# Pure Differential Privacy for Rectangle Queries via Private Partitions

Cynthia Dwork[1], Moni Naor[2]*, Omer Reingold[3], and Guy N. Rothblum[4]

[1] Microsoft Research, dwork@microsoft.com
[2] The Weizmann Institute, moni.naor@weizmann.ac.il
[3] Samsung Research America, omer.reingold@gmail.com
[4] Samsung Research America, rothblum@alum.mit.edu

**Abstract.** We consider the task of data analysis with pure differential privacy. We construct new and improved mechanisms for statistical release of interval and rectangle queries. We also obtain a new algorithm for counting over a data stream under continual observation, whose error has optimal dependence on the data stream's length.
A central ingredient in all of these result is a differentially private partition mechanism. Given set of data items drawn from a large universe, this mechanism outputs a partition of the universe into a small number of segments, each of which contain only a few of the data items.

## 1 Introduction

Differential privacy is a recent privacy guarantee tailored to the problem of statistical disclosure control: how to publicly release statistical information about a set of people without compromising the privacy of any individual [DMNS06] (see the book [DR14] for an extensive treatment). In a nutshell, differential privacy requires that the probability distribution on the published results of an analysis is "essentially the same," independent of whether any individual opts in to, or opts out of, the data set. (The probabilities are over the coin flips of the privacy mechanism.) Statistical databases are frequently created to achieve a social goal, and increased participation in the databases permits more accurate analyses. The differential privacy guarantee supports the social goal by assuring each individual that she incurs little risk by joining the database: anything that can happen is essentially equally likely to do so whether she joins or abstains.

In the differential privacy literature, privacy is achieved by the introduction of randomized noise into the output of an analysis. Moreover, sophisticated mechanisms for differentially private data analysis can incur a significant efficiency overhead. A rich and growing literature aims to minimize the "cost of privacy" in terms of the error and also in terms of computational efficiency. In this work

* Incumbent of the Judith Kleeman Professorial Chair. Research supported in part by grants from the Israel Science Foundation, BSF and Israeli Ministry of Science and Technology and from the I-CORE Program of the Planning and Budgeting Committee and the Israel Science Foundation (grant No. 4/11).

we present new algorithms with improved error for several natural data analysis tasks.

There are several variants of differential privacy that have been studied. Most notably, these include the stronger (in terms of privacy-protection) notion of *pure* differential privacy, and its relaxation to *approximate* differential privacy. Our work focuses on mechanisms that guarantee *pure* differential privacy for the tasks of answering statistical queries, maintaining an online count of significant events in a data stream, and partitioning a large universe into a small number of contiguous segments, none of which contains too many input items (a type of "dimension reduction").

Before proceeding to outline our contributions, we recall the definition of differential privacy:

**Definition 1.1 (Differential Privacy [DMNS06,DKM$^+$06]).** *A randomized algorithm $M : \mathcal{U}^n \to Y$ is* $(\varepsilon, \delta)$-differentially private *if for every pair adjacent databases $x, x'$ that differ only in one row, and for every $S \subset Y$:*

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

*When $\delta = 0$, we say the algorithm provides* (pure) $\varepsilon$-differential privacy*. When $\delta > 0$, we say that the algorithm provides* (approximate) differential privacy*.*

As discussed above, we focus on the stronger guarantee of *pure* differential privacy throughout this work.

## 1.1 Differentially Private Query Release: Interval and Rectangle Queries

Differentially private query release is a central problem in the literature. The goal is releasing the answers to a set of statistical queries while maintaining both differential privacy and low error. We focus on the case of *counting queries* (sometimes referred to as statistical queries). Let $\mathcal{U}$ be the set of possible data items (the data universe). A counting query $q$ is specified by a predicate $q : \mathcal{U} \to \{0, 1\}$. For an $n$-element database $x \in \mathcal{U}^n$, the query output $q(x) \in [0, n]$ counts how many items in the database satisfy the query. The goal, given a set $Q$ of queries and a database $x$, is to approximate $q(x)$ for each $q \in Q$, while $(i)$ guaranteeing differential privacy (for the collection of all answers), and $(ii)$ minimizing error in the answers.

We focus on the (challenging) setting where the query set $Q$ is large. To avoid running in time proportional to $|Q|$ (which is too large), we will produce a differentially private *data synopsis*. Given the database $x$, the mechanism produces a synopsis: a data structure that can later be used to answer any query $q \in Q$. Thus, the synopsis is a small implicit representation for the answers to all queries in $Q$.

Differentially private query release, especially for counting queries, has been the focus of a rich literature. Starting with the works of Dinur, Dwork and Nissim [DN03,DN04], showed how to answer $k$ queries (counting queries or general low-sensitivity queries) using computationally efficient mechanisms, with noise that grew with $k$ for pure $\varepsilon$-DP [DMNS06], or $\sqrt{k}$ for approximate $(\varepsilon, \delta)$-DP. Starting with the work of Blum, Ligett and Roth [BLR08], later works improved the dependence on the number of queries $k$ to logarithmic. The running time for these mechanisms, however, can be prohibitive in many settings. Even the state-of-the-art mechanisms for answering general counting queries [HR10] require running time that is at least linear in the *size of the data universe* $|\mathcal{U}|$ (whereas the running time of earlier mechanisms was logarithmic in $|\mathcal{U}|$). Indeed, for many query sets $Q$, the best differentially private query release mechanisms that are known require either large error (as a function of $|Q|$), or large running time (as a function of $|\mathcal{U}|$ and $|Q|$). Indeed, under cryptographic assumptions, there are inherent limits on the computational efficiency and the accuracy of differentially private query release algorithms for specific sets of counting queries [DNR$^+$09,Ull13,BUV14]. Thus, a significant research effort has aimed to design efficient and accurate DP mechanisms for specific natural sets of counting queries.

Our work continues this effort. We construct new and improved mechanisms for answering interval or threshold queries. We further extend these results to multi-dimensional rectangle queries, and for these queries we are able to increase the data dimensionality with relatively mild loss in accuracy and efficiency.

*Interval Queries.* We consider the natural class of *interval queries.* Here the data universe is the integers from 1 to $D$ (i.e. $\mathcal{U} = [1, D]$, and $|\mathcal{U}| = D$).[5] Each query $q$ is specified by an *interval* $I = [i, j] \subseteq [1, D]$, and associated with the predicate that outputs 1 on data elements that fall in that interval. Usually we think of $D$ as being very large, much larger than (even exponential in) the database size $n$. For example, the data universe could represent a company's salary information, and interval queries approximate the number of employees whose salaries fall in a certain bracket.

In prior work, Dwork *et al.* [DNPR10] showed that this class could be answered with pure $\varepsilon$-differential privacy and error roughly $O(\frac{\log^2 D}{\varepsilon})$ (see the analysis in [CSS11]). They also showed an $\Omega(\log D)$ error lower bound for obtaining pure differential privacy. Our first contribution is a new mechanism that obtains pure differential privacy with error roughly $O(\frac{\log D + (\log^2 n)}{\varepsilon})$. In particular, the error's dependence on $D$ is optimal.

**Theorem 1.2 (DP Intervals).** *The mechanism in Section 3.2 answers interval queries over* $[1, D]$*. For any privacy and accuracy parameters* $\varepsilon, \beta > 0$*, it guarantees (pure)* $\varepsilon$*-differential privacy. For any database* $x$ *of size* $n$*, with all but* $\beta$ *probability over the mechanism's coins, it produces a synopsis that answers all interval queries (simultaneously) with error* $O(\frac{\log D + ((\log^2 n) \cdot \log(1/\beta))}{\varepsilon})$*. The running time to produce the synopsis (and to then answer any interval query) is* $(n \cdot \mathrm{poly}(\log D, \log(1/\varepsilon), \log(1/\beta)))$*.*

[5] throughout this work, for integers $i, j$ s.t. $i \leq j$, we use the notation $[i, j]$ to denote the (closed) interval of integers $\{i, i+1, \ldots, j-1, j\}$.

While the error's dependence on $\log D$ is optimal, we do not know whether the dependence on $\log^2 n$ is optimal (i.e., whether the error is tight for cases where $D$ is not much larger than $n$). This remains a fascinating question for future work.

The main idea behind this mechanism is partitioning the data universe $[1, D]$ into at most $n$ contiguous segments, where the number of items in each segment is not too large. We give a new differentially private mechanism for constructing such a partition, see Sections 1.3 and 2. Given this partition, we treat the $n$ segments as a new smaller data universe, and use the algorithm of [DNPR10] to answer interval queries on this smaller data universe (this is where we incur the $\log^2 n$ error term).

*Related work:* approximately *private threshold queries.* The class of interval queries generalizes the class of threshold queries, where each query is specified by $i \in [1, D]$ and counts how many items in the input database are larger than $i$ (i.e., how many items are in the interval $[i, D]$). In fact, since answers to threshold queries can also be used to answer interval queries, these two classes are equivalent. Answering threshold queries with *approximate* $(\varepsilon, \delta)$-DP was considered in the work of Beimel, Nissim and Stenner [BNS13], who obtain an upper bound of $2^{O(\log^* D)}$. In a beautiful recent independent work, Bun, Nissim, Stemmer and Vadhan [BNSV15] show a lower bound of $\Omega(\log^* D)$ for approximate-DP mechanisms (as well as an improved upper bound of roughly $2^{\log^* D}$). The main difference with our work is that we focus on the stricter guarantee of *pure* differential privacy, which (provably) incurs a larger error.

*Rectangle Queries.* We further study a natural generalization of interval queries: *rectangle queries.* These queries consider multi-dimensional data (in particular, $c$-dimentional for an integer $c > 1$). The data universe is $\mathcal{U} = [1, D]^c$. A rectangle query $q$ is specified by a *rectangle* $R = ([i_1, j_1] \times \ldots \times [i_c, j_c]) \subseteq [1, D]^c$, and associated with the predicate that outputs 1 on data items that fall inside the set $R$. As was the case for interval queries, we usually think of $D$ as larger than $n$, and of $c$ as being smaller than either of these quantities (sub-logarithmic in $n$, or even constant). Continuing the example above, a database could contain employees' salaries, ages, years of experience, rank, etc. Rectangle queries can be used to approximate the numbers of employees that fall into various conjunctions of brackets, e.g. the number of employees in given age, experience and salary brackets. More generally, these queries are useful for multi-dimensional data, where many (or all) of the data dimensions are associated with an ordering on data items in that dimension.

We generalize the intervals mechanism to answer rectangle queries. While in many settings known differentially private algorithms suffer from a "curse of dimensionality" that increases the error or running time as the dimension grows, we give an algorithm whose error and running time have a mild dependence on the data dimensionality. In particular, the error is roughly $O((c^2 \cdot \log D) + ((\log n)^{O(c)})$. The running time is roughly $n \cdot \text{poly}(\log^c n, \log D)$, and does not grow with $D^c$. For the (reasonable) setting of parameters where we think of $n \approx \log D$, the running time is only polynomial in $(\log \log D)^c$.

**Theorem 1.3 (DP Rectangles).** *The mechanism described in Section 3.3 answers c-dimensional rectangle queries over* $[1, D]^c$. *For any privacy and accuracy parameters* $\varepsilon, \beta > 0$*, the mechanism guarantees (pure)* $\varepsilon$*-differential privacy. With all but* $\beta$ *probability over its coins, all rectangle queries (simultaneously) are answered with error* $O(\frac{(c^2 \cdot \log D)+((\log n)^{O(c)} \cdot \log(1/\beta))}{\varepsilon})$. *The running time to produce the synopsis (and to then answer any rectangle query) is* $(n \cdot \text{poly}(\log^c n, \log D, \log(1/\varepsilon), \log(1/\beta)))$.

In prior work, Chan Shi and Song [CSS11] considered rectangle queries and obtained an error bound of roughly $(\log D)^{O(c)}$. Theorem 1.3 roughly replaces this with a $(\log n)^{O(c)}$ term, as well as an additive $O(c^2 \cdot \log D)$ (recall that typically $n << D$). We emphasize that the error's dependence on $\log D$ *does not grow exponentially with the dimentionality c.*

Muthukrishnan and Nikolov [MN12] show an $\Omega((\log n)^{c-O(1)})$ error lower-bound when $n \approx D$, even for (the relaxed notion of) $(\varepsilon, \delta)$-differentially private algorithms (they refer to this as "orthogonal range counting"). Thus, the dependance on $\log n$ in the mechanism of Theorem 1.3 is optimal up to a (small) polynomial factor (the exact term in our upper bound is $O((\log n)^{1.5c+1})$).

The rectangles mechanism is a multi-dimensional generalization of the intervals mechanism (see more above and below). Recall that the intervals algorithm utilized a differentially private partition of the data universe into $n$ segments. It then used the "tree-counter" algorithm of [DNPR10] to answer interval queries over these $n$ segments. This is done by building a binary tree of noisy counts, whose leaves are the $n$ segments. For the rectangle mechanism, we use a $(k, d)$-tree-like data structure (see [Ben75] and see also the rectangle mechanism of [CSS11]), building a "tree of trees" of noisy counts along the $c$ dimensions of the data universe (after reducing the size of each dimension using a differentially private partition). We judiciously prune this tree to avoid an exponential blowup in its size (the naive implementation requires time and memory $n^c$). A careful analysis guarantees that even while we extend to $c$ dimensions, the error (as a function of $D$) only grows to $O(c^2 \cdot D)$.

## 1.2 Counting under Continual Observation

Dwork, Naor, Pitassi and Rothblum [DNPR10] introduced the problem of *counting under continual observation.* The goal is to monitor a stream of $D$ bits, and continually maintain an approximation of the number of 1's that have been observed so far. For privacy, the entire collection of $D$ outputs (where the $i$-th output approximates the count after processing $i$ elements) should maintain $\varepsilon$-differential privacy, masking the value of any single bit. The canonical application is monitoring events, such as the number of influenza patients arriving at medical office, or the number of users visiting a webpage (where privacy hides any single access). Since its introduction online counting has found many applications. In most settings, the data stream is *sparse*: the number of 1's (the stream's "weight") is much smaller than $D$.

The online counter proposed by [DNPR10] (we refer to this as the "tree counter") had error roughly $O(\log^2 D)$ (see the analysis in [CSS11]). As an additional contribution, we present an improved counter (with pure or $(\varepsilon, 0)$ differential privacy) for sparse streams. In particular, thinking of the input as a boolean string where the number of 1's is at most $n$ (and $n \ll D$), the error is improved to roughly $(\log D + (\log^2 n))$ (compared with roughly $(\log^2 D)$ for the tree counter). We note that the dependence on $D$ is optimal, and matches the $\Omega(\log D)$ lower bound in [DNPR10].

**Theorem 1.4.** *For any $\varepsilon, \beta > 0$, the online counter from Section 3.1 guarantees $\varepsilon$-differential privacy. Taking $n$ to be an upper bound on the input stream's weight, with all but $\beta$ probability over the counter's coins, the maximal error over all $D$ items is at most $O(\frac{\log D + ((\log^2 n) \cdot \log(1/\beta))}{\varepsilon})$.*

Here again, we partition the data stream (of length $D$) into at most $n$ segments, where the number of items in each segment is not too large. This is done using an *online* partition mechanism, which can process the items one-by-one, and after processing each item can decide whether a segment is large enough to be "sealed", or whether to keep accumulating the current segment (see Sections 1.3 and 2). Given this *online* partition mechanism, we can run the tree counter of [DNPR10] (or any other counter) on its output. As we process data items, we don't update the count until the current segment is sealed. When a segment is sealed, we feed the count within this segment into the tree counter, and obtain an updated count (we use here the fact that the tree counter can also operate on integer inputs, not just on bits).

## 1.3 Differentially Private Online Partition

As mentioned above, one of the main tools we use is a (pure) $\varepsilon$-differentially private partition algorithm. Given an $n$-item database $x \subseteq [1, D]$, this algorithm partitions the data universe $\mathcal{U} = [1, D]$ into (at most) $n$ contiguous segments $(S_1 = [1, s_1], S_2 = [s_1 + 1, s_2], \ldots, S_n = [s_{n-1} + 1, D])$ (where the $s_i$'s are all integers). The guarantee is that w.h.p. the number of data elements in each of these segments is small, and bounded by roughly $O(\log D)$. These partitions are pervasive in the applications mentioned above. In a nutshell, we treat the segments as a new and reduced data universe. This reduces the size of the data universe from $D$ to $n$, an exponential improvement for some of the parameter regimes of interest. Beyond its applications in this work, we find the partition mechanism to be of independent interest, and hope that it will find further applications.

**Theorem 1.5.** *For any $\varepsilon, \beta > 0$, the Partition mechanism in Section 2 guarantees $\varepsilon$-differential privacy. When run on a database of size $n$, with all but*

*$\beta$ probability over the mechanism's coins, it outputs at most $n$ segments, and each segment is of weight at most $\frac{5(\log D+\log(1/\beta))}{\varepsilon}$. The running time is $n \cdot \mathrm{poly}(\log D, \log(1/\varepsilon), \log(1/\beta))$.*

The Partition algorithm and its analysis are inspired by an algorithm from [DNPR10] for transforming a class of streaming algorithms into ones that are private even under continual observation.

Another important property of this algorithm is that it can be run in an *online* manner. In this setting, the input is treated as a bit-stream of length $D$. The $i$-th input $y_i \in \{0,1\}$ indicates whether item $i$ is in the dataset. Thus, this is a *sparse* stream with total weight $n$. The partition mechanism can process these bits one-by-one, making an online decision about when to "seal" each segment. We use this online of the partition algorithm to obtain an improved online counter.

## 2 Differentially Private Online Partition

*The Mechanism.* The (online) partition algorithm processes the input as a stream $x_1, \ldots, x_D \in \{0,1\}$. We use $n$ to denote the weight of the stream (the number of 1's).[6] The output is a partition of $[D]$ into (contiguous) segments $P = (S_1, \ldots, S_j)$, such that:

1. W.h.p. the number of segments $j$ is smaller than $n$.
2. The weight of the items in each segments is $O((\log D + \log(1/\beta))/\varepsilon)$ (where $\varepsilon$ is the privacy parameter).

This is an online algorithm, in the sense that after processing the $i$-th data item, the algorithm either "seals" a new segment, ending at $i$, or it keeps the current segment "open" and proceeds to the next data item. We emphasize that the algorithm is oblivious to the input stream's weight. The Partition algorithm and its analysis are inspired by an algorithm from [DNPR10] for transforming a class of streaming algorithms into ones that are private even under continual observation (see also the discussion of the "sparse vector" abstraction in [DR14]).

**Theorem 2.1.** *For any $\varepsilon, \beta > 0$, the Partition Algorithm of Figure 1 guarantees $\varepsilon$-differential privacy. Let $n$ be the total weight of the input stream. With all but $\beta$ probability over the algorithm's coins, it outputs at most $n$ segments, and each segment is of weight at most $\frac{5(\log D+\log(1/\beta))}{\varepsilon}$.*

Before proving the partition algorithm's privacy and accuracy, we remark that the dependence on $\log D$ is optimal by the lower bound of [DNPR10]. Moreover, for an *offline* implementation, where the input is given as an $n$-item database $x \subseteq [1, D]$, we can reduce the running time to $\mathrm{polylog} D$:

[6] More generally, we could also work with a stream of integers, and the weight would be the $L_1$ norm.

```
Partition (D, ε, β)
Initialize the threshold $T \leftarrow (3(\log D + \log(1/\beta))/\varepsilon)$, and indices $i, j \leftarrow 0$
Repeat the following loop: (each iteration of the loop seals a new segment)

  1. Initialize the $j$-th segment:
     $j \leftarrow j+1$, $count_j \leftarrow 0$, $\widetilde{T}_j \leftarrow T + Lap(1/\varepsilon)$
  2. Repeat the following loop, processing the $i$-th data item in each iteration:
     (a) $i \leftarrow i+1$, $count_j \leftarrow count_j + x_i$
     (b) $\widetilde{count}_i \leftarrow count_j + Lap(1/\varepsilon)$
     Keep the $j$-the segment open until $(\widetilde{count}_i > \widetilde{T}_j)$ or $(i \geq D)$
  3. Seal the $j$-th segment: $s_j \leftarrow i$

Until $(i \geq D)$. Take $m \leftarrow j$ to be the final number of segments
Output the partition $P = \{[1, s_1], [s_1+1, s_2], \ldots, [s_{m-1}+1, D]\}$ and the number of segments $m$
```

**Fig. 1.** Online DP Partition Algorithm

*Remark 2.2.* [Efficient Offline Implementation] For the offline settings, where the input is an $n$-item database $x \subseteq [1, D]$, we can compute the partition in $n \cdot \text{polylog}(D)$ time as follows. We sort the $n$ items so that $x_1 < x_2 < \ldots < x_n$ (where each $x_k \in [1, D]$). We then process the items one by one. When processing the $k$-th item $x_k$, assume that the last sealed segment was sealed at $s_j$. We count the number of database items in $[s_j + 1, x_k]$. This gives a certain probability $p$ that the $(j+1)$-th segment will be sealed at $x_k$. Until the $(j+1)$-th segment is sealed, for every $y \in [x_k, x_{k+1} - 1]$, the probability that the $(j+1)$-th segment is sealed at $y$ remains $p_k$ (because there are no additional items processed). We can now sample in polylog$D$ time whether the segment is sealed in the range $[x_k, x_{k+1} - 1]$. If we sample that the segment is sealed at some $y^*$ in this range, then we run the above process again starting at $y$ (with a new probability computed from the updated true count, which becomes 0). If not, then we run it again starting at $x_{k+1}$ (again from the updated the count, which is incremented).

*Proof (Proof of Theorem 2.1).* We argue privacy and accuracy:

*Privacy.* Fix databases $x, x'$, which differ in the $i$-th data item (for $i \in [D]$). Consider a partition $P$. Take $S_j \in P$ s.t. $i \in S_j$. Since the data streams are identical up to $S_j$, the probabilities of generating the prefix $S_1, \ldots, S_{j-1}$ are identical on $x$ and $x'$ (for any choice of random coins made in the first $j-1$ segments, the outcome on both databases is identical). Below, we bound the ratio between the probabilities of generating $S_j = [s_{j-1}, s_j]$ as the $j$-th segment in both runs. After generating $S_j$ as the $j$-th segment, the probabilities of the partition's suffix when running on the two databases are again identical, because the data are identical and no state is carried over (beyond the boundary $s_j$ of the $j$-th segment).

We show a bijection between noise values when running on $x$ and on $x'$, such that for any noise value producing $S_j$ on $x$, the bijection gives a noise value of similar probability that produces the same output on $x'$. We conclude that the probability $p'$ of producing $S_j$ on $x'$ is not much smaller than the probability $p$ of producing $S_j$ on $x$, which implies Differential Privacy.

Towards this, take $\widetilde{T}_j, \widetilde{T}'_j$ be the $j$-th noisy thresholds in a run on $x$ and on $x'$ (respectively), and similarly take $\widetilde{count}_{s_j}$ and $\widetilde{count}'_{s_j}$ to be the noisy counts in runs on $x$ and on $x'$. The bijection is defined as follows:

– For the case $x_i = 0$ and $x'_i = 1$, take:

$$\widetilde{T}'_j = \widetilde{T}_j + 1, \widetilde{count}'_{s_j} = \widetilde{count}_{s_j}$$

All other noise values are unchanged in the two runs. This bijection guarantees that if no item before $s_j$ sealed the $j$-th segment on $x$, then no item before $s_j$ will seal the $j$-th segment on $x'$ (whose *count* can only be larger by at most 1 at any point in the segment). Moreover, if $s_j$ seals the $j$-th segment on $x$, then it will also seal the $j$-th segment on $x'$ (because the noisy threshold there is larger by 1, and *count* at $s_j$ is larger by 1 in $x'$).

– For the case $x_i = 1$ and $x'_i = 0$, take:

$$\widetilde{T}'_j = \widetilde{T}_j, \widetilde{count}'_{s_j} = \widetilde{count}_{s_j} + 1$$

All other noise values are unchanged in the two runs. This bijection guarantees that if no item before $s_j$ sealed the $j$-th segment on $x$, then no item before $s_j$ will seal the $j$-th segment on $x'$ (whose *count* can only be smaller at any point in the segment). Moreover, if $s_j$ seals the $j$-th segment on $x$, then it will also seal the $j$-th segment on $x'$ (because the noisy threshold there is smaller by 1, and *count* at $s_j$ is also smaller by 1 in $x'$).

Since the bijection changed the magnitude of a single draw from $Lap(1/\varepsilon)$ by at most 1, we conclude that $p' \geq e^{-\varepsilon} \cdot p$, and the algorithm is $\varepsilon$-differentially private.

*Accuracy.* By construction, the algorithm makes at most $2D$ draws from the $Lap(1/\varepsilon)$ distribution. By the properties of the Laplace distribution, with all but $\beta$ probability, all of these draws will have magnitude at most $((\log D + \log 2 + \log(1/\beta))/\varepsilon)$. Condition on this event for the remainder of the proof. Under this conditioning, whenever $\widetilde{count} > \widetilde{T}$, we have that *count* (the true count within the segment) is greater than 0, and so all the segments are non-empty, and there can be at most $n$ segments (because there are only $n$ items in the dataset). Moreover, under the above conditioning, as soon as we have $count \geq 5(\log D + \log(1/\beta))/\varepsilon)$, we also have $\widetilde{count} > \widetilde{T}$, and so no segment can have weight larger than $5(\log D + \log(1/\beta))/\varepsilon)$.

## 3 From Partitions to Counting, Intervals and Rectangles

In this section we apply the partition algorithm to obtain improved differentially private mechanisms for online counting, and for answering interval and rectangle queries.

### 3.1 Online Counting under Continual Observation

Counting under continual observation was first studied by [DNPR10], and has emerged as an important primitive with many applications. Given a stream of $D$ data items (integers or boolean values), the goal is to process the items one-by-one. After processing the $i$-th item, the counter outputs an approximation to the sum of items $(1 \dots i)$. Taken together, the counter's $D$ outputs should be differentially private, and mask a change of 1 in any particular data item (flipping a bit if the values are boolean, or adding/subtracting 1 if they are integers). A $(D, \alpha, \beta)$-counter guarantees that with all but $\beta$ probability over its own coins, all $D$ estimates it outputs (simultaneously) have error bounded by $\alpha$.

*Recap: The "Tree Counter".* For privacy and error parameters $\varepsilon, \beta > 0$, "tree counter" of [DNPR10] is an $\varepsilon$-differentially private $(D, O(((\log^2 D) \cdot \log(1/\beta))/\varepsilon), \beta)$ counter: W.h.p., for all $D$ outputs simultaneously, the error is bounded by roughly $(\log^2 D)$. The counter works by building a binary tree over the interval $[1, D]$. Each data item is a leaf in the tree, and each internal node at height $\ell$ (where leaves are at height 0) "covers" a sub-segment of length $2^\ell$. The $(D/2^\ell)$ nodes in height $\ell$ partition the interval $[1, D]$ into sub-segments of length $2^\ell$. The online counter maintains a noisy sum for the items in each internal node (filling up these counts as the items $(1, \dots, D)$ are processed). To estimate the number of items in some segment $[1, k]$, they observe that the segment is exactly covered by at most $\log D$ internal nodes of the tree. The counter outputs the sum of these internal nodes as its estimate. The noise for each internal node is drawn from a Laplace distribution with magnitude $O(\log D/\varepsilon)$, so the sum of noises from the $\log D$ noise values is $O(\log^2 D)$ w.h.p. (the error analysis in [DNPR10] is a bit more slack, see [CSS11]). Privacy follows because any "leaf" (i.e. input element) only affects the counts of the $\log D$ internal nodes that "cover" it.

*Improved Online Counter via Partitions.* We show that the (online) partition algorithm of Section 2 gives an improved online counter (with pure or $(\varepsilon, 0)$ Differential Privacy) for the case of *sparse* streams. In particular, thinking of the input as a boolean string where the number of 1's is at most $n$ (and $n << D$), the error is improved to roughly $(\log D + (\log^2 n))$ (compared with roughly $(\log^2 D)$ for the tree counter). We note that the dependence on $D$ is optimal, and matches the $\Omega(\log D)$ lower bound in [DNPR10]. We note that the counter was conceived for (and is usually applied to) scenarios where $D$ is much larger than $n$.

The improved counter operates by running any online counter (and in particular the tree counter) "on top of" a partition obtained from the (online) partition algorithm. Initializing the count to 0, we process each new data item using the

the partition algorithm. If the algorithm keeps the current segment open, then we simply maintain the current count. If the algorithm seals a segment, then we "feed" that segment into the (online) counter as a new data item (using the true number of 1's in the current segment). We then update the current count using the counter's output. I.e. the segments of the partition now form the "leaves" of the tree used in the [DNPR10] online counter.[7] By differential privacy of the partition algorithm and the counter, the output of this composed algorithm is also differentially private.

**Theorem 3.1.** *Composing the Partition algorithm from Figure 1 with the online tree counter from [DNPR10] gives an online counter. For any $\varepsilon, \beta > 0$, the composed algorithm guarantees $\varepsilon$-differential privacy. Let $n$ be an upper bound on the input stream's weight. With all but $\beta$ probability over the counter's coins, the maximal error over all $D$ items is at most $O(\frac{\log D + ((\log^2 n) \cdot \log(1/\beta))}{\varepsilon})$.*

*Proof.* We run the partition algorithm with privacy parameter $(\varepsilon/2)$ and error parameter $(\beta/2)$. By Theorem 2.1, with all but $(\beta/2)$ probability, the online partition algorithm seals at most $n$ segments, where the (true) number of 1's in each segment is at most $\frac{10(\log D + \log(2/\beta))}{\varepsilon}$. We then run the tree counter on this "stream" of $n$ segments, with privacy parameter $(\varepsilon/2)$ and error parameter $(\beta/2)$. The partition into segments is $(\varepsilon/2)$-DP, and the output of the tree counter on the "stream" of $n$ segments (given the true count in each of these segments) is also $(\varepsilon/2)$-DP. By composition of DP mechanisms, the complete output of the composed mechanism is $\varepsilon$-DP. For accuracy:

1. By the error guarantee of the tree counter, the $n$ counts obtained when segments are sealed have error at most $O(\frac{(\log^2 n) \cdot \log(1/\beta)}{\varepsilon})$ (with all but a $(\beta/2)$ probability of error).
2. By the segment-size guarantee of the partition algorithm, the true count in a "open" segment that hasn't been sealed yet is bounded. Thus, the fact that counts are not updated before a segment is sealed incurs only a $O(\frac{\log D + \log(1/\beta)}{\varepsilon})$ additional (additive) error for the $(D - n)$ items that do not "seal" a segment.

By a union bound, with all but $\beta$ probability, the total error is $O(\frac{\log D + ((\log^2 n) \cdot \log(1/\beta))}{\varepsilon})$.

### 3.2 Interval Queries

To answer interval queries on a database $x \subseteq [1, D]$, we run the partition algorithm and obtain a privacy-preserving partition of $[1, D]$ into (at most) $n$ disjoint segments $(S_1, \ldots, S_n)$, where w.h.p. the count of items in each segment is small. We then construct a binary tree "on top of" these $n$ segments, as in the improved online counter (see Section 3.1). I.e., the $n$ segments are the tree's leaves,

[7] We note that, in general, we could compose *any* online counter with the partition algorithm. We are not using any specific properties of the tree counter.

and each internal node at height $h$ "covers" $2^h$ segments. For each node in this tree, covering an interval $[i, j]$, we add independent Laplace noise of magnitude $(\log n/\varepsilon)$, and release the (noisy) size of the intersection $x \cap [i, j]$ (the number of 1's in the interval). Privacy follows because the partition is DP, and given the partition any data item only changes the counts in $\log n$ nodes of the tree. Note that this offline algorithm can be implemented in time $\mathrm{poly}(n, \log D)$ (see Remark 2.2).

Given the tree of noisy counts, we can answer any interval query $I = [i, j]$ as follows. First, observe that any such interval can be "covered" by at most $2 \log n$ nodes of the tree: a collection of nodes whose (disjoint) leaves form the (minimal) collection of segments whose union contains $I$. To find such a cover, consider the lowest node $k$ in the tree such that the segments its sub-tree cover the interval $I$ (but this is not true for either of $k$'s children). Now the "left" and "right" parts of the interval $I$ are the parts contained in the left or right sub-trees of $k$ (respectively). The left part of $I$ is covered by at most $\log n$ nodes in the left sub-tree, and the right part of $I$ is covered by at most $\log n$ nodes in the right sub-tree. Note that we can also find this cover efficiently. Once the above cover is obtains, we answer the query by simply outputting the sum of (noisy) counts of the nodes that cover the interval. Accuracy follows by the fact that the counts in each segment are small, and the noise in the sum of noisy counts is also small.

**Theorem 3.2 (Theorem 1.2, Restated).** *The mechanism described above answers interval queries. For any privacy and accuracy parameters $\varepsilon, \beta > 0$, the mechanism guarantees $\varepsilon$-differential privacy. For any database $x$ of size $n$, with all but $\beta$ probability over the mechanism's coins, all interval queries (simultaneously) are answered with error $O(\frac{\log D + ((\log^2 n) \cdot \log(1/\beta))}{\varepsilon})$. The running time to produce the synopsis (which can later be used to answer any interval query) is $\mathrm{poly}(n, \log D, \log(1/\varepsilon), \log(1/\beta))$.*

*Proof.* We run the partition algorithm with privacy parameter $(\varepsilon/2)$ and error parameter $(\beta/2)$. By Theorem 2.1, with all but $(\beta/2)$ probability, the online partition algorithm outputs at most $n$ segments, where the (true) number of 1's in each segment is at most $\frac{10(\log D + \log(2/\beta))}{\varepsilon}$. We then build a tree of noisy counts on top of these $n$ segments, adding Laplace noise of magnitude $(2 \log n/\varepsilon)$ to each node's true count, and releasing all of these noisy counts. The partition itself is $(\varepsilon/2)$-DP, and since each data item affects exactly $\log n$ counts in the tree, these noisy counts (taken all together and as a function of the partition) are $(\varepsilon/2)$-DP. Thus, the algorithm's output is altogether $\varepsilon$-DP.For an interval query $I = [i, j]$, we argue accuracy as follows:

The algorithm finds a "minimal cover": A collection of segments $(S_k, \ldots, S_\ell)$ s.t. the union of these segments contains the interval $I$, and (by minimality) the union of $(S_{k+1}, \ldots, S_{\ell-1})$ is *contained in* $I$ (we ignore the borderline cases where $\ell - k \leq 1$, which is handled similarly). Let us denote the union of $(S_k, \ldots, S_\ell)$ by $I'$, so that $I \subseteq I'$. We have that:

1. The (true) sum of items in $I'$ is well approximated by the sum of noisy counts computed by the algorithm. In particular, with all but $(\beta/2)$ probability, the error in computing this sum (a sum of $\log n$ Laplacian RVs) is $O(\frac{(\log^2 n)\cdot\log(1/\beta)}{\varepsilon})$.
2. The difference between the (true) counts in $I'$ and in $I$ is at most the sum of counts in $S_k$ and $S_\ell$. This is because the only items that are in $I'$ but not in $I$ are those in $S_k$ or $S_\ell$ (recall that $I$ contains the union of $(S_{k+1},\ldots,S_{\ell-1})$). By the accuracy of the partition algorithm, with all but $(\beta/2)$ probability, this difference is at most $O(\frac{\log D+\log(1/\beta)}{\varepsilon})$

By a union bound, we conclude that with all but $\beta$ probability, the total error in computing the count on interval $I$ is $O(\frac{\log D+((\log^2 n)\cdot\log(1/\beta))}{\varepsilon})$.

### 3.3 Rectangle Queries

To answer $c$-dimensional rectangle queries on a database $x \subseteq [1,D]^c$, we run the partition algorithm on each "axis" of the input space separately. For each dimension $a \in [1,c]$, we partition the line $[1,D]$ into (at most) $n$ segments, where for each of these segments, the number of input elements whose $a$-th coordinate falls into that segment is bounded. I.e., we compute a privacy-preserving partition $(S^a_1,\ldots,S^a_n)$, where for all $i$, the number of database elements whose $a$-th coordinate falls into $S^a_i$ is bounded. For the remainder of the construction, we will consider the partition of the multi-dimensional space $[1,D]^c$ into a collection of rectangles:

$$\{(S^1_{i_1}\times\ldots\times S^c_{i_c})\}_{i_1,\ldots,i_c\in[1,n]}.$$

By the properties of the partition algorithm, these rectangles are disjoint and cover the input space.

*Multi-Dimensional Tree.* We construct a "multi-dimensional tree" of counts over the above partition. The construction is iterative, proceeding one dimension at a time from 1 to $c$:

– The dimension-1 tree is a binary tree, whose leaves are the segments $\{S^1_i\}_{i\in[1,n]}$ (as in the intervals algorithm). Each node of this dimension-1 tree corresponds to an interval $T^1$, a union of some number (a power of 2) of segments $\{S^1_i\}$ from the dimension-1 partition. Each such node contains a noisy count for the number of items whose first coordinate falls in the interval $T^1$. The node also contains a dimension-2 tree, which we call its "successor".
– For $a \in [2,c]$ each dimension-$a$ tree is a binary tree whose leaves are the segments $\{S^a_i\}_{i\in[1,n]}$. The dimension-$a$ tree has a "predecessor", a dimension-$(a-1)$ tree, corresponding to intervals $(T^1,\ldots,T^{a-1})$ in the first $(a-1)$ dimensions.
  Each node in the dimension-$a$ tree corresponds to an interval $T^a$, a union of some number (a power of 2) of segments $\{S^a_i\}$ from the dimension-$a$ partition. Each such node contains a noisy count for the number of items

s.t. for all $i \in [1, a]$, their $i$-th coordinate falls in $T^i$. For $a < c$ (until the "final" dimension), each node also contains a dimensional-$(a+1)$ tree, which we call its "successor".

For privacy parameter $\varepsilon$, the noise added to each count is drawn from a Laplace random variable with magnitude $(4 \log^c n/\varepsilon)$. We view each node in the above construction as specifying a $c$-dimensional rectangle $T = (T^1 \times \ldots \times T^c)$ (for nodes in dimension-$a$ trees where $a < c$, the intervals $(T^{a+1}, \ldots, T^c)$ are "full" and equal $[1, D]$). Each such node contains a noisy count of the number of input elements that fall into this rectangle, i.e. of $|x \cap T|$. The size of this data structure is roughly $n^c$. By pruning this tree, removing nodes with small noisy counts (and their successors), we can obtain a data structure of size $O(n \cdot \log^c n)$ (whose construction also requires time $O(n \cdot \log^c n)$), see Remark 3.4 below.

The following two claims will be used in arguing privacy and accuracy:

*Claim.* Adding or removing an element to the dataset only changes the counts of at most $(2 \log^c n)$ nodes in the multi-dimensional tree.

*Proof.* Let $x_j \in [1, d]^c$ be a data item. Let $(S^1_{i_1}, \ldots, S^n_{i_n})$ be the (unique) segments of the partition s.t. the $a$-th coordinate of $x_j$ is in $S^a_{i_a}$ (for all $a \in [1, c]$). We bound the number of nodes in the tree for which their corresponding rectangle $T$ includes $x_j$ (adding or removing $x_j$ will not affect the counts in any other nodes). In the dimension-1 tree there are only $\log n$ such nodes: the leaf corresponding to the segment $S^1_{i_1}$, and its ancestors in the tree. Now observe that for the other nodes in the dimension-1 tree, their successors (and their successors) will never correspond to rectangles that include $x_j$. For the $\log n$ nodes that do include $x_j$, their successors are dimension-2 trees, and they each have $\log n$ nodes that include $x_j$. Thus, we have $\log^2 n$ nodes in dimension-2 trees that include $x_j$. For all other nodes, their successors will not include $x_j$. Continuing as above, we have that in the dimension-$a$ trees there are $\log^a n$ nodes that include $x_j$. We conclude that in total, the number of nodes in the multi-dimensional tree that include $x_j$ is bounded by:

$$\sum_{a=1}^{c} \log^a n \leq 2 \log^c n$$

*Claim.* For any rectangle $R = (R^1 \times \ldots \times R^c) \subseteq [1, D]^c$, there exists a tight "covering" of that rectangle using a set of at most $m = (2 \log n)^c$ nodes $\mathcal{T} = \{T_1, \ldots, T_m\}$ from the multi-dimensional tree. Taking $Q = \bigcup_i T_i$ we have:

1. $R$ is no larger than $Q$, in particular $R \subseteq Q$.
2. $Q$ is not "much" larger than $R$. In particular, for each dimension $a$ there exist segments $S^a_j, S^a_k$ (segments of the $a$-th partition) s.t. for any element in $y \in (Q \setminus R)$ for some $a \in [1, c]$ the $a$-th coordinate of $y$ is in either $S^a_j$ or $S^a_k$ (and thus, by the properties of the partition algorithm, the size of $(Q \setminus R)$ is not too large).

*Proof.* Similarly to the intervals algorithm, we begin with a separate "cover" for the intervals that constitute each dimension of the rectangle $R$. As in the intervals algorithm, for each dimension $a \in [1, c]$, there exists a collection $\mathcal{T}^a$ of $2 \log n$ intervals corresponding to nodes in the dimension-$a$ tree that "cover" the interval $R^a$ as follows. Taking $Q^a = \bigcup_{T \in \mathcal{T}^a} T$:

1. $R^a \subseteq Q^a$.
2. There exist two segments $(S^a_j, S^a_k)$ of the $a$-th partition, s.t. $(Q^a \backslash (S^a_i \bigcup S^a_j)) \subseteq R^a$

Now the claim follows by taking $\mathcal{T}$, the set of tree nodes, to be $\mathcal{T} = (\mathcal{T}^1 \times \ldots \times \mathcal{T}^c)$. This is a set of at most $(2 \log n)^c$ nodes, as required. Moreover, taking:

$$Q = \bigcup_{T \in \mathcal{T}} T = (Q^1 \times \ldots \times Q^c),$$

by the above properties of the cover on each dimension separately, we get that:

$$R = (R^1 \times \ldots \times R^c) \subseteq (Q^1 \times \ldots \times Q^c) = (\bigcup_{T \in \mathcal{T}} T) = Q.$$

Moreover, for each dimension $a$ we denote $Q'_a = (Q^a \setminus (S^a_j \bigcup S^a_k))$. We have that $Q' = Q'_1 \times \ldots \times Q'_c$ has the properties that $Q' \subseteq R$, and for every element $y \in (Q \setminus Q')$, for some $a \in [1, c]$, its $a$-th coordinate is in $(S^a_j \bigcup S^a_k)$.

*Answering Rectangle Queries.* We use the multi-dimensional tree of noisy counts described above to answer rectangle queries. Given a rectangle query $R = (R^1 \times \ldots \times R^c) \subseteq [1, D]^c$, we decompose it into a "cover" $\mathcal{T}$ of $(2 \log n)^c$ tree nodes as promised in Claim 3.3. We answer the query $R$ by adding up the noisy counts for the these $m$ nodes and outputting this noisy sum. This can be done in time poly$(\log^c n)$.

**Theorem 3.3 (Theorem 1.3, restated).** *The mechanism described above answers c-dimensional rectangle queries. For any privacy and accuracy parameters $\varepsilon, \beta > 0$, the mechanism guarantees $\varepsilon$-differential privacy. With all but $\beta$ probability over its coins, all rectangle queries (simultaneously) are answered with error $O(\frac{(c^2 \cdot \log D) + (c \cdot (2 \log n)^{1.5c+1} \cdot \log(1/\beta))}{\varepsilon})$.*

*Proof.* By composition of DP mechanisms, privacy follows directly from: $(i)$ privacy of the Partition algorithm (for computing the $c$ partitions), and $(ii)$ from Claim 3.3 and the fact that we add Laplace noise of magnitude $(4 \log^c n / \varepsilon)$ to each count.

For accuracy, observe that after we partition the axis, there are $n^{2c}$ possible rectangle queries (rectangles whose covers are identical are essentially equivalent). For each such query $R$, we release a noisy count for its cover $\mathcal{T}$. The noise is a sum of (at most) $(2 \log n)^c$ independent Laplace RVs, each of magnitude $2 \log^c n$. With all but $(\beta/2)$ probability, the maximal noise added to the count of any of these covers is of magnitude at most $O(\frac{c\cdot(2\log n)^{1.5c+1}\cdot\log(1/\beta)}{\varepsilon})$ (see the analysis for the sum of Laplacian RVs in [CSS11]). So for each rectangle $R$ with cover $\mathcal{T}$, the error in the noisy count for $\mathcal{T}$ is bounded.

We run the partition algorithm $c$ times, each with privacy parameter $(\varepsilon/2c)$ and error parameter $(\beta/2c)$. With all but $(\beta/2c)$ probability, the size of each segment in each of the $c$ partitions is at most $O(\frac{c\cdot(\log D+\log c+\log(1/\beta))}{\varepsilon})$. Every point that is in $\mathcal{T}$ but not in $R$ must have one of its coordinates be in a (fixed) set of $2c$ such segments. Thus, by the second property of the cover $\mathcal{T}$ (see Claim 3.3), the difference between the true counts of $R$ and of $\mathcal{T}$ is at most $O(\frac{c^2\cdot(\log D+\log c+\log(1/\beta))}{\varepsilon})$. The error bound follows by a triangle inequality (and a union bound).

*Remark 3.4.* The naive construction of the multi-dimensional tree requires time (and size) $n^c$. We improve this running time dramatically by judiciously "pruning" the tree. We take a threshold $t = O((\log n)^{c+1}\cdot\log(1/\beta))$, and as we construct the multi-dimensional tree (starting with the dimension-1 tree), for any node whose noisy count is smaller than $t$, we set that node to be "empty" (noisy count 0), and do not continue to its children in the current tree, nor to its successor. By this choice of $t$, w.h.p. over the noise, any node that is *not* marked as empty corresponds to a rectangle that is not empty in the input database.

Now when using the noisy counts to reconstruct the answers to a given rectangle, because we might be under-counting for all $(2\log n)^c$ of the nodes that we use to "cover" the query, we obtain a slightly-larger error of $((\log n)^{O(c)}\cdot\log(1/\beta))$. The advantage, however, is that the running time and the size of the multi-dimensional tree are improved to $O(n\cdot\log^c n)$. To see this, recall that any node that is not marked as "empty" must have at least 1 data item in its corresponding rectangle. The bound on the tree size follows by induction over $c$ (as does the improved running time).

## References

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[BLR08] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *STOC*, 2008.

[BNS13] Amos Beimel, Kobbi Nissim, and Uri Stemmer. Private learning and sanitization: Pure vs. approximate differential privacy. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, pages 363–378, 2013.

[BNSV15] Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil P. Vadhan. Differentially private release and learning of threshold functions. *CoRR*, abs/1504.07553, 2015.

[BUV14] Mark Bun, Jonathan Ullman, and Salil Vadhan. Fingerprinting codes and the price of approximate differential privacy. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM, 2014.

[CSS11] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3):26, 2011.

[DKM$^+$06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006*, pages 486–503. Springer, 2006.

[DMNS06] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[DN03] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *PODS*, pages 202–210, 2003.

[DN04] C. Dwork and K. Nissim. Privacy-preserving datamining on vertically partitioned databases. In *CRYPTO*, volume 3152, pages 528–544, 2004.

[DNPR10] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 715–724, 2010.

[DNR$^+$09] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *STOC*, pages 381–390, 2009.

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[HR10] M. Hardt and G. N. Rothblum. A multiplicative weights mechanism for interactive privacy-preserving data analysis. *FOCS*, 2010.

[MN12] S. Muthukrishnan and Aleksandar Nikolov. Optimal private halfspace counting via discrepancy. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1285–1292, 2012.

[Ull13] Jonathan Ullman. Answering n {2+ o (1)} counting queries with differential privacy is hard. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 361–370. ACM, 2013.