OrdSample}$ exponentiating an integer, which represents the numerator of a fraction $a/b = F(z)/F(v)$, to a power $i \geq |\mathcal{X}| - 2n$. To improve the efficiency of $\texttt{OrdSample}$, we want to ensure that

[4] Along with step 4, this process allows us to generate $n+1$ distinct elements of $\mathcal{X} \setminus A$ with running time that has a nearly linear dependence on n (whenever $\beta_1 \geq 1/\log^{O(1)} n$) at the cost of an additive increase in failure probability. However, if we are willing to accept a nearly quadratic dependence on n, we can always sample the distinct elements instead of just with high probability.

the numbers it uses do not exceed some maximum s with bit length polynomial in n and $\log|\mathcal{X}|$. We achieve this by approximating $s \cdot (a/b)^i$ using repeated squaring and truncating each intermediate result to keep its bit length manageable. This process results in sampling the order statistics to within a statistical distance of δ.

Now, we convert $\texttt{SparseHistogram}_{\mathcal{M},\beta_1,\delta}$ to a pure differentially private algorithm by mixing it with random output following Lemma 3.1.

Algorithm 8 $\texttt{PureSparseHistogram}_{\mathcal{M},\varepsilon,\beta_1,\beta_2}(D)$ for $D \in \mathcal{X}^n$ where $\mathcal{M} : [n] \times [d]_+ \to [n]$, $\varepsilon, \beta_1, \beta_2 \in \mathbb{N}^{-1}$ and $|\mathcal{X}| \geq 4n$

1. With probability $1 - \beta_2$ release $\texttt{SparseHistogram}_{\mathcal{M},\beta_1,\delta}(D)$ with
$$\delta = \frac{\varepsilon}{3} \cdot \beta_2 \cdot \left(\frac{1}{6 \cdot |\mathcal{X}|}\right)^n$$
2. Otherwise
   a. Draw $(x_1, \ldots x_n)$ uniformly at random from $\mathcal{X}$.
   b. Let Q be the set of distinct elements from $(x_1, \ldots, x_n)$.
   c. For each $q \in Q$, sample $\tilde{c}_q$ uniformly at random from $[n]$.
   d. Release $h = \{(q, \tilde{c}_q) : q \in Q \text{ and } \tilde{c}_q > 0\} \in \mathcal{H}_{n,n}(\mathcal{X})$.

▶ **Theorem 6.6.** *Let $\varepsilon, \beta_1, \beta_2 \in \mathbb{N}^{-1}$ and deterministic $\mathcal{M} : [n] \times [d]_+ \to [n]$ be $(\varepsilon/2, 0)$-differentially private for counting queries such that $\texttt{BasicHistogram}_{\mathcal{M},\mathcal{X}}$ has (a_1, β_3)-per-query accuracy and (a_2, β_3)-simultaneous accuracy with $a_1 \leq a_2$.*

Then $\texttt{PureSparseHistogram}_{\mathcal{M},\varepsilon,\beta_1,\beta_2} : \mathcal{X}^n \to \mathcal{H}_{n,n}(\mathcal{X})$ has the following properties:

- *$(\varepsilon, 0)$-differential privacy.*
- *$(a_1, \beta_1 + 2\beta_2 + 2\beta_3)$-per-query accuracy on counts larger than $2a_2$.*
- *$(2a_2, \beta_1 + 2\beta_2 + \beta_3)$-simultaneous accuracy.*
- *For $\beta_1 \geq 1/\log^{O(1)} n$ and $\beta_2 \geq 1/O(2^n)$, the running time is*

$$\tilde{O}(n^2 \cdot \log^2|\mathcal{X}| + n \cdot \log(d/\varepsilon) \cdot \log|\mathcal{X}|) + O(n \log n) \cdot (\text{Time}(\mathcal{M}) + \text{Time}(F_{\mathcal{M}}))$$

$\mathcal{M}$	Running Time	(a,β)-Per-Query on $c_x(D) > t$	(a,β)-Simultaneous
`GeoSample`	$\tilde{O}\left(n^2 \cdot \log^2|\mathcal{X}|\right)$	$\left\lceil \frac{9}{2\varepsilon} \ln \frac{4}{\beta} \right\rceil$	$2 \cdot \left\lceil \frac{9}{2\varepsilon} \ln \frac{4\lvert\mathcal{X}\rvert}{\beta} \right\rceil$	$2 \cdot \left\lceil \frac{9}{2\varepsilon} \ln \frac{2\lvert\mathcal{X}\rvert}{\beta} \right\rceil$
`FastSample`	$\tilde{O}\left(n^2 \cdot \log^2|\mathcal{X}|\right)$	$\left\lceil \frac{9}{2\varepsilon} \ln \frac{8}{\beta} \right\rceil$	$2 \cdot \left\lceil \frac{9}{2\varepsilon} \ln \frac{8\lvert\mathcal{X}\rvert}{\beta} \right\rceil$	$2 \cdot \left\lceil \frac{9}{2\varepsilon} \ln \frac{4\lvert\mathcal{X}\rvert}{\beta} \right\rceil$

Figure 2 The running time and errors of $\texttt{SparseHistogram}_{\mathcal{M},\varepsilon,\beta/6,\beta/6}(D)$ for the counting query algorithms of Section 4. For per-query accuracy, the first value is the error a and the second value is the threshold t. Values shown are for a $(\varepsilon, 0)$-differentially private release. We assume $\varepsilon \geq 1/O(n)$ and $\beta \geq 1/\log^{O(1)} n$. For `FastSample`, we take $\gamma = \beta/(4|\mathcal{X}|)$.

7 Better Per-Query Accuracy via Compact, Non-Sparse Representations

In this section, we present a histogram algorithm whose running time is poly-logarithmic in $|\mathcal{X}|$, but, unlike Algorithm 8, is able to achieve $(O(\log(1/\beta)/\varepsilon), \beta)$-per query accuracy. It

will output a histogram from a properly chosen family of succinctly representable histograms. This family necessarily contains histograms that have many nonzero counts to avoid the lower bound of Theorem 1.5.

7.1 The Family of Histograms

We start by defining this family of histograms.

▶ **Lemma 7.1.** *Let $\mathcal{M}_0 : [d_0]_+ \to [n]$, $d_0 = 2^{2\cdot 3^\ell}$ for some $\ell \in \mathbb{N}$, $d_0 \geq |\mathcal{X}|$ and $U \sim \mathrm{Unif}([d_0]_+)$. There exists a multiset of histograms $\mathcal{G}_{\mathcal{M}_0}(\mathcal{X})$ satisfying:*

- *Let $g \sim \mathrm{Unif}(\mathcal{G}_{\mathcal{M}_0}(\mathcal{X}))$. For all $x \in \mathcal{X}$, the marginal distribution g_x is distributed according to $\mathcal{M}_0(U)$.*
- *Let $g \sim \mathrm{Unif}(\mathcal{G}_{\mathcal{M}_0}(\mathcal{X}))$. For all $B \subseteq \mathcal{X}$ such that $|B| \leq n+1$ and for all $c \in [n]^B$*

$$\Pr[\forall x \in B \;\; g_x = c_x] = \prod_{x \in B} \Pr[g_x = c_x]$$

- *For all $g \in \mathcal{G}_{\mathcal{M}_0}(\mathcal{X})$, the histogram g can be represented by a string of length $O(n \cdot \log d_0)$ and given this representation for all $x \in \mathcal{X}$ the count g_x can be evaluated in time $O(n) \cdot \tilde{O}(\log d_0) + \mathrm{Time}(\mathcal{M}_0)$.*
- *For all $A \subseteq \mathcal{X}$ such that $|A| \leq n$ and $c \in [n]^A$ sampling a histogram h uniformly at random from $\{g \in \mathcal{G}_{\mathcal{M}_0}(\mathcal{X}) : \forall x \in A \;\; g_x = c_x\}$ can be done in time $O(n) \cdot \mathrm{Time}(\mathcal{S}) + \tilde{O}(n \cdot \log^2 d_0)$ where $\mathrm{Time}(\mathcal{S})$ is the maximum time over $v \in [n]$ to sample from the distribution $\mathcal{S}_v \sim \mathrm{Unif}(\{u_0 \in [d_0]_+ : \mathcal{M}_0(u_0) = v\})$.*

Proof. (Construction) Let $\mathcal{G}'_{\mathcal{M}_0}(\mathcal{X})$ be the set of all degree at most n polynomials over the finite field $\mathbb{F}_{d_0}$. Now, $\mathcal{G}'_{\mathcal{M}_0}(\mathcal{X})$ is a $(n+1)$-wise independent hash family mapping $\mathbb{F}_{d_0}$ to $\mathbb{F}_{d_0}$. And given any function $p_g \in \mathcal{G}'_{\mathcal{M}_0}(\mathcal{X})$ we construct a histogram $g \in \mathcal{G}_{\mathcal{M}_0}(\mathcal{X})$ by using $p_g(x)$ as the randomness for $\mathcal{M}_0$. More specifically, let $T : \mathbb{F}_{d_0} \to [d_0]_+$ be a bijection and for all $x \in \mathcal{X}$, define $g_x = \mathcal{M}_0(T(p_g(x)))$. ◀

7.2 The Algorithm

We can think of taking $d_0 = d$ and $\mathcal{M}_0(u) = \mathcal{M}(0, u)$ for all $u \in [d]_+$, but for technical reasons (e.g. requiring $d_0 \geq |\mathcal{X}|$), we will allow $\mathcal{M}_0(u)$ to approximate $\mathcal{M}(0, u)$. In this way, a histogram picked uniformly at random from the family $\mathcal{G}_{\mathcal{M}_0}(\mathcal{X})$ will have the desired marginal distributions for all empty bins.

Thus, for our algorithm to have the correct marginal distributions over all bins we first compute the noisy counts for the nonzero bins and then randomly pick a histogram from our family that is consistent with these computed counts.

Algorithm 9 $\texttt{CompactHistogram}_{\mathcal{M},\mathcal{M}_0}(D)$ for $D \in \mathcal{X}^n$ where $\mathcal{M} : [n] \times [d]_+ \to [n]$ and $\mathcal{M}_0 : [d_0]_+ \to [n]$ such that $d_0 = 2^{2\cdot 3^\ell}$ for some $\ell \in \mathbb{N}$ and $d_0 \geq |\mathcal{X}|$

1. Let $A = \{x \in \mathcal{X} : c_x(D) > 0\}$.
2. Let $\{(x, \tilde{c}_x)\}_{x \in A} = \texttt{BasicHistogram}_{\mathcal{M},A}(D)$.
3. Release h drawn uniformly at random from $\{g \in \mathcal{G}_{\mathcal{M}_0}(\mathcal{X}) : \forall x \in A \;\; g_x = \tilde{c}_x\}$.

▶ **Theorem 7.2.** *Let deterministic* $\mathcal{M} : [n] \times [d]_+ \to [n]$ *be* $(\varepsilon_1/2, 0)$*-differentially private for counting queries and have* (a, β)*-accuracy. Let deterministic* $\mathcal{M}_0 : [d_0]_+ \to [n]$ *such that* $d_0 = 2^{2 \cdot 3^\ell}$ *for some* $\ell \in \mathbb{N}$ *and* $d_0 \geq |\mathcal{X}|$*. Assume* $\Pr[\mathcal{M}_0(U_0) \leq a] \geq 1 - \beta$ *and for all* $c \in [n]$

$$e^{-\varepsilon_2} \cdot \Pr[\mathcal{M}_0(U_0) = c] \leq \Pr[\mathcal{M}(0, U) = c] \leq e^{\varepsilon_3} \cdot \Pr[\mathcal{M}_0(U_0) = c]$$

where $U \sim \mathrm{Unif}([d]_+)$ *and* $U_0 \sim \mathrm{Unif}([d_0]_+)$*. Then* $\texttt{CompactHistogram}_{\mathcal{M}, \mathcal{M}_0}(D)$ *has the following properties:*

- $(\varepsilon_1 + \varepsilon_2 + \varepsilon_3, 0)$*-differential privacy.*
- (a, β)*-per-query accuracy.*
- $(a, |\mathcal{X}| \cdot \beta)$*-simultaneous accuracy.*
- *Running time* $\tilde{O}(n \cdot \log d + n \cdot \mathrm{Time}(\mathcal{M}) + n \cdot \mathrm{Time}(\mathcal{S}) + n \cdot \log^2 d_0)$ *where* $\mathrm{Time}(\mathcal{S})$ *is the maximum time over* $v \in [n]$ *to sample from the distribution* $\mathcal{S}_v \sim \mathrm{Unif}(\{u_0 \in [d_0]_+ : \mathcal{M}_0(u_0) = v\})$*.*
- *Given its output* h*, the count* h_x *can be evaluated in time* $O(n) \cdot \tilde{O}(\log d_0) + \mathrm{Time}(\mathcal{M}_0)$ *for all* $x \in \mathcal{X}$*.*

As discussed above, a natural choice for $\mathcal{M}_0$ is to take $d_0 = d$ and $\mathcal{M}_0(u) = \mathcal{M}(0, u)$ for all $u \in [d]_+$. Although d does not does satisfy the required constraints for the counting algorithms of Section 4, as we show in the full version of the paper [1], we can construct $\mathcal{M}_0$ for our counting query algorithms at only a constant loss in privacy.

▶ **Corollary 7.3.** *Let* $\varepsilon \in \mathbb{N}^{-1}$*,* $\beta \in \mathbb{N}^{-1}$ *such that* $\beta \geq 1/n^{O(1)}$ *and* $\mathcal{M} = \texttt{FastSample}_{n, \varepsilon', \gamma}$ *where* $\varepsilon' = 1/\lceil 10/(9\varepsilon) \rceil$ *and* $\gamma = \beta/(2|\mathcal{X}|)$*. Then there exists* $\mathcal{M}_0 : [d_0]_+ \to [n]$ *where* $\log d_0 = \tilde{O}(1/\varepsilon) \cdot (\log n + \log |\mathcal{X}|)$ *such that* $\texttt{CompactHistogram}_{\mathcal{M}, \mathcal{M}_0}$ *has the following properties:*

- $(\varepsilon, 0)$*-differential privacy.*
- (a, β)*-per-query accuracy for* $a = \lceil (5/\varepsilon) \cdot \ln(2/\beta) \rceil$*.*
- (a, β)*-simultaneous accuracy for* $a = \lceil (5/\varepsilon) \cdot \ln(2|\mathcal{X}|/\beta) \rceil$*.*
- *Running time* $O(n) \cdot \tilde{O}((1/\varepsilon^2) \cdot (\log^2 n + \log^2 |\mathcal{X}|))$*.*
- *Given its output, a count can be computed in time* $O(n) \cdot \tilde{O}((1/\varepsilon) \cdot (\log n + \log |\mathcal{X}|))$*.*

Acknowledgements. We thank the Harvard Privacy Tools differential privacy research group, particularly Mark Bun and Kobbi Nissim, for informative discussions and feedback. And we thank Ashwin Machanavajjhala, Frank McSherry, Uri Stemmer, and the anonymous TPDP and ITCS reviewers for their helpful comments.

References

1 Victor Balcer and Salil P. Vadhan. Differential privacy on finite computers. *CoRR*, abs/1709.05396, 2017. arXiv:1709.05396.

2 Amos Beimel, Hai Brenner, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. Bounds on the sample complexity for private learning and private data release. *Machine learning*, 94(3):401–437, 2014.

3 Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to noninteractive database privacy. *J. ACM*, 60(2):12:1–12:25, 2013. doi:10.1145/2450142.2450148.

4 Mark Bun, Kobbi Nissim, and Uri Stemmer. Simultaneous private learning of multiple concepts. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 369–380. ACM, 2016. doi:10.1145/2840728.2840747.

5 Bryan Cai, Constantinos Daskalakis, and Gautam Kamath. Priv'it: Private and sample efficient identity testing. *CoRR*, abs/1703.10127, 2017. arXiv:1703.10127.

6 Karthekeyan Chandrasekaran, Justin Thaler, Jonathan Ullman, and Andrew Wan. Faster private release of marginals on small databases. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 387–402. ACM, 2014. doi:10.1145/2554797.2554833.

7 Mahdi Cheraghchi, Adam Klivans, Pravesh Kothari, and Homin K. Lee. Submodular functions are noise stable. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1586–1592, Philadelphia, PA, USA, 2012. Society for Industrial and Applied Mathematics.

8 Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

9 Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Thanh T. L. Tran. Differentially private summaries for sparse data. In Alin Deutsch, editor, *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 299–311. ACM, 2012. doi:10.1145/2274576.2274608.

10 Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Eurocrypt*, volume 4004, pages 486–503. Springer, 2006.

11 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, volume 3876, pages 265–284. Springer, 2006.

12 Cynthia Dwork, Aleksandar Nikolov, and Kunal Talwar. Efficient algorithms for privately releasing marginals via convex relaxations. *Discrete & Computational Geometry*, 53(3):650–673, 2015. doi:10.1007/s00454-015-9678-x.

13 Ivan Gazeau, Dale Miller, and Catuscia Palamidessi. Preserving differential privacy under finite-precision semantics. In Luca Bortolussi and Herbert Wiklicky, editors, *Proceedings 11th International Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2013, Rome, Italy, March 23-24, 2013.*, volume 117 of *EPTCS*, pages 1–18, 2013. doi:10.4204/EPTCS.117.1.

14 Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012.

15 Anupam Gupta, Aaron Roth, and Jonathan Ullman. Iterative constructions and private data release. *Theory of Cryptography*, pages 339–356, 2012.

16 Moritz Hardt, Guy N. Rothblum, and Rocco A. Servedio. Private data release via learning thresholds. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 168–187, Philadelphia, PA, USA, 2012. Society for Industrial and Applied Mathematics.

17 Moritz Hardt and Kunal Talwar. On the geometry of differential privacy. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 705–714. ACM, 2010. doi:10.1145/1806689.1806786.

18 Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

19 Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, and Alexandros Ntoulas. Releasing search queries and clicks privately. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 171–180. ACM, 2009. doi:10.1145/1526709.1526733.

20 Jacobus Hendricus van Lint. *Introduction to coding theory*. Springer, 1982.

21 Ilya Mironov. On significance of the least significant bits for differential privacy. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 650–661. ACM, 2012. doi:10.1145/2382196.2382264.

22 Arnold Schönhage. Schnelle multiplikation von polynomen über körpern der charakteristik 2. *Acta Informatica*, 7(4):395–398, 1977.

23 Justin Thaler, Jonathan Ullman, and Salil Vadhan. Faster algorithms for privately releasing marginals. In *International Colloquium on Automata, Languages, and Programming*, pages 810–821. Springer, 2012.

24 Jonathan Ullman and Salil P. Vadhan. Pcps and the hardness of generating private synthetic data. In *TCC*, volume 6597, pages 400–416. Springer, 2011.

25 Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.