Optimal Differentially Private Learning of Thresholds and Quasi-Concave Optimization

Optimal Differentially Private Learning of Thresholds and Quasi-Concave Optimization*

Edith Cohen
edith@cohenwang.com
Google Research and Tel Aviv University
Mountain View, California, USA

Xin Lyu
xinlyu@berkeley.edu
UC Berkeley and Google Research
Berkeley, California, USA

Jelani Nelson
minilek@alum.mit.edu
UC Berkeley and Google Research
Berkeley, California, USA

Tamás Sarlós
stamas@google.com
Google Research
Mountain View, California, USA

Uri Stemmer
u@uri.co.il
Tel Aviv University and Google Research
Tel Aviv-Yafo, Israel

ABSTRACT

The problem of learning threshold functions is a fundamental one in machine learning. Classical learning theory implies sample complexity of $O(\xi^{-1}\log(1/\beta))$ (for generalization error ξ with confidence $1-\beta$). The private version of the problem, however, is more challenging and in particular, the sample complexity must depend on the size $|X|$ of the domain. Progress on quantifying this dependence, via lower and upper bounds, was made in a line of works over the past decade. In this paper, we finally close the gap for approximate-DP and provide a nearly tight upper bound of $\widetilde{O}(\log^* |X|)$, which matches a lower bound by Alon et al (that applies even with improper learning) and improves over a prior upper bound of $\widetilde{O}((\log^* |X|)^{1.5})$ by Kaplan et al. We also provide matching upper and lower bounds of $\tilde{\Theta}(2^{\log^* |X|})$ for the additive error of private quasi-concave optimization (a related and more general problem). Our improvement is achieved via the novel Reorder-Slice-Compute paradigm for private data analysis which we believe will have further applications.

CCS CONCEPTS

• **Theory of computation → Sample complexity and generalization bounds**; • **Security and privacy**;

KEYWORDS

differential privacy, PAC learning, threshold functions

ACM Reference Format:
Edith Cohen, Xin Lyu, Jelani Nelson, Tamás Sarlós, and Uri Stemmer. 2023. Optimal Differentially Private Learning of Thresholds and Quasi-Concave Optimization. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3564246.3585148

*Full version of this paper is available at https://arxiv.org/abs/2211.06387

STOC '23, June 20–23, 2023, Orlando, FL, USA

ACM ISBN 978-1-4503-9913-5/23/06.
https://doi.org/10.1145/3564246.3585148

1 INTRODUCTION

Motivated by the large applicability of learning algorithms to settings involving personal individual information, Kasiviswanathan et al. [18] introduced the model of *private learning* as a combination of *probably approximately correct (PAC)* learning [22, 23] and *differential privacy* [10]. For our purposes, we can think of a (non-private) learner as an algorithm that operates on a *training set* containing labeled random examples (from some distribution over some domain X), and outputs a hypothesis h that misclassifies fresh examples with probability at most (say) $\frac{1}{10}$. It is assumed that the "true" classification rule, which is unknown to the learner, is taken from a (known) class C of possible classification rules, where intuitively, learning becomes "harder" as the class C becomes "richer". A *private learner* must achieve the same goal while guaranteeing that the choice of h preserves *differential privacy* of the training set. This means that the choice of h should not be significantly affected by any particular labeled example in the training set. Formally, the definition of differential privacy is as follows.

Definition 1 ([10]). Let $\mathcal{A} : X^* \to Y$ be a randomized algorithm whose input is a dataset $D \in X^*$. Algorithm $\mathcal{A}$ is (ε, δ)*-differentially private (DP)* if for any two datasets D, D' that differ on one point (such datasets are called *neighboring*) and for any outcome set $F \subseteq Y$ it holds that $\Pr[\mathcal{A}(D) \in F] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(D') \in F] + \delta$.

Since its inception, research on the private learning model has largely focused on understanding the *amount of data* that is needed in order to achieve both the privacy and the utility goals simultaneously (a.k.a. the *sample complexity of private learning*). The holy grail in this line of research is to come up with a (meaningful) combinatorial measure that given a class C characterizes the sample complexity of privately learning C. However, after almost 15 years of intensive research, this question is still far from being well-understood. At a high level, works on the sample complexity of private learning can be partitioned into two meta approaches:

1. **Deriving generic upper and lower bounds (as a function of the class C).** This avenue has produced several fascinating results, that relate the sample complexity of private learning to the *Littlestone dimension* of the class C, a combinatorial dimension that is known to characterize *online learnability* (non-privately) [1]. However, the resulting

bounds are *extremely* loose (exhibiting a tower-like gap between them). Furthermore, it is known that, in general, this is the best possible in terms of the Littlestone dimension alone.

2. **Focusing on specific test-cases, squeezing them until the end to reveal structure.** This avenue has produced several fascinating techniques that has found many applications, even beyond the scope of private learning. Arguably, the most well-studied test-case is that of *one dimensional threshold functions*, where the class C contains all functions that evaluate to 1 on a *prefix* of the (totally ordered) domain X.[1] Even though this class is trivial to learn without privacy considerations, in the private setting it is surprisingly complex. The sample complexity of privately learning threshold functions has been studied in a sequence of works [2, 4, 6–9, 12, 16], producing many interesting tools and techniques that are applicable much more broadly.

In this work we present new tools and proof techniques that allow us to obtain a *tight* upper bound on the sample complexity of privately learning threshold functions (up to lower order terms). This concludes a long line of research on this problem. In addition, we present matching upper and lower bounds for the related problem of *private quasi-concave optimization*. Before presenting our new results, we survey some of the progress that has been made on these questions.

1.1 On Our Current Understanding of Privately Learning Threshold Functions

Early works on the sample complexity of private learning focused on the case where the privacy parameter δ is set to zero, known as the *pure private* setting. While this significantly limits the applicability of the model, the pure-private setting is often much easier to analyze. Indeed, already in the initial work on private learning, Kasiviswanathan et al. [18] presented a generic bound of $O(\log|C|)$ on the sample complexity of learning a class C with pure privacy.[2] This implies an upper bound of $O(\log|X|)$ on the sample complexity of privately learning threshold functions over an ordered domain X (because $|C| = |X|$ for this class). Beimel et al. [4] presented a matching lower bound for *proper* pure-private learners (these are learners whose output hypothesis must itself be a threshold function). Feldman and Xiao [12] then showed that this lower bound also holds for pure-private *improper* learners.

The sample complexity of privately learning thresholds in the more general setting, where δ is not restricted to be zero (known as *approximate privacy*), was studied by Beimel at al. [6], who showed an improved upper bound of $\tilde{O}\left(8^{\log^*|X|}\right)$ on the sample complexity. This is a dramatic improvement in asymptotic terms over the pure-private sample complexity (which is $\Theta(\log|X|)$), coming tantalizingly close to the non-private sample complexity of this problem (which is constant, independent of $|X|$). Interestingly, to obtain this result, Beimel at al. [6] introduced a tool for *privately optimizing quasi-concave functions* (to be surveyed next), a generic tool which has since found many other applications. Bun et al. [8] then presented a different approximate-private learner with improved sample complexity of $\tilde{O}\left(2^{\log^*|X|}\right)$, and another different construction with similar sample complexity was presented by [7]. Bun et al. [8] also showed a lower bound of $\Omega(\log^*|X|)$ that holds for any (approximate) private *proper*-learner for thresholds. Alon et al. [2] then proved a lower bound of $\Omega(\log^*|X|)$ that holds even for *improper* learners for thresholds. Finally, a recent work of Kaplan et al. [16] presented an improved algorithm with sample complexity $\tilde{O}((\log^*|X|)^{1.5})$.

To summarize, our current understanding of the task of privately learning thresholds places its sample complexity somewhere between $\Omega(\log^*|X|)$ and $\tilde{O}((\log^*|X|)^{1.5})$.

1.2 Privately Optimizing Quasi-Concave Functions

Towards obtaining their upper bound for privately learning thresholds, Beimel et al. [6] defined a family of optimization problems, called *quasi-concave optimization problems*. The possible solutions are ordered, and quasi-concavity means that if two solutions $x \le z$ have quality of at least q, then any solution $x \le y \le z$ also has quality of at least q. The optimization goal is to find a solution with (approximately) maximal quality. Beimel et al. [6] presented a private algorithm for optimizing such problems, guaranteeing additive error at most $\tilde{O}\left(8^{\log^* T}\right)$, where T is the number of possible solutions. They observed that the task of learning thresholds can be stated as a quasi-concave optimization problem, and that this yields a private algorithm for thresholds over a domain X with sample complexity $\tilde{O}\left(8^{\log^*|X|}\right)$. Since the work of Beimel et al. [6], quasi-concave optimization was used as an important component for designing private algorithms for several other problems, including geometric problems [5, 13], clustering [11, 20], and privately learning halfspaces [5, 17].

We stress that later works on privately learning thresholds (following [6]) did not present improved tools for quasi-concave optimization (instead they worked directly on learning thresholds). As quasi-concave optimization generalizes the task of learning thresholds (properly), the lower bound of [8] also yields a lower bound of $\Omega(\log^* T)$ on the additive error of private algorithms for quasi-concave optimization. That is, our current understanding of private quasi-concave optimization places its additive error somewhere between $\Omega(\log^* T)$ and $\tilde{O}\left(8^{\log^* T}\right)$. An improved upper bound would imply improved algorithms for all of the aforementioned applications, and a stronger lower bound would mean an inherent limitation of the algorithmic techniques used in these papers.

1.3 Our Contributions

For all informal theorems presented in this section, readers can find their formal statements in the full version of our paper.

[1] Let $X \subseteq \mathbb{R}$. A threshold function f over X is specified by an element $u \in X$ so that $f(x) = 1$ if $x \le u$ and $f(x) = 0$ for $x > u$. In the corresponding learning problem, we are given a dataset containing labeled points from X (sampled from some unknown distribution $\mathcal{D}$ over X and labeled by some unknown threshold function f^*), and our goal is to output a hypothesis $h : X \to \{0,1\}$ such that $\mathrm{error}_{\mathcal{D}}(h, f^*) \triangleq \Pr_{x\sim\mathcal{D}}[h(x) \neq f^*(x)]$ is small.

[2] To simplify the exposition, in the introduction we omit the dependency of the sample complexity in the utility and privacy parameters.

Our main result is presenting a private algorithm for learning thresholds, with *optimal* sample complexity (up to lower order terms):

Theorem 1.1 (Informal)**.** *There is an approximate private algorithm for (properly) learning threshold functions over an ordered domain X with sample complexity $\widetilde{O}(\log^* |X|)$.*

This improves over the previous upper bound of $\widetilde{O}\left((\log^* |X|)^{1.5}\right)$ by [16], and matches the lower bound of $\Omega(\log^* |X|)$ by [2, 8] (up to lower order terms). This concludes a long line of research aimed at understanding the sample complexity of this basic problem. A key to our improvement is a novel paradigm, which we refer to as the *Reorder-Slice-Compute* paradigm (to be surveyed next), allowing us to simplify both the algorithm and the analysis of [16].

Inspired by our simplified algorithm for thresholds, we design a new algorithm for private quasi-concave optimization with an improved error of $\tilde{O}\left(2^{\log^* T}\right)$, a polynomial improvement over the previous upper bound of $\tilde{O}\left(8^{\log^* T}\right)$ by [6].

Theorem 1.2 (Informal)**.** *There exists an approximate-private algorithm for quasi-concave optimization with additive error $\widetilde{O}(2^{\log^* T})$, where T is the number of possible solutions.*

As we mentioned, this immediately translates to improved algorithms for all of the applications of private quasi-concave optimization. Given the long line of improvements made for the related task of privately learning thresholds (culminating in Theorem 1.1), one might guess that similar improvements could be achieved also for private quasi-concave optimization, hopefully reaching error linear or polynomial in $\log^* T$. Surprisingly, we show that this is not the case, and present the following *tight* lower bound (up to lower order terms).

Theorem 1.3 (Informal)**.** *Any approximate-private algorithm for quasi-concave optimization must have an additive error at least $\widetilde{\Omega}(2^{\log^* T})$, where T is the number of possible solutions.*

We view this lower bound as having an important conceptual message, because private quasi-concave optimization is the main workhorse (or more precisely, the only known workhorse) for several important tasks, such as privately learning (discrete) halfspaces [5, 17]. As such, current bounds on the sample complexity of privately learning halfspaces are exponential in $\log^* |X|$, but it is conceivable that this can be improved to a polynomial or a linear dependency. The lower bound of Theorem 1.3 means that either this is not true, or that we need to come up with fundamentally new algorithmic tools in order to make progress w.r.t. halfspaces.

1.3.1 The Reorder-Slice-Compute paradigm. Towards obtaining our upper bounds, we introduce a simple, but powerful, paradigm which we call the *Reorder-Slice-Compute (RSC)* paradigm. For presenting this paradigm, let us consider the following algorithm (call it algorithm $\mathcal{B}$) that is instantiated on an input dataset D, and then for $\tau \in \mathbb{N}$ rounds applies a DP algorithm on a "slice" of the dataset.

(1) Take an input dataset $D \in X^n$ containing n points from some domain X.
(2) For round $i = 1, 2, \ldots, \tau$:
  (a) Obtain an integer m_i, an (ε, δ)-DP algorithm $\mathcal{A}_i$ and an ordering $\prec^{(i)}$ over X.
  (b) $S_i \leftarrow$ the largest m_i elements in D under $\prec^{(i)}$.
  (c) $D \leftarrow D \setminus S_i$.
  (d) $r \leftarrow \mathcal{A}(S_i)$.
  (e) Output r.

As $\mathcal{B}$ performs a total of τ applications of (ε, δ)-DP algorithms, standard composition theorems for DP state that algorithm $\mathcal{B}$ itself is $\approx (\varepsilon\sqrt{\tau}, \delta\tau)$-DP. This analysis, however, seems wasteful at first glance, because each $\mathcal{A}_i$ is applied on a *disjoint* portion of the input dataset D. That is, the (incorrect) hope here is that we do not need to pay in composition since each data point from D is "used only once". The failure point of this idea is that by deleting *one* point from the data, we can create a "domino effect" that effects (one by one) many of the sets S_i throughout the execution. This is illustrated in the following example.

Example 1.4. *Suppose that $X = \mathbb{N}$, and that $m_1 = \cdots = m_\tau = m$ (for some parameter m), and that all of the orderings $\prec^{(1)}, \ldots, \prec^{(\tau)}$ are the standard ordering of the natural numbers. Now consider the two neighboring[3] datasets $D = \{1, 2, 3, 4, 5, ...n\}$ and $D' = D \setminus \{1\}$. Then during the execution on D we have that $S_1 = \{1, 2, \ldots, m\}, S_2 = \{m+1, \ldots, 2m\}$, and so on, while during the execution on D' we have that $S'_1 = \{2, \ldots, m+1\}, S'_2 = \{m+2, \ldots, 2m+1\}$, and so on. That is, even though D and D' differ in only one point, and even though this point is "used only once", it generates differences in the output distribution of* all *of the iterations, and hence, does not allow us to avoid paying in composition.*

A natural idea for trying to tackle this issue, which has been contemplated by several previous papers, is to add noise to the size of each slice [8, 16, 21]. Specifically, the modification is that in Step 2b of algorithm $\mathcal{B}$ we let S_i denote the largest $(m_i + \text{Noise})$ elements (for some appropriate noise distribution), instead of the largest m_i elements. The hope is that these noises would "mask" the domino effect mentioned above. Indeed, in Example 1.4, if during the first iteration of the execution on D the noise is bigger by one than the corresponding noise during the execution on D', then we would have that only S_1 and S'_1 differ by one point (the point 1), and after that the two executions continue identically. Thus, the hope is that by correctly "synchronizing" the noises between the two executions (such that only the size of the "correct" set S_i gets modified by 1), we can make sure that only one iteration is effected, and so we would not need to apply composition arguments.

Although very intuitive, analyzing this idea is not straightforward. The subtle issue here is that it is not clear how to synchronize the noises between the two executions. In fact, this appeared in

[3] In this paper, we use the "add-one" definition for neighboring. Namely, we say D and D' are a pair of neighboring data sets if D' can be obtained by adding or removing one item in D.

several papers as an open question.[4] Furthermore, this issue (almost) exactly describes the bottleneck in the algorithm of [16] for privately learning thresholds, capturing the reason for why their algorithm had sample complexity $\tilde{O}\left((\log^* |X|)^{1.5}\right)$. We analyze this algorithm, and present the following result.

Theorem 1.5 (Informal)**.** *For every $\hat{\delta} > 0$, the RSC paradigm, as described in algorithm $\mathcal{B}$ above (with appropriate noises of magnitude $\approx \frac{1}{\varepsilon}$), is $(O(\varepsilon \log(1/\hat{\delta})), \hat{\delta} + 2\tau\delta)$-DP.*

Note that the privacy parameter ε does *not* deteriorate with τ, as it would when using standard composition theorems. This benefit is what, ultimately, allows us to present our improved algorithms for privately learning thresholds and for quasi-concave optimization. As the Reorder-Slice-Compute paradigm looks generic, we hope that it would find additional applications in future work.

1.3.2 A simulation-based proof technique. Towards analyzing our RSC paradigm, we put forward a new proof technique. While obvious in retrospect, and related to prior simulation-based approaches used for proving composition theorems for differential privacy [15, 19], we believe that our formulation of this proof technique is instructive.

Consider an algorithm $\mathcal{A}$ whose input is a dataset, and suppose that we would like to prove that $\mathcal{A}$ is DP. To do this, in the proof technique we propose, we design two interactive algorithms: a *simulator* $\mathcal{S}$ and a *data holder* H with the following properties. The simulator is given two neighboring datasets D^0 and D^1 but does not know which of these two datasets is the actual input. The task of the simulator is to simulate the computation of $\mathcal{A}$ on the actual input dataset D^b. The *data holder* has, in addition to D^0, D^1, access to the private bit b (and therefore knows the identity of the actual dataset D^b). The simulator attempts to perform as much of the computation as they can without accessing the data holder. That is, ideally, the data holder is queried only when it is necessary for a faithful simulation of $\mathcal{A}$ on D^b.

The privacy cost of the simulation is with respect to the leakage of the private bit b during the interaction between the simulator and the data holder. Formally,

Lemma 1.6. *Let $\mathcal{A}$ be an algorithm whose input is a dataset. If there exist a pair of interactive algorithms $\mathcal{S}$ and H satisfying the following 2 properties, then algorithm $\mathcal{A}$ is (ε, δ)-DP.*

1. *For every two neighboring datasets D^0, D^1 and for every bit $b \in \{0,1\}$ it holds that*
$$\left(\mathcal{S}(D^0, D^1) \leftrightarrow H(D^0, D^1, b)\right) \equiv \mathcal{A}(D^b).$$
Here $\left(\mathcal{S}(D^0, D^1) \leftrightarrow H(D^0, D^1, b)\right)$ denotes the outcome of $\mathcal{S}$ after interacting with H.
2. *Algorithm H is (ε, δ)-DP w.r.t. the input bit b.*

The proof of this lemma is immediate. Nevertheless, embracing its terminology can simplify privacy proofs. The potential benefit comes from the fact that in order to prove that $\mathcal{A}$ is DP, we design two other algorithms that are "working together" in order to simulate $\mathcal{A}$, *under the assumption that both of them know the two neighboring datasets*, where H is trying to "steer" $\mathcal{S}$ towards simulating $\mathcal{A}(D^b)$.

Let us elaborate on the benefits of this proof technique in the context of our RSC paradigm (specified in algorithm $\mathcal{B}$ above, with noisy slice sizes m_i). Fix two neighboring datasets D, D'. We design a simulator that, in every iteration $i \in [\tau]$, samples the noisy slice size m_i, and checks if the resulting slices S_i, S'_i (corresponding to D, D') are identical. If so, then the simulator does not need to access the data holder, and therefore does not incur a privacy cost. When the simulator encounters a step where $S_i \neq S'_i$, it calls the data holder to perform the computation. When called, in addition to doing the computation, the data holder also attempts to "synchronize" the two executions, and reports back to $\mathcal{S}$ if it succeeded. Once synchronization is successful, the simulator can proceed without further assistance from the data holder, and no more privacy cost is incurred. We show that, when done correctly, the number of iterations in which we incur a privacy cost is constant in expectation and with probability at least $1 - \hat{\delta}$ it is at most $O(\log(1/\hat{\delta}))$.

1.3.3 Our new upper bound for privately learning thresholds. To obtain our (nearly tight) upper bound on the sample complexity of privately learning thresholds, we present a new analysis (and a simplification) of the algorithm of [16], which is made possible using our new RSC paradigm. We next survey the algorithm of [16] and explain our improvements. We stress that this presentation is oversimplified. Any informalities made herein will be removed in the sections that follow.

The interior point problem [8, 16]. Rather than directly designing an algorithm for learning thresholds, the algorithms of [8, 16] (as is ours) are stated for the simpler *interior point problem*: Given a dataset D containing (unlabeled) elements from an ordered domain X, the interior point problem asks for an element of X between the smallest and largest elements in D. Formally,

Definition 2. An algorithm $\mathcal{A}$ solves the interior point problem (IP) over a domain X with sample complexity n and failure probability β if for every dataset $D \in X^n$,
$$\Pr[\min D \leq \mathcal{A}(D) \leq \max D] \geq 1 - \beta,$$
where the probability is taken over the coins of $\mathcal{A}$.

Note that this problem is trivial without privacy constraints (as any input point is a valid output). Nevertheless, solving it with differential privacy has proven to be quite challenging. In particular, as Bun et al. [8] showed, privately solving this problem is *equivalent* to privately learning thresholds (properly).[5] Thus, all of the aforementioned upper and lower bounds w.r.t. thresholds apply also to the IP problem, and it suffices to study this simpler problem in order to present upper and lower bounds for privately learning thresholds (properly).

[4] We remark that the analysis of algorithm $\mathcal{B}$ (with the noises) becomes significantly easier when all the orderings throughout the execution are the same (as in the setting of Example 1.4). The more general setting (with different orderings) is more challenging, and it is necessary for our applications. We refer the reader to [21] for a more elaborate discussion.

[5] This equivalence is very simple: Given a private algorithm for the IP problem, we can use it to learn thresholds by identifying an interior point of the input points that reside around the decision boundary. For the other direction, given an unlabeled dataset (an instance to the IP problem), sort it, label the first half of the points as 1 and the other half as 0, and use a private algorithm for thresholds in order to identify a decision boundary. This decision boundary is a valid output for the IP problem.

The algorithm of [16]. Kaplan et al. [16] presented an algorithm, called TreeLog, for privately solving the IP problem. At a high level, TreeLog works by embedding the input elements from the domain X in a smaller domain of size $\log|X|$, while guaranteeing that every interior point of the embedded elements can be (privately) translated into an interior point of the input elements. The algorithm is then applied recursively to identify an interior point of the embedded elements. TreeLog can be informally (and inaccurately) described as follows.

Input: Dataset $D \in X^n$ containing n points from the ordered domain X.

(1) Let T be a binary tree with $|X|$ leaves, where every leaf is identified with an element of X.
(2) For a *trimming parameter* $t \approx \frac{1}{\varepsilon_0}\log\frac{1}{\delta}$, let D_{left} and D_{right} denote the t smallest and t largest elements in D, respectively. Let $\hat{D} = D \setminus (D_{\text{left}} \cup D_{\text{right}})$.
(3) Assign *weights* to the nodes of T, where the *weight* of a node u is the number of input points (from $\hat{D}$) that belong to the subtree of T rooted at u.
(4) Identify a path π from the root of T to a node u_π with weight t (in a very particular way).
(5) Use the path π to embed the input points in a domain of size $\log|X|$, where a point $x \in \hat{D}$ is mapped to the *level of the tree* T at which it "falls off" the path π. That is, x is mapped to the level of the last node u in π s.t. x belongs to the subtree rooted at u. Points belonging to the subtree rooted at u_π (the last node in π) are not embedded (there are t such points).
(6) Recursively identify an interior point $\ell^* \in [\log|X|]$ of the $(n-3t)$ embedded points.
(7) Let u^* be the node at level ℓ^* of π. Privately choose between the left-most and the right-most descendants of u^*; one of them is an interior point w.r.t. the dataset $D_{\text{left}} \cup D_{\text{right}}$.

To see that the algorithm returns an interior point, suppose that (by induction) the point ℓ^* from Step 6 is indeed an interior point of the embedded points. This means that at least one embedded point is smaller than ℓ^* and at least one embedded point is larger than ℓ^* (for simplicity we ignore here the case where these points might be *equal* to ℓ^*). This means that at least one input point $x_{\text{before}} \in \hat{D}$ falls off the path π *before* level ℓ^* and at least one input point $x_{\text{after}} \in \hat{D}$ falls off the path π *after* level ℓ^*. Observe that since x_{before} falls off π before level ℓ^*, is does *not* belong to the subtree rooted at u^* (the node at level ℓ^* of π). On the other hand, x_{after}, which falls off π after level ℓ^*, *does* belong to the subtree rooted at u^*. That is, the subtree rooted at u^* contains some, but not all, of the input points (from $\hat{D}$). Hence, either the left-most descendant of u^*, denoted as $u^*_{\text{left-most}}$, or its right-most descendant, $u^*_{\text{right-most}}$, must be an interior point of $\hat{D}$. As $D_{\text{left}} \cup D_{\text{right}}$ contains t points which are bigger than any point in $\hat{D}$ as well as t points which are smaller than any point in $\hat{D}$, we get that one of $u^*_{\text{left-most}}, u^*_{\text{right-most}}$ is a "deep" interior point w.r.t. $D_{\text{left}} \cup D_{\text{right}}$ (with at least t points from each side of it). Choosing such a "deep" interior point (out of 2 choices) can be done using standard differentially private tools.

The privacy analysis of TreeLog is more challenging. The subtle point is that the path π selected in Step 4 is itself *highly non private*. Nevertheless, [16] showed that TreeLog is differentially private. Informally, the idea is as follows. Fix two neighboring datasets D and $D' = D \cup \{z\}$ and suppose that the same path π is selected during both the execution on D and the execution on D'. In that case, the embedded datasets generated by the two executions are neighboring, since except for the additional point z, all other points are embedded identically. If this is indeed the case, and assuming by induction that TreeLog (with one iteration less) is differentially private, then the recursive call in Step 6 satisfies privacy. The issue is that the path selected by TreeLog is *data dependent* and it could be very different during the two executions. Nevertheless, [16] showed that when this path is chosen correctly, then it still holds that neighboring datasets are mapped into neighboring embedded datasets,[6] which suffices for the privacy analysis. Importantly, for this argument to go through, it is essential that we do *not* embed the "last" t points that fall off the path π (the points that belong to the subtree rooted at u_π).

As the domain size reduces logarithmically with each recursive call, after $\log^*|X|$ steps the domain size is constant, and the recursion ends. (This base case, where the domain size is constant, can be handled using standard DP tools.) So there are $\log^*|X|$ steps throughout the execution. Hence, to obtain (ε,δ)-DP overall, [16] applied composition theorems for DP and used a privacy parameter of $\approx \frac{\varepsilon}{\sqrt{\log^*|X|}}$ in every step. This means that we trim $\approx \frac{\sqrt{\log^*|X|}}{\varepsilon}$ points with each iteration (in Steps 2 and 5) and we thus need at least $(\log^*|X|)^{1.5}$ input points in order to make it through to the end of the recursion.

Leveraging the RSC paradigm to obtain our upper bound. As an application of our RSC paradigm, we present a significantly improved analysis for algorithm TreeLog. Using the terminology of the RSC paradigm, we observe that each iteration of TreeLog cuts out three "slices" from the data: Two slices in Step 2 (the t smallest and t largest elements) and one slice in Step 5 (the last t points along the path which are not embedded). We show that the algorithm can be written in terms of the RSC paradigm, where every slice is "used only once". As a result, we get that it suffices to use a privacy parameter of (roughly) $\varepsilon/\log\frac{1}{\delta}$ for each step of the recursion, while still ending up with (ε,δ)-DP overall. So now we only trim $\approx \frac{1}{\varepsilon}\log\frac{1}{\delta}$ points in each step, and hence $\approx \log^*|X|$ points suffice in order to make it till the end of the recursion.

We stress that this is non-trivial to do without the RSC paradigm. In particular, one of the challenges here is that the embedding used by TreeLog is *not* order preserving, and the input points are "shuffled" again and again throughout the execution. As a result, there is no a priori order by which we can define the slices throughout the execution. In fact, to make this work, we need to introduce several technical modifications to the TreeLog algorithm, and to generalize the RSC paradigm to support it.

1.3.4 Another application of the RSC paradigm: axis-aligned rectangles.

We briefly describe another application of our RSC paradigm.

[6] More accurately, the distributions on embedded datasets during the two executions are "close" in the sense that there is a coupling between neighboring embedded datasets which have similar probability mass.

Consider the class C of all axis-aligned rectangles over a finite d-dimensional grid $X^d \subseteq \mathbb{R}^d$. A concept in this class could be thought of as the product of d intervals, one on each axis. Recently, Sadigurschi and Stemmer [21] presented a private learner for this class with sample complexity $\tilde{O}(d \cdot \mathrm{IP}(X))$, where $\mathrm{IP}(X)$ is the sample complexity needed for privately solving the interior point problem over X. As a warmup towards presenting their algorithm, [21] considered the following simple algorithm for this problem.

Input: Dataset $D \in (X^d \times \{0,1\})^n$ containing n labeled points from X^d.
Tool used: An algorithm $\mathcal{A}$ for the IP problem over X with sample complexity m.

(1) Let $S \subseteq D$ denote set of all positively labeled points in D (we assume that there are many such points, as otherwise the all-zero hypothesis is a good output).
(2) For every axis $i \in [d]$:
   (a) Project the points in S onto the ith axis.
   (b) Let A_i and B_i denote the smallest $(m + \text{Noise})$ and the largest $(m + \text{Noise})$ projected points, respectively, without their labels.
   (c) Let $a_i \leftarrow \mathcal{A}(A_i)$ and $b_i \leftarrow \mathcal{A}(B_i)$.
   (d) Delete from S all points (with their labels) corresponding to A_i and B_i.
(3) Return the axis-aligned rectangle defined by the intervals $[a_i, b_i]$ at the different axes.

The utility analysis of this algorithm is straightforward. As for the privacy analysis, observe that there is a total of $2d$ applications of the interior point algorithm $\mathcal{A}$ throughout the execution. Hence, using composition theorems, it suffices to run algorithm $\mathcal{A}$ with a privacy parameter of roughly $\varepsilon/\sqrt{d}$. However, this would mean that m (the sample complexity of $\mathcal{A}$) is at least $\sqrt{d}/\varepsilon$, and hence, each iteration deletes $\approx \sqrt{d}/\varepsilon$ points from the data and we need to begin with $|S| \gg d^{1.5}/\varepsilon$ input points. So this only results in an algorithm with sample complexity $\tilde{O}\left(d^{1.5} \cdot \mathrm{IP}(X)\right)$.

To overcome this, [21] designed a more complex algorithm with sample complexity linear in d. They left open the possibility that a better analysis of the simple algorithm outlined above could also result in near optimal sample complexity. Indeed, this follows immediately from our RSC paradigm: Every iteration reorders the data points along a different axis, takes out a "slice", and computes an interior point of this slice. Hence, by Theorem 1.5, it suffices to run $\mathcal{A}$ with a privacy parameter of $\approx \varepsilon/\log(\frac{1}{\delta})$, which avoids the unnecessary blowup of $\sqrt{d}$ in the sample complexity.

Corollary 1.7. *There is an approximate private algorithm for (properly) learning axis-aligned rectangles over a finite d-dimensional grid $X^d \subseteq \mathbb{R}^d$ with sample complexity $\widetilde{O}(d \cdot \log^* |X|)$.*

1.3.5 Our results for quasi-concave optimization. As we mentioned, Bun et al. [8] showed that privately learning thresholds is equivalent to privately solving the interior point problem. To obtain our results for quasi-concave optimization, we present a *stronger equivalence* in the context of quasi-concave optimization. More specifically, we show that private quasi-concave optimization is equivalent to solving the interior point problem *with "amplified" privacy guarantees.*[7] We leverage these amplified privacy guarantees to strengthen the lower bound of [8] for the interior point problem, thereby obtaining our lower bound of $\Omega(2^{\log^* T})$ for privately optimizing quasi-concave functions. We also leverage this equivalence in the positive direction, and design a suitable variant of our DP algorithm for the IP problem (with "amplified" privacy guarantees), thereby obtaining our upper bound of $\tilde{O}(2^{\log^* T})$ for privately optimizing quasi-concave functions.

1.4 Concluding Remarks

In this work, we present a nearly-optimal algorithm for privately solving the interior point problem. As a result, we can get near-optimal sample complexity for private learning thresholds and a couple of other applications through the reduction presented in [8]. This includes, e.g., finding an approximate median of the data, releasing threshold queries, and learning distribution with respect to Kolmogorov distance. We also present nearly matching upper and lower bounds for the task of private quasi-concave optimization. In particular, the lower bound suggests a natural limit of the current algorithmic techniques toward understanding private learning of discrete halfspaces.

Perhaps more importantly, we introduce new tools and ideas to obtain the aforementioned results. This includes the intriguing Reorder-Slice-Compute paradigm, as well as a simulation-based proof method to establish the privacy claim.

We find the simulation-based proof very useful in analyzing dynamic algorithms. At a high level, by fixing two adjacent data sets D, D', the simulation-based proof allows us to zoom in on the computation and find out the most "privacy-leaking" steps with respect to this *specific* pair of data sets D, D'. Then, we argue the privacy of the algorithm by designing a communication protocol between a "simulator" and a "data holder". We show that, knowing the true input is from the pair $\{D, D'\}$, the simulator only needs to know a *little bit* more about the input to simulate the algorithm faithfully. We quantify this "little bit" by bounding the information leaked to the simulator from the data holder. We hope this analysis method would inspire privacy proofs for more complex algorithms.

As our key application of this simulation-based proof method, we present and analyze the Reorder-Slice-Compute paradigm. The paradigm looks generic: it allows one to *select* a subset of data according to some desired ordering and perform computation on the selected items only. We can show a significant privacy saving if each item is only selected once. It is worth comparing the paradigm with the well-known privacy amplification by subsampling (PAS) technique (see, e.g. [3]), where one amplifies privacy by *sampling* a subset of data and performing computation on the sampled data only. Quantitatively, the privacy saving offered by RSC is strictly weaker: if the data set contains n items, one can use slices of size $\approx \frac{n}{\tau}$ to perform τ ε-DP computation with τ different groups of items, and the total privacy loss is $\widetilde{O}(\varepsilon)$. In contrast, by using PAS to sample $\frac{n}{\tau}$ items for each computation, one can perform a total of $O(\tau^2)$ computation and get overall privacy $\widetilde{O}(\varepsilon)$. However, RSC seems more versatile: it allows the data analyst to customize the data analysis procedure based on the *ordering* of individuals, which

[7] It is not that the privacy parameters are amplified, rather the resulting algorithm for the interior point problem satisfies a stronger (stringent) variant of differential privacy.

may be highly-sensitive information. It is also highly adaptive: the ordering, as well as the algorithms one wants to run on the slices, do not need to be fixed in advance. We hope to see more applications of the RSC paradigm.

2 REORDER-SLICE-COMPUTE

In this section, we introduce a simplified form of the Reorder-Slice-Compute paradigm. We refer to the full version for the most general form of our algorithm.

Notation. For two reals $a, b \geq 0$, we write $a \approx_\varepsilon b$ if $e^{-\varepsilon}b \leq a \leq e^{\varepsilon}b$. A *dataset* $D \in X^n$ can be viewed as a multiset of elements from X: The private algorithms we consider are applied to the respective multiset. We refer to an ordered multiset as a *list*. We consider two multisets or two lists D, D' *adjacent*, if and only if one of them (say, D') can be obtained by inserting a single element into the other. We say that a deterministic mapping $E : X^* \to X^*$ from multisets to lists is *adjacency preserving*, if for every pair of adjacent data sets $D, D' \cup \{x\}$, $E(D)$ and $E(D')$ are equal or adjacent lists. To simplify the presentation, we will sometimes treat lists as multisets and apply set operations on both multisets and lists ($D \cup \{x\}$ is the multiset D with the multiplicity of x incremented by 1).

2.1 The Reorder-Slice-Compute Paradigm

Algorithm 1 (`ReorderSliceCompute`) describes our paradigm. The algorithm performs τ adaptively-chosen computations over disjoint slices of an input dataset D. Each computation $i \in [\tau]$ is specified by a tuple $(m_i, \mathcal{A}_i, E_i)$: an (ε, δ)-DP algorithm $\mathcal{A}_i$, a specified approximate slice size (number of elements) $m_i \in \mathbb{N}$, and an adjacency preserving mapping $E_i : X^* \to X^*$ from data sets to lists.[8]

Given the tuple $(m_i, \mathcal{A}_i, E_i)$, we use E_i to process the input data set D_{i-1}, and select the first $\hat{m}_i := m_i + \mathtt{Geom}(1 - e^{-\varepsilon})$ elements of the list $E_i(D_{i-1})$ into the *slice* S_i. Then, we apply $\mathcal{A}_i$ to S_i, publish the result, and set D_i to be the (multiset of the) elements of the list $E_i(D_{i-1})$ with the prefix S_i removed.

The algorithm includes an optional *delayed-compute* phase, which follows the slicing phase. The slices $(S_i)_{i=1}^{\tau}$ are kept internally. The algorithm then adaptively receives a slice number i and an (ε, δ)-DP algorithm $\mathcal{A}'_i$, and publishes $\mathcal{A}'_i(S_i)$. Note that each slice is called at most once and the choice of the next slice and the selected algorithm may depend on results from prior slices.

We consider the total privacy cost of `ReorderSliceCompute`. Intuitively, we might hope for it to be close to (ε, δ)-DP, as each data element contributes to at most one slice. The slices, however, are selected from D in an adaptive and dependent manner. We can bound the total privacy cost using DP composition, but this results in a factor of τ or $\sqrt{\tau}$ (with advanced composition) increase in the privacy cost. A surprisingly powerful tool is our following theorem that avoids such dependence on τ:

Theorem 2.1 (Privacy of `ReorderSliceCompute`). *For every $\hat{\delta} > 0$, Algorithm 1 is $(O(\varepsilon \log(1/\hat{\delta})), \hat{\delta} + 2\tau\delta))$-DP.*

[8] One example is where E_i is a sorter that receives the data set $D \in X^n$ and a specified order $\prec$ on X and returns the sorted list of D by $\prec$ (as described in the intro). Our paradigm allows for more general data processing than sorting. This flexibility enables us to express a private algorithm for the interior point problem in this paradigm.

Algorithm 1: Reorder-Slice-Compute (RSC)

Input: Dataset $D = \{x_1, \ldots, x_n\} \in X^n$. Integer $\tau \geq 1$. Privacy parameters $0 < \varepsilon, \delta < 1$.

Function `SelectAndCompute`(D, m, $\mathcal{A}$, E)**:**
  $\hat{m} \leftarrow m + \mathtt{Geom}(1 - e^{-\varepsilon})$ `// Geom(p) denotes the geometric distribution with parameter p`
  $S \leftarrow$ the first $\hat{m}$ elements in $E(D)$
  $D \leftarrow E(D) \setminus S$
  $r \leftarrow \mathcal{A}(S)$
  return (D, S, r)

Program:
  `// Slice and Compute Phase:`
  $D_0 \leftarrow D$
  for $i = 1, \ldots, \tau$ **do**
    Receive $(m_i, \mathcal{A}_i, E_i)$ where $m_i \in \mathbb{N}$, an (ε, δ)-DP algorithm $\mathcal{A}_i$, and an adjacency-preserving mapping $E_i : X^* \to X^*$ from multisets to lists
    $(D_i, S_i, r_i) \leftarrow$ `SelectAndCompute`$(D_{i-1}, m_i, \mathcal{A}_i, E_i)$
    Publish r_i
  `// Delayed Compute Phase:`
  $I \leftarrow [\tau]$
  while *I is not empty* **do**
    Receive $(i, \mathcal{A})$, where $i \in I$ and $\mathcal{A}$ is an (ε, δ)-DP algorithm
    $I \leftarrow I \setminus \{i\}$
    Publish $\mathcal{A}(S_i)$

We can consider an extension of `ReorderSliceCompute` where we allow for up to k compute calls for each slice. The calls can be made at different points and adaptively, the only requirement is that they are made after the slice is finalized. Our analysis implies the following:

Corollary 2.2 (Privacy of `ReorderSliceCompute` with k computes per slice). *For every $k \geq 1$ and $\hat{\delta} > 0$, an extension of Algorithm 1 that allows for up to k computations on each slice is $(O(\varepsilon(k + \log(1/\hat{\delta}))), \hat{\delta} + 2k\tau\delta))$-DP.*

We can also consider performing k adaptive applications of `ReorderSliceCompute`. Interestingly, the factor of $\log(1/\hat{\delta})$ loss in privacy is incurred only once:

Theorem 2.3 (k adaptive applications of `ReorderSliceCompute`). *For every $k \geq 1$ and $\hat{\delta} > 0$, k adaptive applications of Algorithm 1 are $(O(\varepsilon(k + \log(1/\hat{\delta}))), \hat{\delta} + 2k\tau\delta))$-DP.*

In the following we prove Theorem 2.1 (privacy analysis of Algorithm 1). We perform the privacy analysis using the simulation-based technique outlined in Section 1.3.2. In Section 2.2 we introduce a tool that we call the synchronization mapping, that facilitates the synchronization performed by the data holder. In Section 2.3 we describe the simulator $\mathcal{S}$ and data holder H and establish that the simulation faithfully follows Algorithm 1. In Section 2.4 we show

that the data holder satisfies the privacy bounds of Theorem 2.1. The proof of Theorem 2.1 then follows using Lemma 1.6.

The proof of Theorem 2.3 is a simple extension and is included in Subsection 2.5.

2.2 The Synchronization Mapping

We specify a pair of randomized mappings $R^b_\varepsilon, b \in \{0,1\}$ that are indexed by a state bit b with the properties described in Lemma 2.4.

Notation. For a set S, $\Delta(S)$ denotes the set of all distributions supported on S. $\mathtt{Geom}(p)$ denotes the geometric distribution with stopping parameter p. Formally, $\Pr[\mathtt{Geom}(p) = k] = (1-p)^k \cdot p$ for every $k \geq 0$.

Lemma 2.4 (Synchronization lemma). *For every $\varepsilon \in (0,1)$, there are two randomized mappings $R^0_\varepsilon, R^1_\varepsilon : \mathbb{N} \to \Delta(\mathbb{N} \times \{0,1\})$ such that the following statements hold.*

1. *For every $m \in \mathbb{N}$, $\mathrm{supp}(R^0_\varepsilon(m)) \subseteq \{(m,0),(m,1)\}$, and $\mathrm{supp}(R^1_\varepsilon(m)) \subseteq \{(m,0),(m-1,1)\}$.*
2. *$R^0_\varepsilon(\mathtt{Geom}(1-e^{-\varepsilon}))$ and $R^1_\varepsilon(\mathtt{Geom}(1-e^{-\varepsilon}))$ are $(\varepsilon,0)$-indistinguishable.*
3. *For both $b \in \{0,1\}$, $\Pr_{(\alpha,\beta)\leftarrow R^b_\varepsilon(\mathtt{Geom}(1-e^{-\varepsilon}))}[\beta = 1] \geq \frac{1}{6}$.*

Proof. We construct a sequence $t_0, \ldots, t_\infty \in [0,1]^{\mathbb{N}}$ as follows:

$$t_i = \max\{0, e^{-i\varepsilon} + e^{-(i+1)\varepsilon} - 1\}, \ \forall i \geq 0.$$

It is easy to see that t_i is non-increasing and $t_i \leq e^{-(i+1)\varepsilon}$. Then, we set

$$R^0_\varepsilon(0) = \begin{cases} (0,0) & \text{w.p. } e^{-\varepsilon} \\ (0,1) & \text{w.p. } 1-e^{-\varepsilon} \end{cases}$$

and $R^1_\varepsilon(0) = (0,0)$ with probability one. For every $i \geq 1$, we explicitly set

$$R^0_\varepsilon(i) = \begin{cases} (i,0) & \text{w.p. } t_i \cdot e^{i\varepsilon} \\ (i,1) & \text{w.p. } 1 - t_i \cdot e^{i\varepsilon} \end{cases},$$

and

$$R^1_\varepsilon(i) = \begin{cases} (i,0) & \text{w.p. } t_i \cdot e^{(i+1)\varepsilon} \\ (i-1,1) & \text{w.p. } 1 - t_i \cdot e^{(i+1)\varepsilon} \end{cases}.$$

Note in particular that $\Pr[R^0_\varepsilon(0) = (0,1)] = 1 - e^{-\varepsilon} = 1 - t_0$.

Now we verify the validity of this construction. Obviously R^0_ε and R^1_ε satisfy Property 1. We verify Property 2 now. We first have

$$\Pr[R^0_\varepsilon(\mathtt{Geom}(1-e^{-\varepsilon})) = (0,0)] \approx_\varepsilon \Pr[R^1_\varepsilon(\mathtt{Geom}(1-e^{-\varepsilon})) = (0,0)].$$

For every $i \geq 1$, we have

$$\begin{aligned} &\Pr[R^0_\varepsilon(\mathtt{Geom}(1-e^{-\varepsilon})) = (i,0)] \\ &= (1-e^{-\varepsilon})e^{-i\varepsilon} \cdot t_i e^{i\varepsilon} \\ &\approx_\varepsilon (1-e^{-\varepsilon})e^{-i\varepsilon} \cdot t_i e^{(i+1)\varepsilon} \\ &= \Pr[R^1_\varepsilon(\mathtt{Geom}(1-e^{-\varepsilon})) = (i,0)]. \end{aligned}$$

Fix $i \geq 0$ and consider the output $(i,1)$. We have

$$\frac{\Pr[R^0_\varepsilon(\mathtt{Geom}(1-e^{-\varepsilon})) = (i,1)]}{\Pr[R^1_\varepsilon(\mathtt{Geom}(1-e^{-\varepsilon})) = (i,1)]} = \frac{e^{-i\varepsilon}(1-e^{-\varepsilon})(1-t_i e^{i\varepsilon})}{e^{-(i+1)\varepsilon}(1-e^{-\varepsilon})(1-t_{i+1}e^{(i+2)\varepsilon})} = e^{\varepsilon}\frac{1-t_i e^{i\varepsilon}}{1-t_{i+1}e^{(i+2)\varepsilon}}.$$

Let us consider $1 - t_i e^{i\varepsilon}$. If $t_i = 0$, then $1 - t_i e^{i\varepsilon} = 1$. Otherwise, $1 - t_i e^{i\varepsilon} = e^{i\varepsilon} - e^{-\varepsilon}$. Combining both cases, we conclude that $1 - t_i e^{i\varepsilon} = \min\{1, e^{i\varepsilon} - e^{-\varepsilon}\}$. Similarly, we have $1 - t_{i+1}e^{(i+2)\varepsilon} = \min\{1, e^{(i+2)\varepsilon} - e^{\varepsilon}\}$. Therefore, it is clear that

$$e^{\varepsilon}\frac{1-t_i e^{i\varepsilon}}{1-t_{i+1}e^{(i+2)\varepsilon}} = \frac{e^{\varepsilon}\cdot\min\{1, e^{i\varepsilon}-e^{-\varepsilon}\}}{\min\{1, e^{(i+2)\varepsilon}-e^{\varepsilon}\}} \in [e^{-\varepsilon}, e^{\varepsilon}].$$

We have fully verified Property 2. It remains to verify Property 3. Let $\gamma \geq 0$ be the minimum integer such that $t_\gamma = 0$. Note that for every input $m \geq \gamma$, with probability one, we have $R^b_\varepsilon(m) = (m-b, 1)$ for both $b \in \{0,1\}$. Therefore, it is suffices to lower bound $\Pr[\mathtt{Geom}(1-e^{\varepsilon}) \geq \gamma] = e^{-\varepsilon\gamma}$. Since γ is the minimum integer such that $t_\gamma = 0$, we have $t_{\gamma-1} = e^{-(\gamma-1)\varepsilon} + e^{-\gamma\varepsilon} - 1 > 0$, implying that $e^{-\varepsilon\gamma} > \frac{1}{1+e^{\varepsilon}} \geq \frac{1}{6}$ as $\varepsilon \leq 1$. □

2.3 The Simulator and Data Holder

We describe the simulator and the data holder and establish that the interaction is a faithful simulation of Algorithm 1 and hence satisfies the first condition of Lemma 1.6. To simplify presentation, we present the simulation for Algorithm 1 without the delayed compute phase, and then explain how the simulation and analysis can be extended to include delayed compute.

The simulator is described in Algorithm 2 and the data holder query response algorithm is described in Algorithm 3. The simulator receives as input two adjacent datasets $D, D' = D \cup \{x\}$. It then runs a simulation of Algorithm 1 and maintains internal state for both cases of the input dataset being D (state $b = 0$) and the input dataset being $D' = D \cup \{x\}$ (state $b = 1$). The simulation is guaranteed to remain perfect only for the correct case b. The simulator initializes $D_0 \leftarrow D$ and updates the active elements $D_i \subset E_{i-1}(D_{i-1})$ and the applicable diff element x. The simulator maintains a status bit that is initially 0 (two cases are not synchronized) and at some point the status becomes and then remains 1 (two cases are synchronized). When the status bit is 0, internal states are maintained for both cases: The active elements for case $b = 0$ are D_i and the active elements for case $b = 1$ are D_i and one additional element x (initially $x \in D' \setminus D$ but can get replaced). When the status bit is 1, the internal state is only that of the true case (the active elements of the true case are D_i), there is no diff element maintained, and the simulation proceeds like Algorithm 1.

Until synchronization, the simulator slices the data set by emulating `SelectAndCompute`. When the slices are such that they are identical for both cases, that is, the $\hat{m}_i$ prefix of $E_{i-1}(D_{i-1})$ is equal to the $\hat{m}_i$ prefix of $E_{i-1}(D_{i-1} \cup \{x\})$, the computation does not depend on the state b and the simulator performs it and reports the result r without accessing the data holder. The set of active elements with the slice removed continue to differ by the one element x' that is the difference of the multisets $E_{i-1}(D_{i-1} \cup \{x\})$ and $E_{i-1}(D_{i-1})$. If the slices for the two cases are different, then let p be the first position of the list $E_{i-1}(D_{i-1} \cup \{x\})$ that does not have the same element as the same position of $E_{i-1}(D_{i-1})$. Note that we must have $\hat{m}_i \geq \max\{p, m_i\}$. The slice for the case $b = 1$ includes an element x' and the slice for $b = 0$ includes a different element y at position $\hat{m}_i$ of $E_{i-1}(D_{i-1})$. The data holder (Algorithm 3) therefore must be called to obtain a correct result r. The data holder redraws the slice size $\hat{m}_i$ conditioned on it being at least $\max\{p, m_i\}$. This provides an

opportunity for synchronization without changing the distribution (from the memorylessness property of the geometric distribution, the difference under such conditioning is an independent draw from the geometric distribution). The data holder attempts to synchronize (that involves applying the randomized mapping that also depends on b). It reports back a triple: The computation result r, status indicating whether synchronization was successful, and a slice size $\hat{q}$ to remove from the prefix of $E_{i-1}(D_{i-1})$ to obtain D_i. If there was no synchronization, the simulator computes the new diff element.

Note that if there is no synchronization, the reported size results in perfect removal by the simulator of the elements that participated in the slice for both cases of $b = 0$ or $b = 1$. The element y that participated in the slice for case $b = 0$ but not in $b = 1$ replaces x. If the synchronization was successful, then the simulator no longer maintains an additional element and the set D_i is exactly $E_{i-1}(D_{i-1})$ with the elements that participated in the slice for the true case removed.

Lemma 2.5. *For $b = 0$ (resp. $b = 1$), Algorithm 2 simulates the execution of Algorithm 1 on the data set D (resp. $D \cup \{x\}$) perfectly.*

Proof. We prove that at the start of each round $i \in [\tau]$, Algorithm 2 maintains the current data set accurately by the triple $(D_{i-1}, x, \text{status})$, in the following sense.

- If $b = 0$, then the current data set is D_{i-1}.
- Otherwise (i.e., $b = 1$), if status $= 0$, the current data set is $D_{i-1} \cup \{x\}$. If status $= 1$, the current data set is D_{i-1}.

We prove the claim by induction on $i \in [\tau]$. This is clearly true for $i = 1$. Now assume the statement holds for $i - 1 \geq 1$. We prove for the case of i. There are three cases:

Case 1. status $= 1$. In this case, the current data set is the same for the two cases (that is, is independent of the private bit b). Therefore, the call to `SelectAndCompute` is a correct simulation.

Case 2. status $= 0$. In this case, let $p \geq 1$ be the rank of x in $D_{i-1} \cup \{x\}$. To simulate `SelectAndCompute`, we need to sample $\hat{m}_i \leftarrow m_i + \mathsf{Geom}(1 - e^{-\varepsilon})$, and use the first $\hat{m}_i$ elements in the applicable list $E_i(D_{i-1})$ or $E_i(D_{i-1} \cup \{x\})$ to do the computation. Algorithm 2 first samples $\hat{m}_i$ and tests if $\hat{m}_i < p$. The test yields two cases:

- $\hat{m} < p$. In this case, for both $b \in \{0, 1\}$, the prefix is the same and Algorithm 1 would select the same subset of elements. Therefore, the simulator can perfectly simulate this case without querying the private bit b. It is easy to see that the update from D_{i-1} to D_i is valid.
- $\hat{m} \geq p$. In this case, the private bit $b \in \{0, 1\}$ does make a difference. Hence, the simulator asks the data holder H to do this round of `SelectAndCompute`, *conditioned on* $\hat{m}_i \geq \max(p, m_i)$ (i.e., at least $\max(p, m_i)$ elements are selected).

  We need a well-known fact about Geometric distribution (the memoryless property): suppose there is a random variable $x = a + \mathsf{Geom}(1 - e^{-\varepsilon})$. Then, conditioned on $x \geq y$ for some $y \geq a$, x is distributed as $y + \mathsf{Geom}(1 - e^{-\varepsilon})$. Therefore, conditioned on $\hat{m}_i \geq \max(p, m_i)$, Lines 2-3 in Algorithm 3 sample the number of participating elements perfectly.

Algorithm 2: The Simulator

Input: A pair of adjacent datasets $D, D' = D \cup \{x\}$. Integer $\tau \geq 1$. Privacy parameters $\varepsilon \in (0, 1), \delta > 0$.

1 **Program:**
2 $D_0 \leftarrow D$
3 $x \leftarrow x$
4 status $\leftarrow 0$ // status = 1 indicates two data sets have been "synchronized"
5 **for** $i = 1, \ldots, \tau$ **do**
6 Receive $m_i \in \mathbb{N}$, an (ε, δ)-DP algorithm $\mathcal{A}_i$, and an adjacency preserving map $E_i : X^* \to X^*$
7 **if** (status $= 0$) $\wedge$ $(E_i(D_{i-1}) = E_i(D_{i-1} \cup \{x\}))$ **then**
// Map E eliminated the diff element
8 status $\leftarrow 1$ // Synchronized
9 **if** status $= 1$ **then**
10 $(D_i, r) \leftarrow \mathtt{SelectAndCompute}(D_{i-1}, m_i, \mathcal{A}_i, E_i)$
11 **else**
12 $\hat{m}_i \leftarrow m_i + \mathsf{Geom}(1 - e^{-\varepsilon})$
13 $x' \leftarrow E_i(D_{i-1} \cup \{x\}) \setminus E_i(D_{i-1})$ // diff element of mapped datasets
14 $p \leftarrow$ the rank of x' in $E_{i-1}(D_{i-1} \cup \{x\})$
15 **if** $\hat{m}_i < p$ **then** // This round does not involve diff element
16 $S_i \leftarrow$ the first $\hat{m}_i$ elements in $E_i(D_{i-1})$
// Slice S_i is the same if selected from $E_i(D_{i-1} \cup \{x\})$
17 $D_i \leftarrow E_i(D_{i-1}) \setminus S_i$
18 $x \leftarrow x'$
19 $r \leftarrow \mathcal{A}_i(S_i)$
20 **else** // This round involves diff element
21 $q \leftarrow \max(p, m_i)$
22 $(\hat{q}, \text{new\_status}, r) \leftarrow$ $\mathtt{Query}(D_{i-1}, x, q, \mathcal{A}_i, E_i))$ // Query the data holder Algorithm 3 and receive a triple $(\hat{q}, \text{new\_status}, r) \in \mathbb{N} \times \{0, 1\} \times \mathcal{Y}$
23 $S \leftarrow$ the first $\hat{q}$ elements in $E_i(D_{i-1})$
24 **if** new_status $= 0$ **then**
// Synchronization failed
25 $x \leftarrow$ the $\hat{q}$-th largest element in $E_i(D_{i-1})$
26 $D_i \leftarrow E_i(D_{i-1}) \setminus S$
27 **else** // Successful synchronization
28 $D_i \leftarrow E_i(D_{i-1}) \setminus S$
29 status $\leftarrow 1$
30 **Publish** r
31 **return** $(D_\tau, \text{status}, x)$

Having sampled $\hat{m}$, Algorithm 3 selects the prefix of $\hat{m}$ elements from either $E_i(D_{i-1})$ or $E_i(D_{i-1} \cup \{x\})$ (depending

Algorithm 3: The Query Algorithm to the Data Holder

Input: A private bit $b \in \{0,1\}$ indicating whether the input data set is D ($b = 0$) or $D' = D \cup \{x\}$ ($b = 1$). Privacy parameters $\varepsilon \in (0,1), \delta > 0$.

Function `Query`$(D, x, q, \mathcal{A}, E)$**:**
- $\Delta \leftarrow \mathsf{Geom}(1 - e^{-\varepsilon})$
- $\hat{m} \leftarrow q + \Delta$
- **if** $b = 0$ **then**
  - $S \leftarrow$ the first $\hat{m}$ elements in $E(D)$
- **else**
  - $S \leftarrow$ the first $\hat{m}$ elements in $E(D \cup \{x\})$
- $r \leftarrow \mathcal{A}(S)$
- $(\alpha, \beta) \leftarrow R^b_\varepsilon(\Delta)$ `// Try to synchronize`
- $\hat{q} \leftarrow q + \alpha$ `// $\hat{q}$ is the reported number of participating elements.`
- **return** $(\hat{q}, \beta, r)$

on $b \in \{0,1\}$), and runs $\mathcal{A}$ on the selected elements. This part simulates Algorithm 1 faithfully.
Finally, Algorithm 3 runs the synchronization mechanism and returns the triple $(\hat{q}, \beta, r)$. Given this triple, we verify the validity of the update from D_{i-1} to D_i. This is straightforward.

If $b = 0$, then it is always the case that $\hat{m} = \hat{q}$, and note that Algorithm 2 always removes the first $\hat{m}$ elements from $E_i(D_{i-1})$, no matter what new_status is.

If $b = 1$, then we have $\hat{m} = \hat{q}$+new_status. Depending on the value of new_status, there are two cases: if new_status = 1, then we update the data set from $D_{i-1} \cup \{x\}$ to D_i, where D_i is obtained by removing the first $\hat{q}$ elements from $E_i(D_{i-1})$. Overall this process removes $\hat{q}$+1 = $\hat{m}$ elements. If new_status = 0, then we update the data set from $D_{i-1} \cup \{x\}$ to $D_i \cup \{y\}$ where y is the $\hat{q}$-th element in $E_i(D_{i-1})$. Over all this process removes the first $\hat{q} = \hat{m}$ elements from $E_i(D_{i-1} \cup \{x\})$.

In summary, assuming the first $(i-1)$-rounds simulate Algorithm 1 perfectly, and the triple $(D_{i-1}, x, \text{status})$ is accurately maintained (in the aforementioned sense), we have shown that the i-th round of simulation is also perfect, and the triple is updated accurately. By induction, this shows that the simulator simulates all of the τ rounds perfectly, as desired. □

Simulation with delayed compute. We outline the modifications needed in the simulation when including the delayed compute phase. It is straightforward to verify that it remains a faithful simulation of `ReorderSliceCompute` with delayed compute.

The modified simulator performs two phases. The first is the slicing phase, that is the same as Algorithm 2 except that the modified simulator stores the slices S_i for steps i that did not require calls to the holder. It also keeps track of the set of steps J for which it called the data holder. Additionally, the calls to the data holder also specify the step number i. In the second phase (delayed-compute) the simulator handles $i \notin J$ by applying the provided algorithm to the stored S_i and publishes the result. When $i \in J$, the simulator calls the data holder with the specified step number i and the provided algorithm.

The modified data holder (Algorithm 3) takes two types of calls, depending on which phase the simulator is in. The first phase calls correspond to the slicing. These calls are as described in Algorithm 3, except that: (1) we allow calls without computations (and results) and (2) the call includes the step number i and the data holder stores internally the applicable slice S_i.

In the delayed-compute phase calls, the input to the data holder is $(i, \mathcal{A})$, where $i \notin J$ is a step number for which it has S_i stored and $\mathcal{A}$ an (ε, δ)-DP. The holder publishes the output $\mathcal{A}(S_i)$.

2.4 Simulation Privacy Analysis

The following two lemmas imply that the data holder satisfies the privacy bound stated in Theorem 2.1. They also imply the bound stated in Corollary 2.2 for the extension where we allow k (ε, δ)-DP computations per slice.

Lemma 2.6. *Each call by Algorithm 2 to Algorithm 3 in the first phase is $(3\varepsilon, 2e^{2\varepsilon}\delta)$-DP and each call in the second phase is $(2\varepsilon, 2e^{2\varepsilon}\delta)$-DP with respect to the private input $b \in \{0,1\}$.*

Proof. On a query, the output of Algorithm 3 is a triple $(\hat{q}, \beta, r)$. Note that the pair $(\hat{q}, \beta)$ does not depend on the result r. It will be convenient for us to analyse the privacy cost by treating Algorithm 3 as first returning $(\hat{q}, \beta)$ and storing S, and then at some later point, taking $\mathcal{A}$ as input and computing and returning $r \leftarrow \mathcal{A}(S)$.

Note that $(\hat{q}, \beta) = (\alpha + q, \beta)$ where $(\alpha, \beta) \sim R^b_\varepsilon(\mathsf{Geom}(1 - e^{-\varepsilon}))$. Therefore, by Property 2 in Lemma 2.4, the pair $(\hat{q}, \beta)$ is $(\varepsilon, 0)$-DP with respect to the private bit b.

The algorithm chooses the set S depending on the private bit b. In the following, we use S^b to denote the respective choice. Next, having learned $(\hat{q}, \beta)$, from the viewpoint of the simulator, she can deduce the following.

- If $\beta = 0$, the set S^b used in this query would be the first $\hat{q}$ elements in $E(D)$, or the first $\hat{q} - 1$ elements in $E(D)$ plus the extra element $\{x\}$, depending on whether b equals 0 or 1. Since S^0 and S^1 differ by at most two elements, the result $r \sim \mathcal{A}(S^b)$ is $(2\varepsilon, 2e^{2\varepsilon}\delta)$-DP w.r.t. b.
- If $\beta = 1$, the set S^b would be the first $\hat{q}$ elements in $E(D)$, or the first $\hat{q}$ elements plus the extra point $\{x\}$, depending on the private bit b. Since S^0 and S^1 differ by at most one element, the result $r \sim \mathcal{A}(S^b)$ is (ε, δ)-DP w.r.t. b.

By the basic composition theorem, $(\hat{q}, \beta, r)$ is $(3\varepsilon, 2e^{2\varepsilon}\delta)$-DP w.r.t. the private bit b.

Note that this holds also for the delayed-computes that are performed in the second phase and are applied to S^b that are $(2\varepsilon, 2e^{2\varepsilon}\delta)$-DP. □

Lemma 2.7. *For every $\hat{\delta} > 0$, with probability $1 - \hat{\delta}$, Algorithm 2 makes at most $O(\log(1/\hat{\delta}))$ queries to Algorithm 3.*

Proof. It suffices to consider the number of calls made during the first phase. Each time the simulator calls Algorithm 3, with probability at least $\frac{1}{6}$, Algorithm 3 responds a triple with $\beta = 1$. Further observe that once the simulator gets a triple with $\beta = 1$, she will never send query to Algorithm 3 again. Therefore, the

probability that the simulator sends more than $w \in \mathbb{N}$ queries is at most $(5/6)^w$, as desired. □

Combining Lemma 2.6 and 2.7 proves Theorem 2.1.

2.5 Analysis for k Adaptive Applications of Reorder-Slice-Compute

We outline the proof of Theorem 2.3. We follow the analysis as in the proof of Theorem 2.1. We apply the simulator for the k executions of `ReorderSliceCompute`. The privacy cost depends on the total number of calls to Algorithm 3 which we bound as follows:

Lemma 2.8. *Let the random variable Z_k be the number of calls to Algorithm 3 in k executions of Algorithm 2. There is a constant c such that for all $k \geq 1$, $\hat{\delta} > 0$, $\Pr[Z_k \geq c \max\{k, \ln(1/\hat{\delta})\}] \leq \hat{\delta}$.*

Proof. The total number of calls to Algorithm 3 is dominated by the sum of k independent $\mathtt{Geom}(p)$ random variables with parameter $p \geq 1/6$. Using tail bounds on the sum of Geometric random variables [14], we obtain that for all $\lambda \geq 1$,

$$\Pr[Z_k \geq \lambda k/p] \leq \exp(-k(\lambda - 1 - \ln \lambda)) .$$

Substituting $n = \lambda k/p$ we obtain for $n \geq 10k/p = \Omega(k)$: $\Pr[Z_k \geq n] \leq \exp(-n/2)$. Therefore for some constant c, for all $\hat{\delta} > 0$, $\Pr[Z_k \geq c \max\{k, \ln(1/\hat{\delta})\}] \leq \hat{\delta}$. □

ACKNOWLEDGMENTS

Edith Cohen is partially supported by Israel Science Foundation (grant no. 1595/19). Uri Stemmer is partially supported by the Israel Science Foundation (grant 1871/19) and by Len Blavatnik and the Blavatnik Family foundation.

REFERENCES

[1] Noga Alon, Mark Bun, Roi Livni, Maryanthe Malliaris, and Shay Moran. 2022. Private and Online Learnability Are Equivalent. *J. ACM* 69, 4 (2022), 28:1–28:34.
[2] Noga Alon, Roi Livni, Maryanthe Malliaris, and Shay Moran. 2019. Private PAC learning implies finite Littlestone dimension. In *STOC*.
[3] Borja Balle, Gilles Barthe, and Marco Gaboardi. 2018. Privacy Amplification by Subsampling: Tight Analyses via Couplings and Divergences. In *NeurIPS*. 6280–6290.
[4] Amos Beimel, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. 2010. Bounds on the Sample Complexity for Private Learning and Private Data Release. In *TCC*.
[5] Amos Beimel, Shay Moran, Kobbi Nissim, and Uri Stemmer. 2019. Private Center Points and Learning of Halfspaces. In *COLT*.
[6] Amos Beimel, Kobbi Nissim, and Uri Stemmer. 2013. Private Learning and Sanitization: Pure vs. Approximate Differential Privacy. In *APPROX-RANDOM*.
[7] Mark Bun, Cynthia Dwork, Guy N. Rothblum, and Thomas Steinke. 2018. Composable and versatile privacy via truncated CDP. In *STOC*.
[8] Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil P. Vadhan. 2015. Differentially Private Release and Learning of Threshold Functions. In *FOCS*.
[9] Kamalika Chaudhuri and Daniel J. Hsu. 2011. Sample Complexity Bounds for Differentially Private Learning. In *COLT*.
[10] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*.
[11] Dan Feldman, Chongyuan Xiang, Ruihao Zhu, and Daniela Rus. 2017. Coresets for differentially private k-means clustering and applications to privacy in mobile sensor networks. In *IPSN*.
[12] Vitaly Feldman and David Xiao. 2015. Sample Complexity Bounds on Differentially Private Learning via Communication Complexity. *SIAM J. Comput.* 44, 6 (2015), 1740–1764. https://doi.org/10.1137/140991844
[13] Yue Gao and Or Sheffet. 2021. Differentially Private Approximations of a Convex Hull in Low Dimensions. In *ITC*.
[14] Svante Janson. 2017. Tail bounds for sums of geometric and exponential variables. https://doi.org/10.48550/ARXIV.1709.08157
[15] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. 2015. The Composition Theorem for Differential Privacy. In *ICML*.
[16] Haim Kaplan, Katrina Ligett, Yishay Mansour, Moni Naor, and Uri Stemmer. 2020. Privately Learning Thresholds: Closing the Exponential Gap. In *COLT*.
[17] Haim Kaplan, Yishay Mansour, Uri Stemmer, and Eliad Tsfadia. 2020. Private Learning of Halfspaces: Simplifying the Construction and Reducing the Sample Complexity. In *NeurIPS*.
[18] Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. 2008. What Can We Learn Privately?. In *FOCS*.
[19] Jack Murtagh and Salil P. Vadhan. 2016. The Complexity of Computing the Optimal Composition of Differential Privacy. In *TCC (A1)*.
[20] Kobbi Nissim, Uri Stemmer, and Salil P. Vadhan. 2016. Locating a Small Cluster Privately. In *PODS*.
[21] Menachem Sadigurschi and Uri Stemmer. 2021. On the Sample Complexity of Privately Learning Axis-Aligned Rectangles. In *NeurIPS*.
[22] Leslie G Valiant. 1984. A theory of the learnable. *Commun. ACM* 27, 11 (1984), 1134–1142.
[23] V. N. Vapnik and A. Ya. Chervonenkis. 1971. On the Uniform Convergence of Relative Frequencies of Events to their Probabilities. *Theory of Probability and its Applications* 16, 2 (1971), 264–280.

Received 2022-11-07; accepted 2023-02-06