Journal of Privacy and Confidentiality
Vol. 15 (3) 2025

Submitted August 2023
Published December 2025

AVOIDING FLOATING-POINT SIDE CHANNELS IN THE REPORT NOISY MAX WITH GAP MECHANISM

KINCHIN TONG, ZEYU DING, JOHN DURRELL, DANIEL KIFER, PROTTAY PROTIVASH, GUANHONG WANG, YUXIN WANG, YINGTAI XIAO, AND DANFENG ZHANG

Binghamton University, Binghamton, NY 13902
e-mail address: ktong1@binghamton.edu

e-mail address: dding1@binghamton.edu

Penn State University, University Park, PA 16802
e-mail address: jmd6968@psu.edu

e-mail address: dkifer@cse.psu.edu

e-mail address: pxp945@psu.edu

University of Maryland, College Park, MD 20742
e-mail address: guanhong@umd.edu

e-mail address: yxwang@psu.edu

e-mail address: yxx5224@psu.edu

Duke University, Durham, NC 27708
e-mail address: danfeng.zhang@duke.edu

Abstract. The Noisy Max mechanism and its variations are fundamental private selection algorithms that are used to select items from a set of candidates (such as the most common diseases in a population), while controlling the privacy leakage in the underlying data. A recently proposed extension, Noisy Top-k with Gap, provides numerical information about how much better the selected items are compared to the non-selected items (e.g., how much more common are the selected diseases). This extra information comes at no privacy cost but crucially relies on infinite precision for the privacy guarantees. In this paper, we provide a finite-precision secure implementation of this algorithm that takes advantage of integer arithmetic.

Key words and phrases: differential privacy, floating-point vulnerability, report noisy top-k.

JPrivacy Confidentiality
www.journalprivacyconfidentiality.org
DOI:10.29012/jpc.894

1. Introduction

Differential privacy [16] is a de facto standard for data collectors to publish information about sensitive datasets while protecting the confidentiality of individuals who contribute data. It is widely adopted by government statistical agencies [36, 8, 25, 1, 20] and in industry [17, 7, 6, 11, 3, 31, 39]. To achieve differential privacy, most algorithms introduce noise from continuous probability distributions (e.g. the Laplace distribution) to mask the effect of any individual's data on the output of the algorithms. In practice, however, these distributions cannot be faithfully represented, much less sampled from, on computers which use only finite precision approximations (e.g., floating-point numbers) to real number arithmetic.

It might appear that such issues are purely of theoretical interest and do not cause serious harm in practice. Unfortunately, this is not the case: Mironov [40] demonstrated that the textbook implementation of the Laplace Mechanism, the most basic algorithm to satisfy differential privacy, can lead to catastrophic failures of privacy, causing entire datasets to be reconstructed with a negligible privacy budget. This is due to a type of side channel attack that exploits the porous approximation of the reals using floating-point numbers. By examining the low-order bits of the noisy output, a large amount of infeasible candidate inputs can be eliminated and the true (noiseless) input value can often be determined. Furthermore, rounding the outputs of an insecure noise distribution does not resolve the problem [40]. As a result of this demonstration, effort has been placed into developing implementations of algorithms that *exactly* sample from discrete distributions [4, 29, 38, 13, 26], given a source of uniform randomness. In many cases, these discrete distributions can replace the approximate (and insecure) continuous noise sampling algorithms that are found in standard statistical libraries. For many differentially private mechanisms, appropriately rounding the inputs [10] and replacing their use of continuous noise with exact discrete samplers is enough to make their implementations secure from these floating point vulnerabilities. However, for the private selection mechanism we study in this paper, Noisy Top-k with Gap [12, 14], such a drop-in replacement of discrete noise for continuous noise does not work and hence we propose a secure implementation.

Briefly, private selection mechanisms take a list of queries and a dataset as input. They output, with high probability, the identity of the query that has the largest value on the input dataset. For example, the Noisy Max algorithm [15] takes a list of queries with sensitivity 1 (i.e., each query answer changes by at most 1 when a person is added to or removed from the data). It adds noise to each query answer and returns the identity of the query with the largest noisy answer. Such mechanisms serve as key components in many privacy preserving algorithms for synthetic data generation [27], ordered statistics [7], quantiles [41], frequent itemset mining [5], hyperparameter tuning [34] for statistical models, etc. Recently, Ding et al. [12, 14] proposed novel variations of these selection mechanisms, including Noisy Max [15], that provide more functionality at the same privacy cost under pure differential privacy.

In the case of the Noisy Max algorithm, Ding et al. [12, 14] showed that, in addition to releasing the identity of the query whose noisy answer is the largest, it is possible to release a numerical estimate of the gap between the values of the returned query and the next best query. This extra information comes at no additional cost to privacy, meaning that the original Noisy Max mechanism threw away useful information. This result can be generalized to the setting in which one wants to estimate the identities of the top k queries – one can release (for free) estimates of all of the gaps between each top k query and the next best query (i.e., the gap between the noisy best and noisy second best queries, the gap

between the noisy second and noisy third best queries, etc). The generalized algorithm is called Noisy Top-k with Gap in their paper. For completeness, we include it as Algorithm 1 in Section 4.

Noisy Top-k with Gap can release strictly more information at no additional cost to privacy which, when combined with subsequent noisy answers to each of the returned queries, can significantly reduce squared error of query estimates [12, 14]. However, as explained in more detail in Section 4.2, this algorithm is much more difficult to implement securely than Noisy Top-k. Its proof of privacy crucially relies on the differences (gaps) between pairs of random variables. Simply working in the integer domain and replacing Laplace with Discrete Laplace or exponential with the geometric distribution will not work as it invalidates the proof [12, 14] of privacy. So more involved alterations are needed to create a secure version of Noisy Top-k with Gap.

In this work, we propose an implementation of Noisy Top-k with Gap that is secure on finite computers. We make two modest assumptions. First, the privacy loss budget parameter ε should be a rational number. Second, the gaps should be represented as rational numbers and the user provides a desired denominator (integer) for these rational numbers. We refer to the reciprocal of this denominator as the *target resolution* γ_*, so that all returned gaps are multiples of γ_* (e.g., multiples of 2^{-10}).

The secure algorithm is designed to be mathematically equivalent to running the ideal, infinite-precision algorithm, and then rounding all the gaps to the nearest multiple of γ_*. Internally, the secure algorithm makes use of the exact geometric distribution sampler over a discrete domain [9]. It dynamically chooses the right level of finite precision it needs to work with so that its control flow matches what the ideal algorithm would have done. This is followed by a carefully controlled gap calculation step that is tricky because the gap (difference) between two discretized distributions is generally not the same as the discretized gap between two continuous distributions.

To summarize, our contributions are:

(1) We propose an implementation of the Noisy Top-k with Gap algorithm [12, 14] that relies solely on integer operations and discrete probability distributions, hence is free from floating point vulnerabilities and can be implemented on finite precision machines.
(2) We prove the correctness of our implementation by carefully analysing its output distribution and show that it is equivalent to the original (ideal) Noisy Top-k with Gap (Algorithm 1) followed by a post-processing step of rounding the gaps.
(3) We evaluate our implementation on real and synthetic datasets and show that our secure implementation incurs moderate overhead over the (insecure) baseline algorithm. Our code is available at `https://github.com/cmla-psu/Secure-GapTopK`.

The rest of the paper is organized as follows. We discuss related work in Section 2 and the relevant background in Section 3. We present our algorithms in Section 4 and proofs for their correctness in Section 5. We implement our algorithms, evaluate their performance on real datasets and report the results in Section 6. Finally we conclude in Section 7.

2. Related Work

The floating point vulnerability in differentially private systems and its severity was first studied by Mironov [40] and Gazeau et al. [21]. They studied the effect of floating point on the resulting privacy guarantee. As an example, Mironov [40] demonstrated that by examining the low-order bits of the noisy outputs of the Laplace mechanism, the noiseless

value can often be determined. As a remedy, Mironov proposed the *clamping mechanism* as a defence, but the defence tended to have worse utility than the Laplace mechanism.

A discrete alternative to the Laplace mechanism was proposed by Ghosh et al. [23] and is equivalent to adding noise from a two-sided geometric distribution. The discrete Laplace mechanism satisfies ε-differential privacy when the inputs are integers or appropriately rounded to integers and can be implemented exactly using the procedure proposed by Canonne et al. [9]. In the same paper, Canonne et al. also showed how to exactly sample from a discrete analogue of the Gaussian distribution [9], which can be used as a replacement for the continuous Gaussian when the inputs are integers or appropriately rounded. Their algorithms employ subroutines that sample from Bernoulli and Geometric distributions without relying on floating-point arithmetic. Our implementation adopts these same subroutines, which are discussed in greater detail in Section 4. It should be noted that the discrete counterparts of the Laplace and Gaussian mechanisms may be susceptible to timing attack: observers can measure the time to draw (discrete) Laplace or Gaussian noise to predict the noise magnitude [30]. Holohan et al. [28] proposed a method to sample Laplace and Gaussian noise in a way that makes it computationally harder for an attacker to reverse-engineer the output, without any rounding.

Secure implementations for several other mechanisms are also available, including histogram approximation [4], the Exponential Mechanism [29] and the Noisy Max algorithm [38, 13]. The Noisy Max algorithm with Gumbel noise is equivalent to the Exponential Mechanism [24, 37], which can be implemented exactly using base-2 exponentiation in place of base-e [29]. The only approximation error introduced arises from converting the base-2 privacy parameter back to its base-e equivalent. More recently, Haney et al. [26] proposed a sampling method called *interval refining* which iteratively shrinks the internal $[0, 1)$ until it is sufficiently small that the inverse images of both end points (hence all points in the interval) under the CDF of the sampling distribution round to the same floating point value. However, none of the above techniques directly apply to the Noisy Top-k with Gap algorithm as its privacy proof crucially depends on the difference between pairs of random variables [12, 14]. For example, if Z_1, Z_2 and Z_3 are random variables, existing techniques can correctly discretize each variable independently, but a secure implementation of Noisy Top-k with Gap may need to discretize pairwise differences like $Z_1 - Z_2$ and $Z_2 - Z_3$ – these pairwise differences are clearly not independent of each other as they have variables in common.

3. Background

3.1. Differential Privacy.

Differential privacy [16] is currently the gold standard for releasing privacy-preserving information about a confidential database. It relies on the notion of adjacent datasets. Two datasets D and D' in some universe $\mathcal{D}$ are adjacent (or neighbors), denoted by $D \sim D'$, if they differ on the presence or absence of some individual's data. Differential privacy ensures that the results of a computation on adjacent datasets are nearly indistinguishable. The degree of indistinguishability is quantified by a parameter $\varepsilon > 0$ called the privacy loss budget; the smaller ε is, the more privacy is provided.

Definition 3.1 (Differential Privacy [16])**.** Let $\varepsilon > 0$. Let $\mathcal{M}$ be randomized algorithm which takes a dataset $D \in \mathcal{D}$ as input. Then $\mathcal{M}$ satisfies (pure) ε-differential privacy if for

all pairs of adjacent datasets $D \sim D' \in \mathcal{D} \times \mathcal{D}$ and all output sets S, we have

$$P\left(\mathcal{M}(D) \in S\right) \leq e^{\varepsilon} P\left(\mathcal{M}(D') \in S\right)$$

where the probability is over the randomness of the algorithm $\mathcal{M}$.

Differential privacy enjoys the following properties:

- Post-processing resilience. If the output of an ε-differentially private algorithm $\mathcal{M}$ goes through another computation $\mathcal{A}$ which does not use the datasets, then the composite algorithm $\mathcal{A} \circ \mathcal{M}$ still satisfies ε-differential privacy. In other words, privacy is not reduced by post-processing.
- Composition. If $\mathcal{M}_1, \mathcal{M}_2$ satisfy differential privacy with privacy loss budgets $\varepsilon_1, \varepsilon_2$, the algorithm that runs both and releases their outputs satisfies $(\varepsilon_1 + \varepsilon_2)$-differential privacy. This result can be generalized to any finite number of differentially private algorithms.

Because of the compositional property of differential privacy, algorithms that satisfy differential privacy are usually built on smaller components called *mechanisms*. Many differentially private algorithms take advantage of the Laplace mechanism [16], which provides a noisy answer to a vector-valued function f based on its ℓ_1-*global sensitivity* Δ_f, defined as follows:

Definition 3.2 (Global Sensitivity [15])**.** The (ℓ_1-)global sensitivity Δ_f of a vector-valued function f with domain $\mathcal{D}$ is

$$\Delta_f = \sup_{D \sim D'} \|f(D) - f(D')\|_1$$

where the supremum is taken over all adjacent pairs $D \sim D'$ from $\mathcal{D}$.

Theorem 3.3 (Laplace Mechanism [16])**.** *Given a privacy loss budget ε, consider the mechanism that returns $\mathcal{M}(D) = f(D) + H$, where H is a vector of independent random samples from the* $\mathrm{Lap}(\Delta_f/\varepsilon)$ *distribution. Then $\mathcal{M}$ satisfies ε-differential privacy.*

Other kinds of additive noise distributions that can be used in place of Laplace in Theorem 3.3 include Discrete Laplace [23] and Staircase [22].

3.2. **Floating Point Vulnerability.** The most common method to sample a random variable X from a distribution with a known cumulative distribution function (CDF) $F(\cdot)$ is the inverse sampling method: draw a sample U from the uniform distribution on $[0, 1)$ and apply the inverse CDF to obtain $X = F^{-1}(U)$. It's easy to check that the CDF of X is indeed $F(\cdot)$: $\forall x, P(X \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F(x)$. The inverse CDF of Laplace and exponential distributions are particularly simple. Thus most software libraries use this method to sample from these (and many other) distributions.

However, the Laplace and exponential distributions are both continuous over the real numbers. As such, it is not possible to even represent a sample from them on a finite computer, much less to produce one. On one hand, given the non-uniform density of floating-point numbers, a uniform distribution over $[0, 1)$ is not well-defined. On the other hand, floating-point operations involved in applying the inverse CDF will result in missing values and values that appear more frequently than they should [40]. It may seem that such issues are mainly of theoretical interest and do not cause serious harm in practice. Unfortunately, this is not the case: Mironov [40] demonstrated that the textbook implementation of the Laplace Mechanism can lead to catastrophic failures of privacy. In particular, by examining the low-order bits of the noisy output, the noiseless value can often be determined. In one

proof of concept attack, an entire dataset of 18K records was reconstructed with a negligible ($< 10^{-6}$) total privacy budget.

More recently, Casacuberta et al. [10] showed that floating point calculations, unless carefully controlled, can cause algorithms to have much higher sensitivity when implemented with floating-point numbers than if the same algorithm were implemented with real numbers, thereby underestimating the amount of noise needed to protect privacy.

3.3. **Other Distributions.** In this paper, we also make use of the following distributions, whose notation we present here.

The exponential distribution with scale λ/ε, written $\text{Exp}(\lambda/\varepsilon)$, is a probability distribution over nonnegative real numbers and has probability density function $f(x) = \frac{\varepsilon}{\lambda} e^{-\varepsilon x/\lambda}$ and cumulative distribution function $F(x) = 1 - e^{-\varepsilon x/\lambda}$.

The geometric distribution with success probability p, written $\text{Geo}(p)$, is a discrete distribution over the nonnegative integers $0, 1, 2, \ldots$ with probability mass function $P(k) = (1-p)^k p$ and cumulative distribution function $F(k) = 1 - (1-p)^{k+1}$.

Both distributions are memoryless in the sense that if X is distributed either as an exponential or geometric random variable, then for any three nonnegative numbers x, y, z such that $z > y$, we have $P(X \in [x+y, x+z] \mid X \geq x) = P(X \in [y, z])$.

These two distributions are also related by truncation. Let $\lfloor x \rfloor$ denote the largest integer $\leq x$. Let $X \sim \text{Exp}(\lambda/\varepsilon)$ and $Y \sim \text{Geo}(1 - e^{-\varepsilon/\lambda})$. Then comparing their CDFs, we see that for any nonnegative integer k,

$$P(X \leq k+1) = P(\lfloor X \rfloor \leq k) = P(Y \leq k)$$

This means that a sample from an $\text{Exp}(\lambda/\varepsilon)$ random variable X, which can be written as $\lfloor X \rfloor + (X - \lfloor X \rfloor)$, is probabilistically equivalent to the sum of two *independent* random variables $Y + X'$, where $Y \sim \text{Geo}(1 - e^{-\varepsilon/\lambda})$, and X' is a sample from $\text{Exp}(\lambda/\varepsilon)$ conditioned on it being between 0 and 1. This is a special case of Lemma 5.5 (with $\beta = \lambda/\varepsilon$ and $\gamma = 1$).

4. Algorithms

4.1. **The Noisy Top-k with Gap Algorithm.** The basic Noisy Max mechanism [15] and its generalization, Noisy Top-k, are fundamental private selection algorithms that are used to select items from a set of candidates (such as the most common diseases in a population), while controlling the privacy leakage in the underlying data. It takes a list of queries, each with sensitivity[1] 1 (e.g., the count for any disease can change by at most 1 when a person is added to or removed from the data). It then adds noise to each query answer (e.g., the count for each disease) and returns the identity of the query with the k largest noisy answers (e.g., the likely most common diseases).

The preferred instantiation of the Noisy Max mechanism [38, 13] for pure ε-differential privacy adds $\text{Exp}(2/\varepsilon)$ noise, with pdf $f(x) = \frac{\varepsilon}{2} e^{-\varepsilon x/2}$, to each query answer and returns the identity (not value) of the query with the largest noisy answer. The extension to Noisy Top-k changes the noise parameter from $2/\varepsilon$ to $2k/\varepsilon$ and returns the identities of the queries with the top k noisy answers.

[1]Query implementations must be designed with care, as the sensitivity of a query computed using finite data types can be significantly higher than that of the same query evaluated under idealized real number arithmetic [10].

A recently proposed extension, Noisy Top-k with Gap [12, 14], provides numerical information about how much better the selected items are compared to the non-selected items. Its pseudo code is shown in Algorithm 1. There are two major differences between this extension and the Noisy Top-k mechanism: 1) The Noisy Top-k with Gap internally keeps track of the $(k+1)^{\text{th}}$ query (Line 5), and 2) The gaps (i.e., differences in values) between each query in the top k and the corresponding next best query are calculated and returned together with the identities of the top k queries (i.e., the gap between the noisy best and noisy second best queries, the gap between the noisy second and noisy third best queries, etc., are returned), whereas the Noisy Top-k only returns query identities. Remarkably, this extra gap information comes at no additional privacy cost: both mechanisms satisfy pure differential privacy with the exact same privacy parameter ε [12].

Algorithm 1: Noisy Top-k with Gap

1 **function** `GapTopK`$(q_1, \ldots, q_n, k, \varepsilon)$**:**
2 **for** $i = 1, \cdots, n$ **do**
3 $X \leftarrow \text{Exp}(2k/\varepsilon)$
4 $\widetilde{q}_i \leftarrow q_i + X$
5 $j_1, \ldots, j_{k+1} \leftarrow \arg\max_{k+1}(\widetilde{q}_1, \ldots, \widetilde{q}_n)$
6 **for** $i = 1, \ldots, k$ **do**
7 $g_i \leftarrow \widetilde{q}_{j_i} - \widetilde{q}_{j_{i+1}}$
8 **return** $(j_1, g_1), \ldots, (j_k, g_k)$

Theorem 4.1 ([12])**.** *Algorithm 1 satisfies ε-differential privacy.*

Nevertheless, the extra gap information can offer better utility. For example, if an algorithm later asks for noisy answers to those queries that were returned by Noisy Top-k with Gap, these noisy answers can be combined with the gaps to reduce their variance [12, 14]. Hence the gaps can be useful and that is why we focus on a numerically secure implementation of this algorithm.

4.2. **Why the Gap Complicates the Implementation.** There are two common strategies for converting a differentially private mechanism, $\mathcal{M}_{cont}$ with continuous noise, into a discrete algorithm, $\mathcal{M}_{disc}$ that avoids floating point computation, and hence can be implemented without numerical vulnerabilities. The first approach is to directly discretize $\mathcal{M}_{cont}$ and use analytic expressions of its output probabilities to prove differential privacy. This is the approach taken by the discrete Laplace mechanism [23], the discrete Gaussian [9] and the base-2 exponential mechanism [29]. In the case of Noisy Top-k, and especially Noisy Top-k with Gap, the output distributions are complicated infinite summations, making this a less promising direction. The second approach is to design $\mathcal{M}_{disc}$ so that it avoids floating point computation, yet is equivalent to running the ideal mechanism $\mathcal{M}_{cont}$ followed by a deterministic rounding algorithm. Therefore, privacy follows from the postprocessing property of differential privacy. This is the approach we take in the paper. It is a rather straightforward exercise for Noisy Top-k, but becomes non-trivial for Noisy Top-k with Gap. The intuition is that without the gap, Noisy Top-k has categorical outputs (the ids of selected queries) and can be implemented by replacing the exponential distribution with the geometric

distribution, taking advantage of the memorylessness properties of both. However, Noisy Top-k with Gap adds continuous outputs and considers the *pairwise* differences between noisy query answers. The pairwise differences of exponential or geometric random variables no longer have a memorylessness property to take advantage of. To see this more concretely, consider the following lemma, whose proof shows that replacing exponential noise in Noisy Max with geometric noise is enough to make the resulting algorithm equal to the ideal Noisy Max (a subsequent rounding step is not even needed because the output is already discrete).

Lemma 4.2. *Let $q_1, \ldots, q_n$ be integer-valued queries, each with sensitivity 1. Let $\mathcal{M}_{snm}$ be the mechanism such that, on input D, $\mathcal{M}_{snm}(D)$ first computes $x_i \equiv q_i(D) + Y_i$ where $Y_i \sim \text{Geo}(1 - e^{-\varepsilon/2})$, then computes the set of maximums: $S = \{i \mid x_i = \max(x_1, \ldots, x_n)\}$, and returns an element from S uniformly at random. Then $\mathcal{M}_{snm}$ satisfies ε-differential privacy and is probabilistically equivalent to Noisy Max with exponentially distributed noise.*

Proof sketch. The proof sketch is as follows. Consider the mechanism $\mathcal{M}_{exp}$ that is identical to $\mathcal{M}_{snm}$ except that $\mathcal{M}_{exp}$ uses exponential noise instead of geometric. It first computes $x'_i = q_i(D) + X_i$, where $X_i \sim \text{Exp}(2/\varepsilon)$ and returns the i for which x'_i is largest (because the noise distribution is continuous, ties occur with probability 0). Mechanism $\mathcal{M}_{exp}$ is known to satisfy ε-differential privacy [13].

In terms of notation, $\mathcal{M}_{snm}$ uses geometric random variables Y_i and $\mathcal{M}_{exp}$ uses exponential random variables X_i. However, as explained in Section 3.3, $q_1(D) + X_1, \ldots, q_n(D) + X_n$ is probabilistically equivalent to $q_1(D) + Y_1 + X'_1, \ldots, q_n(D) + Y_n + X'_n$, where the X'_i are all identically and independently distributed random variables with support $[0, 1)$. This equivalence is true because the exponential and geometric distributions are memoryless. Thus $\mathcal{M}_{exp}$ can be thought of as computing the same set S as $\mathcal{M}_{snm}$, which is $S = \{i \ : \ q_i(D) + Y_i = \max_j(q_j(D) + Y_j)\}$ and then using the X'_j to break ties. However, since the X'_j are all i.i.d., using them to break ties is equivalent to picking an element of S uniformly at random [13].

Thus $\mathcal{M}_{snm}$ and $\mathcal{M}_{exp}$ have the same exact probabilistic relationship between input and output, except that $\mathcal{M}_{snm}$ only relies on discrete distributions. □

However, a numerically secure version of Noisy Max with Gap (i.e., Noisy Top-k with Gap with $k = 1$) that uses discrete noise will need to provide a discretized gap. Referring back to the notation in the proof sketch for Noisy Max, releasing a discretized gap would involve releasing a discretized version of $q_i(D) + X_i - (q_j(D) + X_j)$, where i is the query for which $q_i(D) + X_i$ is largest and j is the query for which $q_j(D) + X_j$ is second largest. Even if we are lucky and there are no ties, the discretized versions of $q_i(D) + X_i$ and of $q_j(D) + X_j$ do not provide enough information to compute the discretized version of their difference $q_i(D) + X_i - (q_j(D) + X_j)$. In other words, simply replacing the exponential noise with geometric noise in Noisy Max with Gap is not equivalent to rounding the output of Noisy Max with Gap. Our solution to this problem will require randomized rounding routines that rely on random permutations to determine how quantities are rounded. Also, breaking ties will require replacing the exponential distribution with a discrete distribution whose domain needs to be dynamically determined (e.g., instead of being over integers, it may need to be over multiples over some rational number that is not known in advance).

4.3. **Notation and Setup for the Secure Implementation.** To describe the secure implementation, we use the following notation. Let γ_* denote the target resolution (a rational

number), so that all returned gaps are multiples of γ_* (e.g., multiples of 2^{-10}). In particular, γ_* is the reciprocal of an integer, so that all integer-valued query answers are multiples of γ_*. We use γ for other resolutions that are refinements of γ_* (i.e., γ_* is always a multiple of γ). For a positive integer k we use $[k]$ to denote the set of integers from 1 to k: $[k] \triangleq \{1, \ldots, k\}$. We use π denote a permutation on $[k]$, i.e., a bijective function $\pi : [k] \to [k]$. For a real number $x \in \mathbb{R}$, we use $\lfloor x \rfloor$ to denote the floor of x (the largest integer $\leq x$). We use $\lfloor x \rfloor_\gamma$ to denote the largest multiple of γ that is $\leq x$, which can be expressed mathematically as $\lfloor x \rfloor_\gamma \triangleq \lfloor \frac{x}{\gamma} \rfloor \cdot \gamma$. This notation is summarized in Table 1.

Table 1. Notation

Symbol	Meaning
$[k]$	$\{1, \ldots, k\}$
π	permutation on $[k]$ (bijective map $\pi : [k] \to [k]$)
γ_*	target resolution (e.g., 2^{-10}), reciprocal of an integer
γ	other resolutions that are refinements of γ_* (e.g., $2^{-11}, 2^{-12}, \ldots$)
$\lfloor x \rfloor$	floor of x (the largest integer $\leq x$)
$\lfloor x \rfloor_\gamma$	rounding of x down to the nearest multiple of γ, $\lfloor x \rfloor_\gamma = \lfloor \frac{x}{\gamma} \rfloor \cdot \gamma$

The proofs and intermediate steps, that transform algorithms based on continuous noise into algorithms based on rational numbers and discrete noise, use distributions summarized in Table 2. The two main distributions are the exponential and geometric distributions. For consistency we use X (resp. Y) to denote a random variable following the exponential (resp. geometric) distribution. When one forms the corresponding conditional distributions, conditioned on the random variable being less than some threshold τ, the result is a truncated distribution. In the case of exponential and geometric distributions, the truncated versions are the same as taking the original random variable modulo the threshold τ. We use X' (resp. Y') to denote a random variable following the *truncated* exponential (resp. *truncated* geometric) distribution. We use Z_γ to denote a (γ-) scaled geometric random variable, which is the same as a geometric random variable multiplied by γ, and so its domain is over integer multiples of γ.

Table 2. Noise Distributions

Distribution	Symbol	Support	Density/Mass
Exponential	$X \sim \text{Exp}(\beta)$	$[0, \infty)$	$\frac{1}{\beta} e^{-\frac{x}{\beta}}$
Truncated exponential	$X' \sim \text{Exp}(\beta) \bmod \gamma$	$[0, \gamma)$	$\frac{1}{\beta} e^{-\frac{x}{\beta}} / (1 - e^{-\frac{\gamma}{\beta}})$
Geometric	$Y \sim \text{Geo}(p)$	$\{0, 1, 2, \ldots\}$	$p(1-p)^m$
Truncated geometric	$Y' \sim \text{Geo}(p) \bmod M$	$\{0, 1, \ldots, M-1\}$	$\frac{p(1-p)^m}{1-(1-p)^M}$
Scaled geometric	$Z_\gamma \sim \gamma \cdot \text{Geo}(p)$	$\{0, \gamma, 2\gamma, \ldots\}$	$p(1-p)^m$

4.4. **Model of Computation.** We adopt the standard w-bit word RAM model of computation, consistent with prior literature [4, 9]. In this model, memory accesses and basic operations on w-bit words, including arithmetic, comparisons, and logical operations, are assumed to execute in constant time. Our algorithms involve parameters that are rational numbers, which we represent as pairs of integers. We assume that these pairs can be stored using a constant number of words (e.g., if a Big Integer package is used, the constant number of words is roughly equivalent to the size of the RAM divided by the number of queries). It is important to note that all operations within our secure algorithm are performed over rational numbers, relying solely on basic integer operations and comparisons. While some parameters may theoretically involve irrational expressions (e.g. $e^{-\frac{s}{t}}$ where $s, t \in \mathbb{N}$), we do not evaluate such expressions in practice, thereby avoiding any need for floating-point approximations. Furthermore, our algorithms require access to randomness, which we model as a source of independent, uniformly distributed random bits.

The runtime of our algorithms is inherently randomized; however, we will later demonstrate that the expected runtime is $O(n^2 \log n)$ where n is the number of queries. The expected memory usage of our algorithms is $O(n^2)$.

Finally, we note that, technically speaking, our algorithm only satisfies (ε, δ)-differential privacy because it is possible that a noise value becomes too large to fit in memory and disk, or a computation whose running time is a random variable with small finite expected running time could extend beyond the physical life of the hardware. Addressing such challenges is an engineering issue that we ignore.

4.5. **Secure Primitives for the Implementation.** We use the following sampling primitives to implement our algorithms. These sampling algorithms do not use floating-point operations and are thus free of the floating-point vulnerability. The fundamental assumption is that there is access to a sequence of independent, uniformly random bits.

Random integer generation in an interval. Given a sequence of uniformly random bits, one can generate random integers in a finite range, e.g. $0, 1, \ldots, k-1$, using rejection sampling [43, 19, 42, 33].

The Fisher-Yates random shuffle. Another commonly used technique is the Fisher-Yates shuffle [18, 32]. It randomly permutes a list of n elements in place so that all $n!$ possible permutations are equally probable. This algorithm is based on uniform sampling of integers.

Sampling from a Bernoulli distribution. For certain values of success probability p (even irrational values), it is possible to reduce sampling from Bernoulli(p) to sampling random integers without any use of floating point. First, when $p = n/d$ is a rational number, it suffices to draw an integer N uniformly from the range $[0, d)$ and output 1 if $N < n$ and 0 otherwise. When $p = e^{-\theta}$ for some positive rational number $\theta \in \mathbb{Q}^+$, Canonne et al. [9] reduced the task of sampling from Bernoulli($e^{-\theta}$) to that of sampling from Bernoulli(θ/k) for various integers $k \geq 1$ (see Algorithm 1 in [9]), without the need for computing $e^{-\theta}$. Notably, this sampling algorithm requires only uniform sampling over integers and basic integer arithmetic, and its runtime is constant in expectation [9].

Sampling from a geometric distribution. Canonne et al. [9] also showed that for certain values of success probability p, namely those in the form of $1 - e^{-\theta}$ where θ is a positive rational number, sampling from a Geo$(1 - e^{-\theta})$ distribution can be efficiently reduced to sampling from Bernoulli$(e^{-\theta'})$, where θ' is a rational number that depends on θ. Canonne et. al sample a geometric distribution inside of a discrete Laplace sampling algorithm (Algorithm 2 in [9]), so for completeness, we extract the geometric sampler from their algorithm and list it in the appendix. Like before, this algorithm completely avoids use of floating point numbers, and its runtime is constant in expectation [9].

4.6. **A Secure Implementation of Noisy Top-k with Gap.** A common approach to making a differentially private algorithm secure on finite machines is to take the ideal algorithm $\mathcal{M}$ (Algorithm 1 in our case) that assumes infinite precision and define an intermediate "rounded" version $\mathcal{M}'$ that differs from $\mathcal{M}$ in only two ways. First, the ideal (real) inputs to $\mathcal{M}'$ are rounded to rational numbers, which can be represented as pairs of integers. Then $\mathcal{M}'$ rounds the ideal output of $\mathcal{M}$ to rational numbers. The final step is to create a secure algorithm $\mathcal{M}''$ that completely avoids irrational numbers in its intermediate state while still being probabilistically equivalent to $\mathcal{M}'$ – in other words, for every dataset D, the output distributions $P(\mathcal{M}'(D))$ and $P(\mathcal{M}''(D))$ are the same. Following this approach, we first create a rounded-input/output version of Algorithm 1 by rounding the inputs and each gap in the output down to its nearest multiple of γ_*. Thus $\mathcal{M}'$ from the above discussion is the following Algorithm 2 (note that the input rounding happens on Line 4). Then, to create the final secure algorithm $\mathcal{M}''$, we proceed with two steps. First, we create another intermediate algorithm (Algorithm 3) that has both continuous random variables and discrete random variables that are derived from the continuous ones. Last we create $\mathcal{M}''$ (Algorithm 4) by removing the continuous random variables, while keeping the joint distribution of the discrete random variables unchanged.

Algorithm 2: Noisy Top-k with Gap (Rounded)

input : $q_1, \ldots, q_n$: answers to sensitivity 1 queries.
γ_* is the target resolution; 1 is a multiple of γ_*
ε is rational

1 **function** `GapTopK`$(q_1, \ldots, q_n, k, \varepsilon)$**:**
2 **for** $i = 1, \cdots, n$ **do**
3 $X \leftarrow \text{Exp}(2k/\varepsilon)$
4 $\widetilde{q}_i \leftarrow \lfloor q_i \rfloor_{\gamma_*} + X$
5 $j_1, \ldots, j_{k+1} \leftarrow \arg\max_{k+1}(\widetilde{q}_1, \ldots, \widetilde{q}_n)$
6 **for** $i = 1, \ldots, k$ **do**
7 $g_i \leftarrow \lfloor \widetilde{q}_{j_i} - \widetilde{q}_{j_{i+1}} \rfloor_{\gamma_*}$
8 **return** $(j_1, g_1), \ldots, (j_k, g_k)$

To show that Algorithm 2 is differentially private, we first show that rounding down does not affect the sensitivity.

Lemma 4.3. *Suppose q is a scalar function with ℓ_1-sensitivity Δ. Suppose that γ is a positive number and that Δ is an integer multiple of γ. Then $\lfloor q \rfloor_\gamma$ also has ℓ_1-sensitivity Δ.*

Proof. Let D and D' be two neighboring datasets that differ on one person. Without loss of generality, assume $q(D) \geq q(D')$. Then $0 \leq q(D) - q(D') \leq \Delta$ which means there exists a nonnegative integer k with $k \leq \Delta/\gamma$ and a number $s \in [0, \gamma)$ such that $q(D)-q(D')+k\gamma+s = \Delta$ and so $q(D)-\Delta+k\gamma+s = q(D')$. Let $\text{rem}(q(D)) = q(D)-\lfloor q(D)\rfloor_\gamma$. Then

$$\lfloor q(D)\rfloor_\gamma - \Delta + k\gamma + \text{rem}(q(D)) + s = \lfloor q(D')\rfloor_\gamma + \text{rem}(q(D'))$$

Noting that $\text{rem}(q(D)) + s \in [0, 2\gamma)$, this means

$$\lfloor q(D)\rfloor_\gamma - \Delta + k'\gamma + s' = \lfloor q(D')\rfloor_\gamma + \text{rem}(q(D'))$$

where $k' \in \{k, k+1\}$ and $s' \in [0, \gamma)$. This means that $s' = \text{rem}(q(D'))$ as both quantities are nonnegative and less than γ, while everything else is a nonnegative multiple of γ. Rounding both sides down to the nearest multiple of γ, we get

$$\lfloor q(D)\rfloor_\gamma - \Delta + k'\gamma = \lfloor q(D')\rfloor_\gamma$$

where k' is an integer between 0 and $\Delta/\gamma + 1$. Thus

$$\lfloor q(D)\rfloor_\gamma - \lfloor q(D')\rfloor_\gamma = \Delta - k'\gamma \in [-\gamma, \Delta] \subseteq [-\Delta, \Delta]$$

since Δ is a multiple of γ. □

Theorem 4.4. *Algorithm 2 is ε-differentially private.*

Proof. Algorithm 2 is simply Algorithm 1 applied to sensitivity 1 queries $\lfloor q_i(D)\rfloor_{\gamma_*}$ followed by a post-processing step (which therefore does not affect privacy parameters) of rounding every gap down to the closest multiple of γ_*. □

Thus, from now on, we assume that the query answers are integer multiples of γ_*. Next, to obtain a secure implementation of Algorithm 2, we need to create an algorithm whose internal states are machine representable (i.e., rational numbers), and whose output distribution is identical to that of Algorithm 2. This is challenging because as pointed out previously, any straightforward discretization of the noisy query answers $\widetilde{q}_i$ (e.g., $\lfloor \widetilde{q}_i \rfloor_{\gamma_*}$) will cause the probability of ties to be nonzero. Furthermore, it can happen that $\lfloor \widetilde{q}_i - \widetilde{q}_j \rfloor_{\gamma_*} \neq \lfloor \widetilde{q}_i \rfloor_{\gamma_*} - \lfloor \widetilde{q}_j \rfloor_{\gamma_*}$ so the discretized gaps are not readily available from the discretized noisy queries. We solve this problem by a two-step approach. The approach is described here and proofs are presented in Section 5.

The first step is to create an intermediate algorithm (Algorithm 3) that has both continuous random variables and discrete random variables that are derived from them. The last step (Algorithm 4) will remove the continuous random variables, while keeping the joint distribution of the discrete random variables unchanged.

In the intermediate Algorithm 3, discrete variables $\widehat{q}_i^{(0)}$ are initially introduced by discretizing the continuous noisy query answers $\widetilde{q}_i$ (Line 4). These discrete variables are used for the decision-making (i.e., determining the winning queries) in Line 5. Because of the possibility of ties in discrete random variables, several new features are introduced by Algorithm 3.

- First, we get the top $k+2$ noisy query answers instead of $k+1$ answers. This is done so that we know if there are ties in the $k+1^{\text{th}}$ place. If there are ties anywhere in the top $k+1$ places, we will need to break them using the second feature of Algorithm 3, described next.

Algorithm 3: Noisy Top-k with Gap (Intermediate)

input : γ_* is the target resolution; 1 is a multiple of γ_*
$q_1, \ldots, q_n$: answers to sensitivity 1 queries, all are integer multiples of γ_*.
ε is rational
M is an integer used to refine the resolution.

1 **function** `GapTopK`$(q_1, \ldots, q_n, k, \varepsilon)$**:**
2 **for** $i = 1, \cdots, n$ **do** `// add noise to query answers`
3 $X \leftarrow \mathrm{Exp}(2k/\varepsilon)$
4 $\widetilde{q}_i \leftarrow q_i + X, \quad \widehat{q}_i^{(0)} \leftarrow \lfloor \widetilde{q}_i \rfloor_{\gamma_*}$
5 $j_1, \ldots, j_{k+2} \leftarrow \arg\max_{k+2}(\widehat{q}_1^{(0)}, \ldots, \widehat{q}_n^{(0)})$ `// choose top queries`
6 $t \leftarrow 0, \quad \gamma_0 \leftarrow \gamma_*$
7 **while** *there is a tie among* $\widehat{q}_{j_1}^{(t)}, \ldots, \widehat{q}_{j_{k+2}}^{(t)}$ **do** `// break ties`
8 $t \leftarrow t + 1, \quad \gamma_t \leftarrow \frac{1}{M}\gamma_{t-1}$
9 **for** $i = 1, \ldots, n$ **do**
10 $\widehat{q}_i^{(t)} \leftarrow \lfloor \widetilde{q}_i \rfloor_{\gamma_t}$
11 $j_1, \ldots, j_{k+2} \leftarrow \arg\max_{k+2}(\widehat{q}_1^{(t)}, \ldots, \widehat{q}_n^{(t)})$
12 **for** $i = 1, \ldots, k$ **do** `// compute gaps`
13 $g_i \leftarrow \lfloor \widetilde{q}_{j_i} - \widetilde{q}_{j_{i+1}} \rfloor_{\gamma_*}$
14 **return** $(j_1, g_1), \ldots, (j_k, g_k)$

- Next, we introduce a tie-breaking loop (Line 7) which keeps going until all ties in the top $k+1$ places are resolved. If there is a tie between any two such queries, the algorithm increases the precision of rounding – instead of rounding the continuous noisy answers down to a multiple of the target resolution γ_*, the algorithm revisits those noisy answers and rounds them down to a multiple of γ_*/M (where the integer $M > 1$ is an algorithm parameter that is 10 in our experiments). This results in the next version of the discrete random variables $\widehat{q}_i^{(1)}$. This finer precision may break the ties, and if not, the precision is refined again until eventually all ties are broken. Later, Algorithm 4 will simulate this loop using only discrete random variables.

After the distinct top $k+1$ queries are determined, the gaps are computed. However, for now, the discretized gaps are still computed by rounding the gaps computed from continuous noisy answers (this problem will be solved by Algorithm 4).

The next algorithm, Algorithm 4 switches to only using discrete random variables, so that all internal states are rational numbers. This involves removing the exponential random variables that were used by Algorithm 3, simulating the tie-breaking loop using only discrete random variables, and finally simulating the gap using only the discrete random variables. We handle the difficulties as follows:

- Since the continuous random variable $\widetilde{q}_i$ (query answer plus exponential noise) is rounded down to a multiple of γ_* to get $\widehat{q}_i^{(0)}$, the distribution of the latter is the same as the query answer plus geometric noise over multiples of γ_* (see Section 3.3). These are represented as the random variables $\overline{q}_i^{(0)}$ in Line 4 of Algorithm 4.

Algorithm 4: Noisy Top-k with Gap (Secure)

input : γ_* is the target resolution; 1 is a multiple of γ_*
$q_1, \ldots, q_n$: answers to sensitivity 1 queries, all are integer multiples of γ_*.
ε is rational
M is an integer used to refine the resolution.

1 **function** `GapTopK`$(q_1, \ldots, q_n, k, \varepsilon)$**:**
2 **for** $i = 1, \cdots, n$ **do** `// add noise to query answers`
3 $Y_0 \leftarrow \mathrm{Geo}(1 - e^{-\varepsilon\gamma_*/2k})$
4 $\breve{q}_i^{(0)} \leftarrow q_i + \gamma_* \cdot Y_0$
5 $j_1, \ldots, j_{k+2} \leftarrow \arg\max_{k+2}(\breve{q}_0^{(0)}, \ldots, \breve{q}_n^{(0)})$ `// choose top queries`
6 $t \leftarrow 0, \quad \gamma_0 \leftarrow \gamma_*$ `// Loop 1`
7 **while** *there is a tie among* $\breve{q}_{j_1}^{(t)}, \ldots, \breve{q}_{j_{k+2}}^{(t)}$ **do** `// break ties`
8 $t \leftarrow t + 1, \quad \gamma_t \leftarrow \frac{1}{M}\gamma_{t-1}$
9 **for** $i = 1, \ldots, n$ **do**
10 $Y_t \leftarrow \mathrm{Geo}(1 - e^{-\varepsilon\gamma_t/2k})$
11 $\breve{q}_i^{(t)} \leftarrow \breve{q}_i^{(t-1)} + \gamma_t \cdot (Y_t \bmod M)$
12 $j_1, \ldots, j_{k+2} \leftarrow \arg\max_{k+2}(\breve{q}_1^{(t)}, \ldots, \breve{q}_n^{(t)})$
13 $x_1, \ldots, x_{k+1} \leftarrow$ `Shuffle`$(1, \ldots, k+1)$ `// Fisher-Yates shuffle`
14 **for** $i = 1, \ldots, k$ **do** `// compute gaps`
15 **if** $x_i < x_{i+1}$ **then**
16 $g_i \leftarrow \lfloor \breve{q}_{j_i}^{(t)} - \breve{q}_{j_{i+1}}^{(t)} - \gamma_t \rfloor_{\gamma_*}$
17 **else**
18 $g_i \leftarrow \lfloor \breve{q}_{j_i}^{(t)} - \breve{q}_{j_{i+1}}^{(t)} \rfloor_{\gamma_*}$
19 **return** $(j_1, g_1), \ldots, (j_k, g_k)$

- The discrete variable $\widehat{q}_i^{(j)} = \lfloor \widetilde{q}_i \rfloor_{\gamma_*/M^j}$ is equal to the continuous variable $\widetilde{q}_i$ rounded down to a multiple of γ_*/M^j. But this also means that $\widehat{q}_i^{(j)}$ is equal to $\widehat{q}_i^{(j+1)}$ rounded down to a multiple of γ_*/M^j (i.e., $\widehat{q}_i^{(j)} = \lfloor \widehat{q}_i^{(j+1)} \rfloor_{\gamma_*/M^j}$). This allows us to use a modified version of the memoryless property of geometric distributions to simulate $\widehat{q}_i^{(j+1)}$ from $\widehat{q}_i^{(j)}$ and their joint distribution would be the same as if they were derived from the continuous random variable. Namely the conditional distribution of the part that was rounded down, which is $\widehat{q}_i^{(j+1)} - \widehat{q}_i^{(j)} = \widehat{q}_i^{(j+1)} - \lfloor \widehat{q}_i^{(j+1)} \rfloor_{\gamma_*/M^j}$, conditioned on $\widehat{q}_i^{(j)}$, is a truncated geometric random variable in the interval $[0, \gamma_*/M^j)$ with support equal to integer multiples of γ_*/M^{j+1}. In Algorithm 4, these variables are called $\breve{q}_i^{(j+1)}$ and $\breve{q}_i^{(j)}$ and the generation of the former from the latter using truncated geometric noise is performed in Lines 10 and 11.
- Next we consider the intuition behind how Algorithm 4 simulates the discretized gap from these discrete random variables. In general, for any two numbers $X_i, X_j \in \mathbb{R}$ and resolution γ, the quantities $\lfloor X_i - X_j \rfloor_\gamma$ and $\lfloor X_i \rfloor_\gamma - \lfloor X_j \rfloor_\gamma$ are not equal, but they differ by a correction term that we can simulate. Specifically, we use this fact from Lemma 5.8

in Section 5.

$$\lfloor X_i - X_j \rfloor_\gamma = \lfloor X_i \rfloor_\gamma - \lfloor X_j \rfloor_\gamma - \delta_{ij}\gamma \quad \text{where } \delta_{ij} = \begin{cases} 0 & \text{if } X_i - \lfloor X_i \rfloor_\gamma \geq X_j - \lfloor X_j \rfloor_\gamma \\ 1 & \text{otherwise} \end{cases}$$

We use this fact as follows. In the notation of intermediate Algorithm 3, X_i is the same as the continuous noisy answer $\widetilde{q}_i$. Given a resolution γ, the values $\widetilde{q}_i - \lfloor \widetilde{q}_i \rfloor_\gamma$ are, due to the memoryless property, truncated exponentials supported on $[0, \gamma)$ and they are i.i.d. for all i. Therefore, the ordering of the $\widetilde{q}_i - \lfloor \widetilde{q}_i \rfloor_\gamma$ values, for all i, is uniformly random. Hence, Line 13 in Algorithm 4 determines this ordering with a random permutation and using that, it determines how to simulate the discretized gap correction term.

Now note that Algorithm 4 only uses rational numbers ($1/\gamma_*$ is an integer and ε must be rational) and discrete distributions (the parameter of the geometric distribution is not explicitly computed). The randomness primitives are:

- Generating an exact sample from a geometric distribution with parameter $(1 - e^{-\varepsilon\gamma_*/2k})$ can be done using the algorithm from Canonne et al. [9] (reproduced in the appendix here) without ever computing $e^{-\varepsilon\gamma_*/2k}$.
- Generating a truncated geometric distribution can be performed by generating a geometric distribution and computing the result modulo a number as in Line 11.
- A random permutation can be generated from the Fischer-Yates shuffle algorithm [18] and only requires choosing a random integer from a bounded range, which can also be performed exactly given a source of uniform bits [9].

4.7. **Further Improvements.** Algorithm 4 can be further improved for efficiency. Note that if $\lfloor X_i \rfloor_{\gamma_t} < \lfloor X_j \rfloor_{\gamma_t}$, then for any refinement $\gamma_{t'} < \gamma_t$ (γ_t is a multiple of $\gamma_{t'}$), we also have $\lfloor X_i \rfloor_{\gamma_{t'}} < \lfloor X_j \rfloor_{\gamma_{t'}}$. Thus if at some resolution γ_t, it can be determined that a query q_i is not among the top candidates that contribute to the output indexes and gaps, then this query can be dropped. Thus we can maintain a pool of queries that are relevant to the output of the algorithm (i.e., those whose noisy answers at the current resolution are tied with something in the top $k+1$) and only refine the noise to those queries. Initially, the pool will have all n queries. But after the first iteration, the pool will have only roughly $O(k)$ queries left, making subsequent iterations much faster if $k \ll n$. We call this optimization *early query pruning* and report its performance in Section 6.

4.8. **Side Channels.** Due to the inherent randomness in the tie-breaking loop (Line 6 to Line 12), both the runtime and memory consumption of Algorithm 4 are stochastic. This introduces two potential concerns: (1) prolonged execution time may inadvertently reveal the presence of ties among the top queries, and (2) the algorithm may exceed available memory resources before completion. To mitigate these concerns, we first derive bounds on the expected runtime and memory usage of Algorithm 4.

Theorem 4.5. *Let* $\theta = 1 - \frac{(1-e^{-\varepsilon\gamma_*/2k})(1+e^{-\varepsilon\gamma_* M/2k})}{(1+e^{-\varepsilon\gamma_*/2k})(1-e^{-\varepsilon\gamma_* M/2k})}$. *The expected runtime of Algorithm 4 is* $O(n^2 \log n/\theta)$. *The expected memory consumption is* $O(n^2/\theta)$.

We emphasize that the above bounds are derived using the upper bound of n/θ on the expected number of iterations of the tie-breaking loop. This upper bound is relatively loose;

in practice, the actual number of iterations is significantly lower, as demonstrated by our empirical evaluation.

Consequently, the timing channel can be mitigated by enforcing a fixed minimum number of iterations $T_{\min} = O(n/\theta)$. Although this approach does not entirely eliminate the timing channel, it significantly reduces the likelihood of inferring the presence of ties. By increasing $T_{\min}$, the probability of successfully exploiting the timing channel can be made arbitrarily small.

To address the potential issue of Algorithm 4 exhausting available memory, we may impose an upper limit $T_{\max}$ on the number of iterations of the tie-breaking loop. However, this modification results in the algorithm satisfying only approximate (ε, δ)-differential privacy, where δ corresponds to the probability that the iteration limit is reached. Given that the number of iterations follows an exponential distribution (as shown in the proof of Theorem 4.5), this probability δ is negligibly small in practical scenarios.

5. Proofs

In this section, we prove the privacy guarantees of Algorithm 3 and Algorithm 4 by showing that they are equivalent to Algorithm 2.

Definition 5.1. *We say that two algorithms $\mathcal{M}_1$ and $\mathcal{M}_2$ are equivalent if for all datasets D and all measurable sets S, $P(\mathcal{M}_1(D) \in S) = P(\mathcal{M}_2(D) \in S)$.*

Theorem 5.2. *Algorithm 3 is equivalent to Algorithm 2 and therefore is ε-differentially private.*

Proof of Theorem 5.2. First, we show that Algorithm 3 terminates with probability 1. Since $\widetilde{q}_1, \ldots, \widetilde{q}_n$ uses continuous noise, with probability 1 there is no tie among $\widetilde{q}_1, \ldots, \widetilde{q}_n$. Therefore, let $\delta = \min_{i \neq j} |\widetilde{q}_i - \widetilde{q}_j|$, then we have $\delta > 0$ with probability 1. Thus when $\gamma_t = \gamma_* / M^t \leq \delta$, i.e. $t \geq \log_M(\frac{\gamma_*}{\delta})$, we have no ties among $\widehat{q}_1^{(t)}, \ldots, \widehat{q}_n^{(t)}$.

Next, we show that when the loop terminates, the indices $j_1, \ldots, j_{k+1}$ are indeed from the top $k+1$ noisy queries. Note that $\lfloor x \rfloor_\gamma > \lfloor y \rfloor_\gamma \implies x > y$. By the termination condition, at the end of the loop we have that $\widehat{q}_{j_1} > \ldots > \widehat{q}_{j_{k+1}} > \widehat{q}_{j_{k+2}} \geq \widehat{q}_s, s \notin \{j_1, \ldots, j_{k+2}\}$. Therefore, we have $\widetilde{q}_{j_1} > \ldots > \widetilde{q}_{j_{k+1}} > \widetilde{q}_s, s \notin \{j_1, \ldots, j_{k+1}\}$. Thus $j_1, \ldots, j_{k+1}$ are the indices of the largest $k+1$ queries among $\widetilde{q}_1, \ldots, \widetilde{q}_n$. This means if $\widetilde{q}_1, \ldots, \widetilde{q}_n$ are the same in Algorithm 2 and Algorithms 3, then the indices $j_1, \ldots, j_{k+1}$ are the same. Since Algorithm 2 and Algorithm 3 only differ in how they choose the top $k+1$ noisy queries and the winning queries are the same, the two algorithms are equivalent. □

Theorem 5.3. *Assume there is access to a sequence of independent, uniformly random bits. Then Algorithm 4 is equivalent to Algorithm 3 and therefore is ε-differentially private.*

This proof contains multiple stages and requires several supporting results. To establish the equivalence between Algorithm 3 and Algorithm 4, we need a few lemmas to establish the connection between successive roundings of an exponential random variables (with increasing resolutions) and a sequence of independent geometric random variables.

Lemma 5.4. *Let $X \sim \text{Exp}(\beta)$. Then the distribution of $Z_\gamma \triangleq \lfloor X \rfloor_\gamma$ is the scaled geometric distribution over $\{0, \gamma, 2\gamma, \ldots\}$ with success probability $p = 1 - e^{-\frac{\gamma}{\beta}}$. In other words, $\lfloor X \rfloor_\gamma$ follows the same distribution as $\gamma \cdot Y$ where Y geometric distribution (over $\{0, 1, 2, ...\}$) with success probability $p = 1 - e^{-\frac{\gamma}{\beta}}$.*

Proof. For any value $m \in \{0, 1, \ldots\}$ we have

$$P(\lfloor X \rfloor_\gamma = m\gamma) = P(m\gamma \le X < m\gamma + \gamma) = \int_{m\gamma}^{m\gamma+\gamma} \frac{1}{\beta} e^{-\frac{x}{\beta}}\, dx = -e^{-\frac{x}{\beta}}\Big|_{m\gamma}^{m\gamma+\gamma}$$
$$= e^{-\frac{m\gamma}{\beta}} - e^{-\frac{m\gamma+\gamma}{\beta}} = e^{-\frac{m\gamma}{\beta}}(1 - e^{-\frac{\gamma}{\beta}}) = (1-p)^m p$$

where we let $p = 1 - e^{-\frac{\gamma}{\beta}}$ and hence $(1-p)^m = e^{-\frac{m\gamma}{\beta}}$. □

Lemma 5.5. *Let $X \sim \mathrm{Exp}(\beta)$ and $X' = X - \lfloor X \rfloor_\gamma$. Then X' follows the truncated exponential distribution on $[0, \gamma)$. Furthermore, $P(X' \mid \lfloor X \rfloor_\gamma) = P(X')$, i.e., X' and $\lfloor X \rfloor_\gamma$ are independent. Therefore, X is probabilistically equivalent to the sum of two independent random variables $Z_\gamma + X'$ where $Z_\gamma \sim \gamma \cdot \mathrm{Geo}(1 - e^{-\frac{\gamma}{\beta}})$ and $X' \sim \mathrm{Exp}(\beta) \bmod \gamma$.*

Proof. By definition of $\lfloor X \rfloor_\gamma$ we have $X' \in [0, \gamma)$. For any $x \in [0, \gamma)$,

$$X' \le x \implies X - \lfloor X \rfloor_\gamma \le x \implies X \in [l\gamma, l\gamma + x], l = 0, 1, \ldots.$$

Thus

$$P(X' \le x) = \sum_{l=0}^{\infty} P(l\gamma \le X < l\gamma + x) = \sum_{l=0}^{\infty} \int_{l\gamma}^{l\gamma+x} \frac{1}{\beta} e^{-\frac{s}{\beta}}\, ds$$
$$= \sum_{l=0}^{\infty} e^{-\frac{l\gamma}{\beta}}(1 - e^{-\frac{x}{\beta}}) = (1 - e^{-\frac{x}{\beta}}) \sum_{l=0}^{\infty} e^{-\frac{l\gamma}{\beta}} = \frac{1 - e^{-\frac{x}{\beta}}}{1 - e^{-\frac{\gamma}{\beta}}}$$

Hence the density

$$f(X' = x) = \frac{d}{dx} P(X' \le x) = \frac{\frac{1}{\beta} e^{-\frac{x}{\beta}}}{1 - e^{-\frac{\gamma}{\beta}}} \cdot \mathbf{1}_{[0,\gamma)}.$$

Therefore, $X' \sim \mathrm{Exp}(\beta) \bmod \gamma$. From Lemma 5.4, $\lfloor X \rfloor_\gamma \sim \gamma \cdot \mathrm{Geo}(1 - e^{-\frac{\gamma}{\beta}})$. The independence of X' and $\lfloor X \rfloor_\gamma$ is due to the memoryless property of exponential distribution:

$$P(X' \le x \mid \lfloor X \rfloor_\gamma = l\gamma) = \frac{P(l\gamma \le X \le l\gamma + x)}{P(l\gamma \le X \le l\gamma + \gamma)} = \frac{e^{-\frac{l\gamma}{\beta}} - e^{-\frac{l\gamma+x}{\beta}}}{e^{-\frac{l\gamma}{\beta}} - e^{-\frac{l\gamma+\gamma}{\beta}}} = \frac{1 - e^{-\frac{x}{\beta}}}{1 - e^{-\frac{\gamma}{\beta}}} = P(X' \le x).$$

Hence, $X = \lfloor X \rfloor_\gamma + X'$ has the same distribution as the sum of two independent random variables, one following a scaled geometric distribution, and the other following a truncated exponential distribution. □

Lemma 5.6. *Let $Y \sim \mathrm{Geo}(p)$ and $Y' = Y \bmod M$ for some $M > 1$. Then the distribution of Y' is the truncated geometric distribution with success probability p on $\{0, \ldots, M-1\}$.*

Proof. For any value $m \in \{0, \ldots, M-1\}$

$$P(Y' = m) = \sum_{l=0}^{\infty} P(Y = lM + m) = \sum_{l=0}^{\infty} (1-p)^{lM+m} p = (1-p)^m p \sum_{l=0}^{\infty} (1-p)^{lM}$$
$$= \frac{(1-p)^m p}{1 - (1-p)^M}$$

□

Lemma 5.7. *Let M be a positive integer. Let $\gamma_1, \gamma_2 > 0$ be such that $\gamma_1 = M\gamma_2$. Let $X \sim \text{Exp}(\beta)$, $Z_{\gamma_1} = \lfloor X \rfloor_{\gamma_1}$ and $Z_{\gamma_2} = \lfloor X \rfloor_{\gamma_2}$. Then $Z_{\gamma_1} = \lfloor Z_{\gamma_2} \rfloor_{\gamma_1}$ and $P(Z_{\gamma_2} \mid Z_{\gamma_1})$ is the same as the probability mass function of $Z_{\gamma_1} + \gamma_2(Y \bmod M)$ where $Y \sim \text{Geo}(1 - e^{-\frac{\gamma_2}{\beta}})$ is independent of Z_{γ_1}.*

Proof. First we show $Z_{\gamma_1} = \lfloor Z_{\gamma_2} \rfloor_{\gamma_1}$. Intuitively, this means that the effect of rounding down a number to a finer resolution (γ_2) first then rounding down the result again to a coarser resolution (γ_1) is the same as rounding the number directly to the coarser resolution (γ_1). Let $Z_{\gamma_1} = n\gamma_1$, then $X = n\gamma_1 + s$ for some $s \in [0, \gamma_1)$. Thus we have $Z_{\gamma_2} = \lfloor X \rfloor_{\gamma_2} = n\gamma_1 + \lfloor s \rfloor_{\gamma_2}$ because $n\gamma_1$ is already a multiple of γ_2. Since $\lfloor s \rfloor_{\gamma_2} \leq s < \gamma_1$, we have $\lfloor \lfloor s \rfloor_{\gamma_2} \rfloor_{\gamma_1} = 0$ and $\lfloor Z_{\gamma_2} \rfloor_{\gamma_1} = n\gamma_1 = Z_{\gamma_1}$. Moreover, for $m \in \{0, \ldots, M-1\}$:

$$
\begin{aligned}
&P(Z_{\gamma_2} = n\gamma_1 + m\gamma_2 \mid Z_{\gamma_1} = n\gamma_1) \\
&= P(n\gamma_1 + m\gamma_2 \leq X < n\gamma_1 + m\gamma_2 + \gamma_2 \mid n\gamma_1 \leq X < n\gamma_1 + \gamma_1) \\
&= \frac{\int_{n\gamma_1 + m\gamma_2}^{n\gamma_1 + m\gamma_2 + \gamma_2} \frac{1}{\beta} e^{-\frac{s}{\beta}}\, ds}{\int_{n\gamma_1}^{n\gamma_1 + \gamma_1} \frac{1}{\beta} e^{-\frac{s}{\beta}}\, ds} = \frac{e^{-\frac{(n\gamma_1 + m\gamma_2)}{\beta}}(1 - e^{-\frac{\gamma_2}{\beta}})}{e^{-\frac{n\gamma_1}{\beta}}(1 - e^{-\frac{\gamma_1}{\beta}})} \\
&= \frac{(1 - e^{-\frac{\gamma_2}{\beta}})}{(1 - e^{-\frac{\gamma_1}{\beta}})} \cdot e^{-\frac{m\gamma_2}{\beta}} \quad (\text{let } p = 1 - e^{-\frac{\gamma_2}{\beta}}) \\
&= \frac{p(1-p)^m}{1 - (1-p)^M}
\end{aligned}
$$

Thus from Lemma 5.6, this is the probability mass function of a truncated geometric distribution and is independent of the value of $n\gamma_1$. Since Z_{γ_1} is a rounded down version of Z_{γ_2}, this also means that the distribution of $Z_{\gamma_2} - Z_{\gamma_1}$ is this same truncated geometric distribution. Thus we have $Z_{\gamma_2} = Z_{\gamma_1} + \gamma_2(Y \bmod M)$ where Y is $\text{Geo}(1 - e^{-\frac{\gamma_2}{\beta}})$. □

Lemma 5.8. *For any two numbers X_1, X_2, the following is true:*

$$
\lfloor X_i - X_j \rfloor_{\gamma_t} = \lfloor X_i \rfloor_{\gamma_t} - \lfloor X_j \rfloor_{\gamma} - \begin{cases} 0 & \text{if } X_i - \lfloor X_i \rfloor_{\gamma_t} \geq X_j - \lfloor X_j \rfloor_{\gamma_t} \\ \gamma_t & \text{otherwise} \end{cases}
$$

Furthermore, if $X_1, \ldots, X_n$ are i.i.d. exponential random variables and if π is a random permutation of $1, \ldots, n$ chosen uniformly at random, then the joint distribution of all the comparisons $X_i - \lfloor X_i \rfloor_{\gamma_t} \geq X_j - \lfloor X_j \rfloor_{\gamma_t}$ for all i, j conditioned on knowledge of all of the $\lfloor X_i \rfloor_{\gamma_t}$ for all i is the same as the joint distribution of the comparisons $\pi(i) \geq \pi(j)$ for all i, j.

Proof. First, $X_i = \lfloor X_i \rfloor_{\gamma_t} + (X_i - \lfloor X_i \rfloor_{\gamma_t})$ so

$$
\begin{aligned}
\lfloor X_i - X_j \rfloor_{\gamma_t} &= \lfloor \lfloor X_i \rfloor_{\gamma_t} + (X_i - \lfloor X_i \rfloor_{\gamma_t}) - \lfloor X_j \rfloor_{\gamma_t} - (X_j - \lfloor X_j \rfloor_{\gamma_t}) \rfloor_{\gamma_t} \\
&= \lfloor \Big(\lfloor X_i \rfloor_{\gamma_t} - \lfloor X_j \rfloor_{\gamma_t} \Big) + (X_i - \lfloor X_i \rfloor_{\gamma_t}) - (X_j - \lfloor X_j \rfloor_{\gamma_t}) \rfloor_{\gamma_t} \\
&= \lfloor X_i \rfloor_{\gamma_t} - \lfloor X_j \rfloor_{\gamma_t} + \lfloor (X_i - \lfloor X_i \rfloor_{\gamma_t}) - (X_j - \lfloor X_j \rfloor_{\gamma_t}) \rfloor_{\gamma_t}
\end{aligned}
$$

Now, since $\gamma_t > X_i - \lfloor X_i \rfloor_{\gamma_t} \geq 0$ (and similarly for j), then if $X_i - \lfloor X_i \rfloor_{\gamma_t} \geq X_j - \lfloor X_j \rfloor_{\gamma_t}$ we have $\gamma_t > X_i - \lfloor X_i \rfloor_{\gamma_t} - (X_j - \lfloor X_j \rfloor_{\gamma_t}) \geq 0$ and so rounding it down to the nearest multiple of γ_t makes it 0.

On the other hand, if $X_i - \lfloor X_i \rfloor_{\gamma_t} < X_j - \lfloor X_j \rfloor_{\gamma_t}$, then $0 < X_i - \lfloor X_i \rfloor_{\gamma_t} - (X_j - \lfloor X_j \rfloor_{\gamma_t}) \leq -\gamma_t$ and so rounding it down to the nearest multiple of γ_t makes it $-\gamma_t$. This proves the first part.

For the second part, Let $X_1, \ldots, X_n$ be i.i.d. exponential random variables. Let $X'_i = X_i - \lfloor X_i \rfloor_{\gamma_t}$. Then from Lemma 5.5 we know that $X'_1, \ldots, X'_n \mid \lfloor X_1 \rfloor_{\gamma_t}, \ldots, \lfloor X_n \rfloor_{\gamma_t}$ follow *independent* identical truncated exponential distribution on $[0, \gamma)$. Therefore, any ordering of the X'_i (given all of the $\lfloor X_j \rfloor_{\gamma_t}$) is equally likely and so has the same distribution as an ordering given by a uniformly random permutation. □

Now we are ready to establish the equivalence between Algorithm 3 and Algorithm 4.

Proof of Theorem 5.3. We first show that the output indices $j_1, \ldots, j_k$ from Algorithm 3 follow the same distribution as those from Algorithm 4. Note that $j_1, \ldots, j_k$ are determined by $\widehat{q}_i^{(t)}$ in Algorithm 3 and by $\breve{q}_i^{(t)}$ in Algorithm 4 respectively. Therefore, it suffices to show that for all $t \geq 0$, the set of random variables $(\widehat{q}_1^{(0)}, \ldots, \widehat{q}_n^{(0)}, \ldots, \widehat{q}_1^{(t)}, \ldots, \widehat{q}_n^{(t)})$ in Algorithm 3 and $(\breve{q}_1^{(0)}, \ldots, \breve{q}_n^{(0)}, \ldots, \breve{q}_1^{(t)}, \ldots, \breve{q}_n^{(t)})$ in Algorithm 4 have the same joint distribution.

For this part, it helps to view the variables as existing for all $t \geq 0$ before the algorithm even runs, but the algorithm only looks at the ones it needs (e.g., if the algorithm broke all ties for $t = t^*$, it does not look at $\widehat{q}_i^{(t^*+1)}$ or $\breve{q}_i^{(t^*+1)}$ even though they exist).

Since for $i_1 \neq i_2$, the set of random variables $\{\widehat{q}_{i_1}^{(0)}, \ldots, \widehat{q}_{i_1}^{(t)}\}$ (resp. $\{\breve{q}_{i_1}^{(0)}, \ldots, \breve{q}_{i_1}^{(t)}\}$) is independent of $\{\widehat{q}_{i_2}^{(0)}, \ldots, \widehat{q}_{i_2}^{(t)}\}$ (resp. $\{\breve{q}_{i_2}^{(0)}, \ldots, \breve{q}_{i_2}^{(t)}\}$), we have

$$P(\widehat{q}_1^{(0)}, \ldots, \widehat{q}_n^{(0)}, \ldots, \widehat{q}_1^{(t)}, \ldots, \widehat{q}_n^{(t)}) = \prod_{i=1}^{n} P(\widehat{q}_i^{(0)}, \ldots, \widehat{q}_i^{(t)})$$

$$P(\breve{q}_1^{(0)}, \ldots, \breve{q}_n^{(0)}, \ldots, \breve{q}_1^{(t)}, \ldots, \breve{q}_n^{(t)}) = \prod_{i=1}^{n} P(\breve{q}_i^{(0)}, \ldots, \breve{q}_i^{(t)})$$

Thus it suffices to show that $\forall i, t$, $P(\widehat{q}_i^{(0)}, \ldots, \widehat{q}_i^{(t)}) = P(\breve{q}_i^{(0)}, \ldots, \breve{q}_i^{(t)})$. Recall that in Algorithm 3, $\widetilde{q}_i = q_i + X$ where $X \sim \mathrm{Exp}(2k/\varepsilon)$ is random noise from the exponential distribution. Furthermore, we have $\widehat{q}_i^{(t)} = \lfloor \widetilde{q}_i \rfloor_{\gamma_t} = q_i + \lfloor X \rfloor_{\gamma_t}$ since by assumption q_i is a multiple of $\gamma_* = M^t \gamma_t$ (and hence also a multiple of γ_t). On the other hand, in Algorithm 4 we have $\breve{q}_i^{(t)} = q_i + \gamma_0 Y_0 + \gamma_1 (Y_1 \bmod M) \ldots + \gamma_t (Y_t \bmod M)$. Let $Z_{\gamma_t} = \lfloor X \rfloor_{\gamma_t}$ and $Z'_{\gamma_t} = \gamma_0 Y_0 + \sum_{j=1}^{t} \gamma_j (Y_j \bmod M)$. Then we just need to show that $P(Z_{\gamma_0}, \ldots, Z_{\gamma_t}) = P(Z'_{\gamma_0}, \ldots, Z'_{\gamma_t})$. We do this by induction on t.

The base case for $t = 0$ is simple. By Lemma 5.4, $Z_{\gamma_0} = \lfloor X \rfloor_{\gamma_0}$ follows the γ_0-scaled geometric distribution with success probability $p_0 = 1 - e^{-\frac{\varepsilon \gamma_0}{2k}}$, which is the same as $Z'_0 = \gamma_0 Y_0$.

Next, we assume (as an inductive hypothesis) that $P(Z_{\gamma_0}, \ldots, Z_{\gamma_t}) = P(Z'_{\gamma_0}, \ldots, Z'_{\gamma_t})$ and proceed to show that $P(Z_{\gamma_0}, \ldots, Z_{\gamma_{t+1}}) = P(Z'_{\gamma_0}, \ldots, Z'_{\gamma_{t+1}})$.

Note that by Lemma 5.7 the values of $Z_{\gamma_0}, \ldots, Z_{\gamma_{t-1}}$ are uniquely determined by Z_{γ_t} and so $P(Z_{\gamma_{t+1}} \mid Z_{\gamma_0}, \ldots, Z_{\gamma_t}) = P(Z_{\gamma_{t+1}} \mid Z_{\gamma_t})$.

Also, $0 \leq Z'_{\gamma_t} - Z'_{\gamma_{t-1}} = \gamma_t (Y_t \bmod M) < M\gamma_t = \gamma_{t-1}$, so $Z'_{\gamma_{t-1}} = \lfloor Z'_{\gamma_{t-1}} \rfloor_{\gamma_{t-1}} = \lfloor Z'_{\gamma_t} \rfloor_{\gamma_{t-1}}$. This means that the values of $Z'_{\gamma_0}, \ldots, Z'_{\gamma_{t-1}}$ are uniquely determined by Z'_{γ_t} and so $P(Z'_{\gamma_{t+1}} \mid Z'_0, \ldots, Z'_{\gamma_t}) = P(Z'_{\gamma_{t+1}} \mid Z'_{\gamma_t})$.

Therefore, it suffices to show that $P(Z_{\gamma_{t+1}} \mid Z_{\gamma_t}) = P(Z'_{\gamma_{t+1}} \mid Z'_{\gamma_t})$. Since $\gamma_t = M\gamma_{t+1}$, by Lemma 5.7 we have that $Z_{\gamma_{t+1}} = Z_{\gamma_t} + \gamma_{t+1}(Y \bmod M)$ where $Y \sim \text{Geo}(1 - e^{-\frac{\varepsilon\gamma_{t+1}}{2k}})$. By definition we have $Z'_{\gamma_{t+1}} = Z'_{\gamma_t} + \gamma_{t+1}(Y_{t+1} \bmod M)$ with $Y_{t+1} \sim \text{Geo}(1 - e^{-\frac{\varepsilon\gamma_{t+1}}{2k}})$. Thus they have the same distribution.

Lastly we show that the output gaps $g_1, \ldots, g_k$ from the two algorithms follow the same distribution. In Algorithm 3, the gaps are computed using $g_i = \lfloor \widetilde{q}_{j_i} - \widetilde{q}_{j_{i+1}} \rfloor_{\gamma_*} = q_{j_i} - q_{j_{i+1}} + \lfloor X_{j_i} - X_{j_{i+1}} \rfloor_{\gamma_*}$ (since the q_i are multiples of γ_*). To describe the gap computation in Algorithm 4, we need the following notation:

- π is a uniformly random permutation on $1, \ldots, n$ (total number of queries).
- $\breve{\delta}^{\pi}_{ij} = -1$ if $\pi(i) < \pi(j)$ and 0 otherwise.
- $Z'_{\gamma_t,i}$ is the noise in the i^{th} noisy query at resolution γ_t. I.e., $Z'_{\gamma_t,i} = \breve{q}^{(t)}_i - q_i$ for the value of t (tie breaking iteration number) at the gap calculation step.

Then the gaps in Algorithm 4 are computed as $\lfloor \breve{q}^{(t)}_{j_i} - \breve{q}^{(t)}_{j_{i+1}} - \gamma_t \breve{\delta}^{\pi}_{j_i,j_{i+1}} \rfloor_{\gamma_*} = q_{j_i} - q_{j_{i+1}} + \lfloor Z'_{\gamma_t,j_i} - Z'_{\gamma_t,j_{i+1}} - \gamma_t \breve{\delta}^{\pi}_{j_i,j_{i+1}} \rfloor_{\gamma_*}$, where j_i is the index of the query with the i^{th} largest noisy answer based on the resolution γ_t (when all ties have been broken). Note Algorithm 4 used a permutation over $1, \ldots, k+1$ and we are using a permutation over $1, \ldots, n$ here. This is completely equivalent because $\breve{\delta}^{\pi}_{i,j}$ is determined by how the permutation re-orders the numbers $1, \ldots, n$ (each ordering is equally likely); restricting the ordering to a subset of $1, \ldots, n$ (i.e., $j_1, \ldots, j_{k+1}$) still results in a uniformly random ordering on the subset.

Next we note that by Lemma 5.7, $\lfloor X_{j_i} - X_{j_{i+1}} \rfloor_{\gamma_*} = \lfloor \lfloor X_{j_i} - X_{j_{i+1}} \rfloor_{\gamma_t} \rfloor_{\gamma_*}$. Also, from the proof above, we know that the $Z'_{\gamma_t,i}$ variables (for all i) follow the same joint distribution as the $\lfloor X_i \rfloor_{\gamma_t}$'s. Therefore, we just need to show that $\lfloor X_i - X_j \rfloor_{\gamma_t}$ and $\lfloor X_i \rfloor_{\gamma_t} - \lfloor X_j \rfloor_{\gamma_t} - \gamma_t \breve{\delta}^{\pi}_{i,j}$ follow the same joint distribution over all i, j conditioned on values for all of the $\lfloor X_i \rfloor_{\gamma_t}$ for all i (since that is what is used to determine which queries to return and is the information available to the algorithm right before the gap calculation). This follows from Lemma 5.8. □

Lastly, we prove an upper bound on the expected runtime and memory of Algorithm 4.

Lemma 5.9. *Let $n \geq 2$ and $Y'_1, \ldots, Y'_n$ be i.i.d. random variables where $Y'_i \sim \text{Geo}(p) \bmod M$. The probability that $Y'_1 = \cdots = Y'_n$ is at most $\frac{(1-\varphi)(1+\varphi^M)}{(1+\varphi)(1-\varphi^M)}$ where $\varphi = 1 - p$.*

Proof. Since Y'_i are i.i.d., we have

$$P\left(Y'_1 = \cdots = Y'_n\right) = \sum_{m=0}^{M-1} P\left(Y'_1 = \cdots = Y'_n = m\right) = \sum_{m=0}^{M-1} \left(P\left(Y'_i = m\right)\right)^n$$

Since $P\left(Y'_i = m\right) < 1$, this sum is decreasing with respect to n. Therefore, $P\left(Y'_1 = \cdots = Y'_n\right)$ is maximum when $n = 2$. Let $\varphi = 1 - p$, this maximum value is

$$\sum_{m=0}^{M-1} \left(\frac{p\varphi^m}{1-\varphi^M}\right)^2 = \frac{p^2}{(1-\varphi^M)^2} \sum_{m=0}^{M-1} \varphi^{2m} = \frac{p^2}{(1-\varphi^M)^2} \frac{1-\varphi^{2M}}{1-\varphi^2} = \frac{1-\varphi}{1+\varphi} \cdot \frac{1+\varphi^M}{1-\varphi^M}.$$

□

Lemma 5.10. *For $m > 1$, the function $f(x) = \frac{(1-x)(1+x^m)}{(1+x)(1-x^m)}$ is decreasing on $(0,1)$.*

Proof. Let $L(x) = \ln f(x) = \ln(1-x) + \ln(1+x^m) - \ln(1+x) - \ln(1-x^m)$. We have

$$\begin{aligned} L'(x) &= -\frac{1}{1-x} + \frac{mx^{m-1}}{1+x^m} - \frac{1}{1+x} + \frac{mx^{m-1}}{1-x^m} \\ &= mx^{m-1}\left(\frac{1}{1-x^m} + \frac{1}{1+x^m}\right) - \left(\frac{1}{1-x} + \frac{1}{1+x}\right) \\ &= \frac{2mx^{m-1}}{1-x^{2m}} - \frac{2}{1-x^2} = \frac{2mx^{m-1}}{(1-x^2)(1+x^2+\cdots+x^{2m-2})} - \frac{2}{1-x^2} \\ &= \frac{2}{1-x^2}\left(\frac{mx^{m-1}}{1+x^2+\cdots+x^{2m-2}} - 1\right). \end{aligned}$$

By the AM-GM inequality, we have

$$\frac{1+x^2+\cdots+x^{2m-2}}{m} > (1\cdot x^2 \cdot \cdots \cdot x^{2m-2})^{\frac{1}{m}} = x^{\frac{0+2+\cdots+(2m-2)}{m}} = x^{m-1}.$$

Therefore, $1+x^2+\cdots+x^{2m-2} > mx^{m-1}$, hence $\frac{mx^{m-1}}{1+x^2+\cdots+x^{2m-2}} - 1 < 0$. For $x \in (0,1)$, $1-x^2 > 0$. So $L'(x) < 0$. Therefore, $f(x)$ is decreasing on $(0,1)$. □

Proof of Theorem 4.5. Let Loop 1 be the tie breaking loop in Algorithm 4, namely, Line 6 to Line 12 of Algorithm 4. Consider a modified version of Algorithm 4 in which ties are resolved among all queries, rather than only among the top $k+2$ queries. In this modified algorithm, Loop 1 is revised as shown on the left below as Loop 2; all other components of the algorithm remain unchanged. The only change is to the loop condition on Line 7. It is evident that Algorithm 4 terminates no later than the modified algorithm, as the original loop condition implies that of the modified loop. Consequently, the expected number of iterations of Loop 1 is upper bounded by that of Loop 2. In what follows, we demonstrate that Loop 2 terminates, in expectation, earlier than the alternative Loop 3 shown on the right.

To see that, consider the extreme case where $\breve{q}_1^{(0)} = \cdots = \breve{q}_n^{(0)}$, i.e., all n queries are tied. Note that if $\breve{q}_i^{(t)} < \breve{q}_j^{(t)}$ at some iteration t, then $\breve{q}_i^{(t')} < \breve{q}_j^{(t')}$ for all $t' > t$ since the addition of further noise at higher resolutions affects only less significant bits of the query values. In other words, once a tie between two queries is resolved, it remains resolved in all subsequent iterations. Therefore, after each iteration, the number of tied queries either remains unchanged or decreases by at least one. Assume at the start of iteration t, $\breve{q}_{i_1}^{(t)} = \cdots = \breve{q}_{i_s}^{(t)}$ are tied. Then by Lemma 5.9, the probability that they all remain tied at the end of iteration t is

$$P\left(\breve{q}_{i_1}^{(t+1)} = \cdots = \breve{q}_{i_s}^{(t+1)}\right) = P\left(Y'_{i_1} = \cdots = Y'_{i_s}\right) \le \frac{(1-\varphi_t)(1+\varphi_t^M)}{(1+\varphi_t)(1-\varphi_t^M)}$$

where $p_t = 1 - e^{-\varepsilon\gamma_t/2k}$ and $\varphi_t = 1 - p_t = e^{-\varepsilon\gamma_t/2k}$. Since $\gamma_t = \gamma_*/M^t$, we have $\gamma_t < \gamma_*$ and $\varphi_t > \varphi_* = e^{-\varepsilon\gamma_*/2k}$. By Lemma 5.10, $P\left(\breve{q}_{i_1}^{(t+1)} = \cdots = \breve{q}_{i_s}^{(t+1)}\right) \le \frac{(1-\varphi_*)(1+\varphi_*^M)}{(1+\varphi_*)(1-\varphi_*^M)}$. Therefore, with probability at least $\theta = 1 - \frac{(1-\varphi_*)(1+\varphi_*^M)}{(1+\varphi_*)(1-\varphi_*^M)}$, the number of tied queries is decreased by (at least) one after each iteration. Meanwhile, the variable *tie* in Loop 3 is initialized to n and decreases by one with probability θ in each iteration. As a result, after each iteration,

the number of tied queries in Loop 2 is less than the value of *tie* in Loop 3 in expectation. Consequently, the Loop 2 terminates earlier on average than Loop 3, and so does Loop 1 in Algorithm 4.

```
// Loop 2
t ← 0,    γ_0 ← γ_*
while there is a tie among q̆_1^(t), ..., q̆_n^(t) do
    t ← t + 1,    γ_t ← (1/M) γ_{t−1};
    for i = 1, ..., n do
        Y_t ← Geo(1 − e^{−εγ_t/2k})
        q̆_i^(t) ← q̆_i^(t−1) + γ_t · (Y_t mod M);
    j_1, ..., j_{k+2} ← arg max_{k+2}(q̆_1^(t), ..., q̆_n^(t))
```

```
// Loop 3
tie ← n
p_* ← 1 − e^{−εγ_*/2k}
φ_* ← 1 − p_*
θ ← 1 − ((1−φ_*)(1+φ_*^M)) / ((1+φ_*)(1−φ_*^M))
while tie > 0 do
    B ← Bernoulli(θ)
    if B == 1 then
        tie ← tie − 1
```

Next, we compute the expected runtime of Loop 3. Let T be the number of iterations this loop executes. Clearly, $T \geq n$. For any $k \geq 0$, the loop ends after iteration $T = n + k$ if $B = 1$ in the $(n+k)^{\text{th}}$ iteration, and B takes the value of 1 for $n-1$ times in the previous $n + k - 1$ iterations. Therefore, we have

$$P\left(T = n + k\right) = \theta \cdot \binom{n+k-1}{n-1}\theta^{n-1}(1-\theta)^k = \binom{n+k-1}{k}\theta^n(1-\theta)^k, \quad \text{and}$$

$$E(T) = \sum_{k=0}^{\infty}(n+k)P\left(T = n+k\right) = \sum_{k=0}^{\infty}(n+k)\binom{n+k-1}{k}\theta^n(1-\theta)^k$$

$$= \sum_{k=0}^{\infty} n\binom{n+k}{k}\theta^n(1-\theta)^k = n\theta^n\sum_{k=0}^{\infty}\binom{n+k}{k}(1-\theta)^k = n\theta^n \cdot \frac{1}{\theta^{n+1}} = \frac{n}{\theta}.$$

The second last equality is due to the following Taylor expansion:

$$\frac{1}{(1-x)^{n+1}} = \sum_{k=0}^{\infty}\binom{n+k}{n}x^k = \sum_{k=0}^{\infty}\binom{n+k}{k}x^k, \quad |x| < 1.$$

Therefore, the expected number of iterations of Loop 3 is n/θ, and the same bound applies to Loop 1 in Algorithm 4. Since each iteration requires $O(n \log n)$ time (primarily due to sorting) and $O(n)$ memory (for the increased resolution maintained for each query), the overall expected runtime and memory usage of Algorithm 4 are $O(n^2 \log n/\theta)$ and $O(n^2/\theta)$, respectively.

6. Experiments

We next evaluate the performance of the floating-point secure implementation. We implemented our algorithms in Python. All experiments are performed on an Intel® Core™ i9-10900X @ 3.7GHz CPU machine with 64 GB memory. In all experiments, we set the privacy budget ε to be 1 and the target precision γ_* is set to be 0.1, meaning all noisy gap values are rounded to the first decimal place. The precision increment factor M is set to 10. Thus if a precision of 0.1 is not enough to break all ties among top noisy queries, the algorithm switches precision 0.01 during tie-breaking, then 0.001 if necessary, etc.

Datasets. Since the performance of the algorithm can be affected by the distribution of query answers (which affect the probability of ties in the noisy query answers), we evaluate the implementation on two real datasets BMSPOS and Kosarak from [35], and a synthetic dataset T40I10D100K [2]. These datasets are collections of transactions (each transaction is a set of items). In the experiments, each query is associated with an item and the value of the query is the number of transactions the associated item appears in. The statistics of the datasets are listed below.

TABLE 3. Statistics of datasets

Dataset	**# of Records**	**# of Unique Items**
T40I10D100K	100,000	942
BMS-POS	515,597	1,657
Kosarak	990,002	41,270

Performance. The first set of experiments compare the running times of (1) "Secure": the unoptimized secure sampling algorithm (i.e., when a tie occurs, the resolution of all noisy query answers is increased); (2) "Opt. Secure": the optimized version that uses early query pruning; (3) the non-secure Python implementation of Algorithm 1 which uses exponential noise as implemented in the NumPy library. The results are shown in Table 4. For $k = 25, 50, 100, 200, 400$ and 800, we run each of the three algorithms for 1000 times and report the average running time on all three datasets.

TABLE 4. Running time (milliseconds)

Dataset	Algorithm	k=25	k=50	k=100	k=200	k=400	k=800
T40I10D100K $n = 942$	Secure	10.53	10.80	11.64	13.95	19.80	25.89
	Opt. Secure	10.59	10.64	10.83	11.51	14.74	23.70
	Baseline	2.25	2.24	2.27	2.32	2.38	2.52
BMS-POS $n = 1657$	Secure	18.51	18.51	19.03	22.11	35.21	48.72
	Opt. Secure	18.43	18.47	18.56	19.11	22.85	33.09
	Baseline	3.92	3.91	3.96	3.99	4.06	4.20
Kosarak $n = 41270$	Secure	458.22	459.45	482.02	575.38	976.72	1223.04
	Opt. Secure	455.32	455.17	454.89	455.59	459.73	471.30
	Baseline	98.71	97.53	97.38	98.69	98.95	99.03

For relatively small values of k, both versions of the secure algorithm are approximately 4.7× slower than the insecure algorithm, but this overhead is negligible because the overall runtime is in tens of milliseconds. When k is large, the unoptimized implementation becomes visibly slower because large values of k will make ties more likely to happen.

Running Time Breakdown. Next we focus on the optimized secure implementation and analyze its potential bottleneck. We separate Algorithm 4 into three parts. The first part (from Line 2 to Line 5) is to initially identify the possible top k noisy queries. Because of the possibility of the existence of ties among the chosen queries, the next part (from Line 6 to Line 12) uses a loop to iteratively increase noise precision until all ties are resolved. Finally, the last part (from Line 13 to Line 18) is to obtain numeric gaps estimates among chosen queries at target precision. We instrument the algorithm with timing instructions and report the average time spent on the three parts over 100 iterations. The results are summarised in Table 5.

TABLE 5. Detailed time for the optimized secure algorithm (milliseconds)

Dataset	Time spent to	k=25	k=50	k=100	k=200	k=400	k=800
T40I10D100K $n = 942$	Possible top-k	10.921	10.530	10.500	10.457	10.527	10.555
	Resolve ties	0.005	0.017	0.190	0.622	3.813	11.573
	Find gaps	0.049	0.081	0.154	0.300	0.599	1.192
BMS-POS $n = 1657$	Possible top-k	18.475	18.294	18.261	18.199	18.159	18.188
	Resolve ties	0.005	0.023	0.057	0.288	4.092	14.187
	Find gaps	0.045	0.081	0.154	0.302	0.598	1.181
Kosarak $n = 41270$	Possible top-k	461.56	457.82	457.371	458.721	456.256	459.552
	Resolve ties	0.008	0.012	0.074	0.744	5.077	15.586
	Find gaps	0.080	0.114	0.186	0.340	0.618	1.218

From the results there are several observations. First, the time spent to initially identify the possible top k queries is determined by n (total number of queries). This is consistent with our expectation because the algorithm needs to add noise to all n queries and then sort the noisy queries, which takes $O(n \log n)$ time (or $O(kn)$ when searching directly for the top $k + 1$ queries without sorting). Second, the time needed to resolve ties among top k queries is roughly $O(k \log^2 k)$. This is because each tie-breaking iteration runs in $O(k \log k)$ time (adding noise to roughly k queries and sorting them). Although more queries could result in more ties, a tie that is resolved in an iteration will not reappear in a later iteration. Thus the expected number of ties among the top k queries decrease exponentially with the number of iterations, and the expected number of iterations needed to break all ties is roughly $\log k$. Third, the time to get gaps among chosen queries is proportional to k, and is generally negligible. We remark that these numbers are consistent with the numbers reported in Table 4, in that the time spent on the three parts of the optimized algorithm sum up to roughly the time reported in Table 4 (middle row, Opt. Secure).

Time Spent in Different Sampling Commands. Last, we use a Python profiler (cProfile) to track the number of samples drawn from different sampling primitives during a single run of the optimized algorithm. We run our optimized algorithm on the Kosarak dataset ($n = 41270$) with $k = 800$. Recall that the optimized algorithm uses noise drawn from the geometric distribution (see Algorithm 5 in the Appendix), which subsequently draws samples from the Bernoulli and uniform distributions over integers (see Algorithm 1 in [9]). The number of evocations for each sampling command and the total time spent on it (including all subfunction calls) are reported in Table 6. We also include the sorting function in this

table as it clearly indicates how many times the tie breaking loop is run. Recall that the optimized algorithm first adds geometric noise to all $n = 41270$ queries and calls the sorting function to identify possible top queries. Then for each tie breaking loop iteration, additional geometric noise is drawn for each query in the pool of relevant queries to increase query answer resolution, and the sorting is called exactly once.

TABLE 6. Statistics on function invocations during a single execution

Subroutine name	**Total time spent**	**Number of calls**
Noisy Top-k with Gap (optimized)	1.208	1
Geometric distribution sampling	1.011	42,874
Bernoulli sampling	0.735	303,216
Uniform sampling over bounded integers	0.779	369,613
Identifying top noisy queries (argsort)	0.002	3

From this table, we can see that the optimized algorithm ran 2 loop iterations to resolve ties (argsort is called 3 times). Note also that the total number of geometric noise sampled is 42874. Since there are $n = 41270$ queries in the dataset, $42874 - 41270 = 1604$ additional geometric samples are used for tie breaking, 802 in each of the two loop iterations. This means early query elimination successfully removed queries that are definitely not among the top $k = 800$, leaving only $k + 2 = 802$ (the minimum required by the algorithm) queries for further investigation. On average, each secure geometric sampling command calls 7.07 Bernoulli and 8.62 uniform sampling commands. Lastly, it can be seen from the table that most of the time is spent on securely sampling random noise from various distributions.

TABLE 7. Statistics on function invocations in a single execution of Algorithm 1

Subroutine name	**Total time spent**	**Number of calls**
Noisy Top-k with Gap (insecure)	0.023	1
Adds Exp($2/\varepsilon$) noise to each query	0.078	41,270
Identifying top noisy queries (argsort)	0.002	1

For comparison, we also profiled the insecure implementation of the Noisy Top-k with Gap algorithm (Algorithm 1), with results summarized in Table 7. As shown, Algorithm1 was executed once, invoking its main subroutines accordingly. The subroutine responsible for adding exponential noise to each query was invoked 41,270 times, matching the total number of queries in the dataset (n=41,270). This confirms that, during the noise injection loop, the algorithm performs a single pass over all queries, adding noise sampled from the Exp($2/\varepsilon$) distribution to each query score. The cumulative time spent on noise addition was 0.078 seconds, which constitutes the majority of the total runtime. In contrast, the identification of the top-k noisy queries using the argsort subroutine was performed only once and accounted for a negligible 0.002 seconds.

To empirically assess the runtime behavior of the tie-breaking loops, we conducted an experiment comparing three variants: (i) resolving ties among the top $k + 2$ queries using truncated geometric noise (Loop 1 in Algorithm 4, Line 6 to Line 12), (ii) resolving all ties

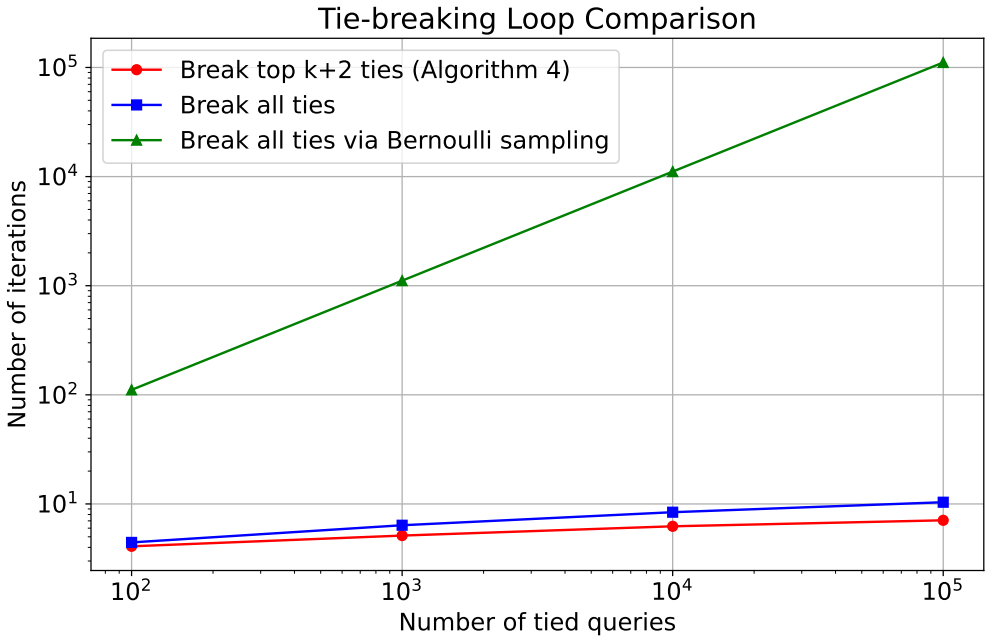

FIGURE 1. Iteration counts for tie-breaking loops across varying numbers of tied queries.

using truncated geometric noise (Loop 2 from the proof of Theorem 4.5), and (iii) resolving all ties using Bernoulli sampling (Loop 3 from the proof of Theorem 4.5). In this experiment, the number of queries n was set to 100, 1,000, 10,000 and 100,000, with all query values initialized to zero (so they are all tied). For each value of n, each loop was executed from start to termination 100 times, and the average number of iterations over these runs is reported in Figure 1.

As shown in Figure 1, the average number of iterations for Loop 3 scales linearly with the number of queries n and serves as an upper bound for the number of iterations required by the other two loops, thereby supporting the correctness of Theorem 4.5. As expected, resolving ties among all queries consistently requires more iterations than resolving ties among only the top $k+2$ queries. Moreover, the substantial gap between the iteration counts of Loop 3 and Loop 1 indicates that the bound established in Theorem 4.5 is conservative, and that Algorithm 4 terminates significantly faster in practice.

7. Conclusion

In this paper, we proposed an implementation of the differentially private Noisy Top-k with Gap algorithm that avoids the use of floating point and hence is secure against floating point vulnerability. The algorithm is probabilistically equivalent to the ideal algorithm that uses continuous noise and rounds its inputs and outputs. The algorithm makes heavy use of a variety of memoryless properties of geometric and exponential random variables to dynamically determine the appropriate level of discretization for the noise and uses carefully controlled randomized rounding routines to provide the gap information.

Acknowledgment

This work was support by NSF awards CNS 1702760, CNS 1931686, CNS 2317232 and CNS 2317233.

References

[1] J. M. Abowd. The u.s. census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, pages 2867–2867, New York, NY, USA, 2018. ACM. https://doi.org/10.1145/3219819.3226070.

[2] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499. Santiago, Chile, 1994.

[3] Apple Differential Privacy Team. Learning with privacy at scale. *Apple Machine Learning Journal*, 1(8), 2017. https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html.

[4] V. Balcer and S. Vadhan. Differential privacy on finite computers. *Journal of Privacy and Confidentiality*, 9(2), Sep. 2019. https://doi.org/10.29012/jpc.679.

[5] R. Bhaskar, S. Laxman, A. Smith, and A. Thakurta. Discovering frequent patterns in sensitive data. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010. https://doi.org/10.1145/1835804.1835869.

[6] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 441–459, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5085-3. https://doi.org/10.1145/3132747.3132769.

[7] J. Blocki, A. Datta, and J. Bonneau. Differentially private password frequency lists. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. https://doi.org/10.14722/ndss.2016.23328.

[8] U. S. C. Bureau. On the map: Longitudinal employer-household dynamics. https://lehd.ces.census.gov/applications/help/onthemap.html#!confidentiality_protection, 2019.

[9] C. L. Canonne, G. Kamath, and T. Steinke. The discrete gaussian for differential privacy. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

[10] S. Casacuberta, M. Shoemate, S. Vadhan, and C. Wagaman. Widespread underestimation of sensitivity in differentially private libraries and how to fix it. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 471–484, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394505. https://doi.org/10.1145/3548606.3560708.

[11] B. Ding, J. Kulkarni, and S. Yekhanin. Collecting telemetry data privately. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17,

pages 3574–3583, USA, 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4. http://dl.acm.org/citation.cfm?id=3294996.3295115.

[12] Z. Ding, Y. Wang, D. Zhang, and D. Kifer. Free gap information from the differentially private sparse vector and noisy max mechanisms. *Proc. VLDB Endow.*, 13(3):293–306, nov 2019. ISSN 2150-8097. https://doi.org/10.14778/3368289.3368295.

[13] Z. Ding, D. Kifer, S. Saghaian, T. Steinke, Y. Wang, Y. Xiao, and D. Zhang. The permute-and-flip mechanism is identical to report-noisy-max with exponential noise. *CoRR*, abs/2105.07260, 2021. https://arxiv.org/abs/2105.07260.

[14] Z. Ding, Y. Wang, Y. Xiao, G. Wang, D. Zhang, and D. Kifer. Free gap estimates from the exponential mechanism, sparse vector, noisy max and related algorithms. *The VLDB Journal*, 2022. https://doi.org/10.1007/s00778-022-00728-2.

[15] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, aug 2014. ISSN 1551-305X. https://doi.org/10.1561/0400000042.

[16] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, page 265–284, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3540327312. https://doi.org/10.1007/11681878_14.

[17] U. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1054–1067, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2957-6. https://doi.org/10.1145/2660267.2660348.

[18] R. A. Fisher and F. Yates. Statistical tables for biological, agricultural and medical research. *Science*, 88(2295):596–597, 1938. https://doi.org/10.1126/science.88.2295.596.

[19] Free Software Foundation. The gnu c++ library api reference, 2022. https://gcc.gnu.org/onlinedocs/libstdc++/api.html. Accessed on 06.20.2022.

[20] S. L. Garfinkel, J. M. Abowd, and S. Powazek. Issues encountered deploying differential privacy. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, 2018. https://doi.org/10.1145/3267323.3268949.

[21] I. Gazeau, D. Miller, and C. Palamidessi. Preserving differential privacy under finite-precision semantics. *Theoretical Computer Science*, 655:92–108, 2016. https://doi.org/10.1016/j.tcs.2016.01.015.

[22] Q. Geng and P. Viswanath. The optimal noise-adding mechanism in differential privacy. *IEEE Transactions on Information Theory*, 62(2):925–951, 2016. https://doi.org/10.1109/TIT.2015.2504967.

[23] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 351–360, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585062. https://doi.org/10.1145/1536414.1536464.

[24] E. Gumbel. *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures*. Applied mathematics series. U.S. Government Printing Office, 1954. https://books.google.com/books?id=SNpJAAAAMAAJ.

[25] S. Haney, A. Machanavajjhala, J. M. Abowd, M. Graham, M. Kutzbach, and L. Vilhuber. Utility cost of formal privacy for releasing national employer-employee statistics. In

Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17, pages 1339–1354, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4197-4. https://doi.org/10.1145/3035918.3035940.

[26] S. Haney, D. Desfontaines, L. Hartman, R. Shrestha, and M. Hay. Precision-based attacks and interval refining: how to break, then fix, differential privacy on finite computers. *CoRR*, abs/2207.13793, 2022. https://doi.org/10.48550/arXiv.2207.13793.

[27] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2339–2347, USA, 2012. Curran Associates Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/208e43f0e45c4c78cafadb83d2888cb6-Paper.pdf.

[28] N. Holohan and S. Braghin. Secure random sampling in differential privacy. In *Computer Security – ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part II*, page 523–542, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-88427-7. https://doi.org/10.1007/978-3-030-88428-4_26.

[29] C. Ilvento. *Implementing the Exponential Mechanism with Base-2 Differential Privacy*, page 717–742. Association for Computing Machinery, New York, NY, USA, 2020. https://doi.org/10.1145/3372297.3417269.

[30] J. Jin, E. McMurtry, B. I. P. Rubinstein, and O. Ohrimenko. Are we there yet? timing and floating-point attacks on differential privacy systems. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 473–488, 2022. https://doi.org/10.1109/SP46214.2022.9833672.

[31] N. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for sql queries. *Proceedings of the VLDB Endowment*, 11(5):526–539, 2018.

[32] D. E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison-Wesley, 1969. ISBN 0201038021. https://www.worldcat.org/oclc/310551264.

[33] D. Lemire. Fast random integer generation in an interval. *ACM Trans. Model. Comput. Simul.*, 29(1), jan 2019. ISSN 1049-3301. https://doi.org/10.1145/3230636.

[34] J. Liu and K. Talwar. Private selection from private candidates. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 298–309, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367059. https://doi.org/10.1145/3313276.3316377.

[35] M. Lyu, D. Su, and N. Li. Understanding the sparse vector technique for differential privacy. *Proc. VLDB Endow.*, 10(6):637–648, Feb. 2017. ISSN 2150-8097. https://doi.org/10.14778/3055330.3055331.

[36] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber. Privacy: From theory to practice on the map. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 277–286, 2008. https://doi.org/10.1109/ICDE.2008.4497436.

[37] C. J. Maddison, D. Tarlow, and T. Minka. A* sampling. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3086–3094, Cambridge, MA, USA, 2014. MIT Press. https://proceedings.neurips.cc/paper_files/paper/2014/file/937debc749f041eb5700df7211ac795c-Paper.pdf.

[38] R. McKenna and D. Sheldon. Permute-and-flip: A new mechanism for differentially private selection. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. https://proceedings.neurips.cc/paper_files/paper/2020/file/01e00f2f4bfcbb7505cb641066f2859b-Paper.pdf.

[39] S. Messing, C. DeGregorio, B. Hillenbrand, G. King, S. Mahanti, Z. Mukerjee, C. Nayak, N. Persily, B. State, and A. Wilkins. Facebook Privacy-Protected Full URLs Data Set. https://doi.org/10.7910/DVN/TDOAPG, 2020.

[40] I. Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 650–661, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450316514. https://doi.org/10.1145/2382196.2382264.

[41] A. Smith. Privacy-preserving statistical estimation with optimal convergence rates. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, page 813–822, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306911. https://doi.org/10.1145/1993636.1993743.

[42] The Go Authors. Package rand implements pseudo-random number generators, 2022. https://github.com/golang/go/blob/master/src/math/rand/rand.go. Accessed on 06.20.2022.

[43] J. von Neumann. Various techniques used in connection with random digits. In A. S. Householder, G. E. Forsythe, and H. H. Germond, editors, *Monte Carlo Method*, volume 12 of *National Bureau of Standards Applied Mathematics Series*, chapter 13, pages 36–38. US Government Printing Office, Washington, DC, 1951.

Appendix A. Geometric Sampling Primitive

Algorithm 5: Sampling from a Geometric Distribution (extracted from Algorithm 2 in [9])

input : Integers $s, t \geq 1$
output : One sample from $\text{Geo}(1 - e^{-s/t})$

function `Geometric`$(1 - e^{-s/t})$**:**
  $D \leftarrow 0$
  while $D = 0$ **do**
    $U \leftarrow \mathcal{U}\{0, \ldots, t-1\}$
    $D \leftarrow \text{Bernoulli}(e^{-U/t})$ `// Use Algorithm 1 in [9]`
  $V \leftarrow 0$ `// Generate` V `from` $\text{Geo}(1 - e^{-1})$
  while `true` **do**
    $A \leftarrow \text{Bernoulli}(e^{-1})$ `// Use Algorithm 1 in [9]`
    if $A = 0$ **then**
      Break
    else
      $V \leftarrow V + 1$
  $X \leftarrow U + t \cdot V$ `//` X `is` $\text{Geo}(1 - e^{-1/t})$
  $Y \leftarrow \lfloor X/s \rfloor$ `//` Y `is` $\text{Geo}(1 - e^{-s/t})$
  return Y