Fast Private Data Release Algorithms for Sparse Queries

Avrim Blum[⋆] and Aaron Roth[⋆⋆]

[1] Carnegie Mellon University
avrim@cs.cmu.edu
[2] University of Pennsylvania
aaroth@cis.upenn.edu

Abstract. We revisit the problem of accurately answering large classes of statistical queries while preserving differential privacy. Previous approaches to this problem have either been very general but have not had run-time polynomial in the size of the database, have applied only to very limited classes of queries, or have relaxed the notion of worst-case error guarantees. In this paper we consider the large class of *sparse* queries, which take non-zero values on only polynomially many universe elements. We give efficient query release algorithms for this class, in both the interactive and the non-interactive setting. Our algorithms also achieve better accuracy bounds than previous general techniques do when applied to sparse queries: our bounds are independent of the universe size. In fact, even the runtime of our interactive mechanism is independent of the universe size, and so can be implemented in the "infinite universe" model in which no finite universe need be specified by the data curator.

1 Introduction

A database $\mathcal{D}$ represents a finite collection of individual records from some *data universe* $\mathcal{X}$, which represents the set of all *possible* records. We typically think of $\mathcal{X}$ as being extremely large: exponentially large in the size of the database, or in some cases, possibly even infinite. A fundamental task in private data analysis is to accurately answer statistical queries about a database $\mathcal{D}$, while provably preserving the privacy of the individuals whose records are contained in $\mathcal{D}$. The privacy solution concept we use in this paper is *differential privacy*, which has become standard, and which we define in section 2.

Accurately answering statistical (i.e. linear or counting) queries is the most well studied problem in differential privacy, and the results to date come in two types. There are a large number of extremely general and powerful techniques (see for example [BLR08, DNR$^+$09, DRV10, RR10, HT10, HR10]) that can accurately answer arbitrary families of statistical queries which can be exponentially large in the size of the database. Unfortunately, these techniques all

[⋆] Department of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213.
[⋆⋆] Department of Computer and Information Science, University of Pennsylvania, Philadelphia PA 19104. This research was partially supported by an NSF CAREER Grant and NSF grant CNS-1065060.

have running time that is at least linear in the size of the data universe $|\mathcal{X}|$ (i.e. possibly *exponential* in the size of the database), and so are in many cases impractical. There are also several techniques that do run in polynomial time, but that are limited: either they can answer queries from a very general and structurally rich class (i.e. all low-sensitivity queries), but can only answer a linear number of such queries (i.e. [DMNS06]), or they can answer a very large number of queries, but only from a structurally very simple class (i.e. intervals on the unit line[1] [BLR08]), or as in several recent results (for conjunction and parity queries respectively) [GHRU11, HRS12] they run in polynomial time, but offer only average case guarantees for randomly chosen queries. One of the main open questions in data privacy is to develop general data release techniques comparable in power to the known exponential time techniques that run in polynomial time. There is evidence, however, that this is not possible for arbitrary linear queries [DNR+09, UV11, GHRU11].

In this paper, we consider a restricted but structurally rich class of linear queries which we call *sparse* queries. We say that a query is m-sparse if it takes non-zero values on only m universe elements, and that a class of queries is m-sparse if each query it contains is m' sparse for some $m' \leq m$. We will typically think of m as being some polynomial in the database size n. Note that although each individual query is restricted to have support on only a polynomially sized subset of the data universe, different queries in the same class can have different supports, and so a class of sparse queries can still have support over the entire data universe. This is what prevents previous algorithms from being made efficient, simply by running them on a restricted universe: it may not be possible to consider a restricted universe for a set of sparse queries, because every universe element may take positive value on some query in a sparse class! Note that the class of m-sparse queries is both very large (of size roughly $|\mathcal{X}|^m$), and very structurally complex (the class of m-sparse queries have VC-dimension m). Sparse queries represent questions about individuals whose answer is rarely "yes" when asked about an individual who is drawn uniformly at random from the data population. Nevertheless, such questions can be useful to a data analyst who has some knowledge about which segment of the population a database might be drawn from. For example, a database resulting from a medical study might contain individuals who have some rare disease, but the data analyst does not know *which* disease – although there may be many such queries, each one is sparse. Alternately, a data analyst might have knowledge about the participants of several previous studies, and might want to know how much overlap there is between the participants of each previous study and of the current study. In general, sparse queries will only be useful to a data analyst who has some knowledge about the database, beyond that it is merely a subset of an exponentially sized data universe. Our results can therefore be viewed as a way of privately releasing information about a database that is useful to specialists – but is privacy preserving no matter who makes use of it. In general, this work can be thought

[1] The algorithm of [BLR08] can be generalized to answer axis-aligned rectangle queries in constant dimension, but this is still a class that has only constant VC-dimension.

of as part of an agenda to find ways to make use of the *domain knowledge* of the data analyst, to make private analysis of large-scale data-sets feasible.

1.1 Results

We give two algorithms for releasing accurate answers to m-sparse queries while preserving differential privacy: one in the interactive setting, in which the data curator acts as an intermediary and must answer an adaptively chosen stream of queries as they arrive, and one in the non-interactive setting, in which the data curator must in one shot output a data-structure which encodes the answers to every query of interest. In the interactive setting, we require that the running time needed to answer each query is bounded by a polynomial in n, the database size (so to answer any sequence of k queries takes time $k \cdot \text{poly}(n)$). In the non-interactive setting, the entire computation must be performed in time polynomial in n, and the time required to evaluate any query on the output data structure must also be polynomial. Therefore, from the point of view of running time, the non-interactive setting is strictly more difficult than the interactive setting.

In the interactive setting, we give the following utility bound:

Theorem 1 (Informal, some parameters hidden). *There exists an (ϵ, δ)-differentially private query release mechanism in the interactive setting, with running time per query $\tilde{O}(m/\alpha^2)$ that is α-accurate with respect to any set of k adaptively chosen m-sparse queries with:*

$$\alpha = O\left(\frac{(\log m)^{1/4}\left(\log\frac{1}{\delta}\log k\right)^{1/2}}{(\epsilon n)^{1/2}}\right)$$

In the non-interactive setting, we give the bound:

Theorem 2 (Informal, some parameters hidden). *There exists an (ϵ, δ)-differentially private query release mechanism in the non-interactive setting, with running time polynomial in the database size n, m, and $\log|\mathcal{X}|$, that is α-accurate with respect to any class of k m-sparse linear queries, with:*

$$\alpha = \tilde{O}\left(\log k \frac{\sqrt{m \log\left(\frac{1}{\delta}\right)}}{\epsilon n}\right)$$

Several aspects of these theorems are notable. First, the accuracy bounds do not have any dependence on the size of the data universe $|\mathcal{X}|$, and instead depend only on the sparsity parameter m. Therefore, in addition to efficiency improvements, these results give accuracy improvements for sparse queries, when compared to the general purpose (inefficient) mechanisms for linear queries, which typically have accuracy which depends on $\log|\mathcal{X}|$. Since we typically view $|\mathcal{X}|$ as exponentially large in the database size, whereas m is only polynomially large in the database size for these algorithms to be efficient, this can be a large improvement in accuracy.

Second, the interactive mechanism does not even have a dependence on $|\mathcal{X}|$ in its running time! In fact, it works even in an *infinite* universe (e.g. data entries with string valued attributes without pre-specified upper bound on length)[2]. In this setting, queries may still be concisely specified as a list of polynomially many individuals from the possibly infinite universe that satisfy the query. Moreover, because the accuracy of this mechanism depends only very mildly on m, and the running time is linear in m, it can be used to answer m-sparse queries for arbitrarily large polynomial values of m, where the mechanism is constrained only by the available computational resources.

The non-interactive mechanism in contrast has a worse dependence on m. This bound essentially matches the error that would result from releasing the perturbed *histogram* of the database, but does so in a way that requires computation and output representation only polynomial in n (rather than linear in $|X|$, as releasing a histogram would require). Because accuracy bounds > 1 are trivial, this mechanism only guarantees non-trivial accuracy for m-sparse queries with $m << n^2/\log k$ (This is still of course a very large class of queries: there are roughly $|\mathcal{X}|^{n^2/\log k}$ such queries, i.e., super-exponentially many in n). Nevertheless, there are distinct advantages to having a non-interactive mechanism that only needs to be run once. This is among the first *polynomial time* non-interactive mechanisms for answering an exponentially large, unstructured class of queries while preserving differential privacy.

We note that our results give as a corollary, more efficient algorithms for answering conjunctions with many literals. This complements the beautiful recent work of Hardt, Rothblum, and Servedio [HRS12], who give more efficient algorithms for answering conjunctions with few literals, based on reductions to threshold learning problems.

1.2 Techniques

Our interactive mechanism is a modification of the very general multiplicative weights mechanism of Hardt and Rothblum [HR10]. We give the interactive mechanism via the framework of [GRU12] which efficiently maps objects called *iterative database constructions* (defined in section 3) into private query release mechanisms in the interactive setting. IDC algorithms are very similar to online learning algorithms in the mistake bound model, and we use this analogy to implement a version of the multiplicative weights IDC of Hardt and Rothblum [HR10] analogously to how the Winnow algorithm is implemented in the *infinite*

[2] The algorithm must be able to read a *name* for each universe element it deals with, and so it can of course not deal with elements that have no finite description length. But for a (countably) infinite universe, the running time would depend on the length of the largest string used to denote a universe element encountered during the running of the algorithm, and not in any a-priori way on the (unboundedly large) size of the universe.

attribute model of learning, defined by Blum [Blu90]. The algorithm roughly works as follows: the multiplicative weights algorithm normally maintains a distribution over $|\mathcal{X}|$ elements, one for each element in the data universe. It can be easily implemented in such a way so that when it is updated after a query Q arrives, only those weights corresponding to elements in the support of the query Q are updated: for an m-sparse query, this means it only need update m positions. It also comes with a guarantee that it never needs to perform more than $\log|\mathcal{X}|/\alpha^2$ updates before achieving error α, and so at most $m\log|\mathcal{X}|/\alpha^2$ elements ever need to be updated. The key insight is to pick a smaller universe, $\widehat{\mathcal{X}}$, such that $\widehat{\mathcal{X}} \geq m\log\widehat{\mathcal{X}}/\alpha^2$, but *not to commit to the identity of the elements in this universe* before running the algorithm, letting all elements be initially unassigned. The algorithm then maintains a hash table mapping elements of $\mathcal{X}$ to elements of $\widehat{\mathcal{X}}$. Elements in $\mathcal{X}$ are assigned temporary mappings to elements in $\widehat{\mathcal{X}}$ as queries come in, but are only assigned permanent mappings when an update is performed. Because only $\log\widehat{\mathcal{X}}/\alpha^2$ updates are ever performed, and $\widehat{\mathcal{X}}$ was chosen such that $\widehat{\mathcal{X}} \geq m\log\widehat{\mathcal{X}}/\alpha^2$, the algorithm never runs out of elements of $\widehat{\mathcal{X}}$ to permanently assign. Because $|\widehat{\mathcal{X}}|$ depends only on the desired accuracy α and the sparsity parameter m, and *not* on $\mathcal{X}$ in any way, the algorithm can be implemented and run without any knowledge of $\mathcal{X}$ (even for infinite universes), and neither the running time nor the resulting accuracy depend on $|\mathcal{X}|$. We emphasize that the advantage of this approach over an alternative technique of running multiplicative weights on a small random projection of the data universe, is that our approach works even with *adaptively chosen queries.* Hashing the data-universe to a smaller size using standard methods would no longer permit guarantees for queries that may be adaptively chosen.

The non-interactive mechanism releases a random projection of the database into polynomially many dimensions, together with the corresponding projection matrix. Queries are evaluated by computing their projection using the public projection matrix, and then taking the inner product of the projected query and the projected database. The difficulty comes because the projection matrix projects vectors from $|\mathcal{X}|$-dimensional space to poly(n) dimensional space, and so normally would take $|\mathcal{X}|$poly(n)-many bits to represent. Our algorithms are constrained to run in time poly(n), however, and so we need a concise representation of the projection matrix. We achieve this by using a matrix implicitly generated by a family of limited-independence hash functions which have concise representations. This requires using a limited independence version of the Johnson-Lindenstrauss lemma, and of concentration bounds. This algorithm also gives accuracy bounds which are independent of $|\mathcal{X}|$.

1.3 Related Work

Differential privacy was introduced by Dwork, McSherry, Nissim, and Smith [DMNS06], and has since become the standard solution concept for privacy in the theoretical computer science literature. There is now a vast literature concerning differential privacy, so we mention here only the most relevant work, without attempting to be exhaustive. Dwork et al. [DMNS06] also introduced the *Laplace* mechanism, which is able to efficiently answer arbitrary low-sensitivity queries in the interactive setting. The Laplace mechanism does not make efficient use of the *privacy budget* however, and can answer only linearly many queries in the database size.

Blum, Ligett, and Roth [BLR08] showed that in the non-interactive setting, it is possible to answer *exponentially* sized families of counting queries. This result was extended and improved by Dwork et al. [DNR$^+$09] and Dwork, Rothblum, and Vadhan [DRV10], who gave improved running time and accuracy bounds, and for (ϵ, δ)-differential privacy gave similar results for arbitrary low sensitivity queries. Roth and Roughgarden [RR10] showed that accuracy bounds comparable to [BLR08] could be achieved even in the *interactive* setting, and this result was improved in both accuracy and running time by Hardt and Rothblum, who give the multiplicative weights mechanism, which achieves nearly optimal accuracy and running time [HR10]. Gupta, Roth, and Ullman [GRU12] generalize the algorithms of [RR10, HR10] into a generic framework in which objects called *iterative database constructions* efficiently reduce to private data release mechanisms in the interactive setting. Unfortunately, the running time of all of the algorithms discussed here is at least linear in $|\mathcal{X}|$, and so typically exponential in the size of the private database. Moreover, there are both computational and information theoretic lower bounds suggesting that it may be very difficult to give private release algorithms for generic linear queries with substantially better run time [DNR$^+$09, UV11, GHRU11]. As in this work, these algorithms give a guarantee on the worst-case error of any answered query.

There is also a small body of work giving more efficient query release mechanisms for specific classes of queries. [BLR08] gave an efficient (running time polynomial in the database size n) algorithm for releasing the answers for 1-dimensional intervals on the discretized unit-line in the non-interactive setting. As far as we know, prior to this work, this was the only efficient mechanism in either the interactive or non-interactive settings for releasing the answers to an exponentially sized family of queries with worst-case error. This class is however structurally very simple: it has VC-dimension only 2. Other efficient algorithms relax the notion of utility, no longer guaranteeing worst-case error for all queries. [BLR08] also give an efficient algorithm for releasing *halfspace* queries in the unit sphere, but this algorithm only guaranteed accurate answers for halfspaces that happened to have large *margin* with respect to the points in the database. Gupta et al [GHRU11] gave an algorithm for releasing *conjunctions* over d attributes to *average* error α over any product distribution (over conjunctions), which runs in time $d^{O(1/\alpha)}$. This was improved to have running time $O(d^{\log 1/\alpha})$ by Cheraghchi et al. [CKKL12]. Note that these algorithms only run in polynomial time for constant values of α, and only give accuracy bounds in expectation over random queries. Recently, Hardt, Rothblum, and Servedio [HRS12] gave an algorithm for releasing conjunctions defined on k out of d literals with an average-error guarantee *for any* pre-specified distribution in time $d^{\tilde{O}(\sqrt{k})}$. Using the private boosting algorithm of [DRV10], they leverage this result to give an algorithm for releasing k-literal conjunctions with worst-case error guarantees, which increases the running time to $d^{\tilde{O}(k)}$, although still only requiring databases of size $d^{\tilde{O}(\sqrt{k})}$. They also gave an efficient (i.e. running time polynomial in n) algorithm for releasing *parity* queries to low average error over product distributions. We remark that our results give a complementary bound for large conjunctions (with a better sample complexity requirement). Our online algorithm can release all conjunctions on $d-k$ out of d literals with worst-case error guarantees in time $d^{\tilde{O}(k)}$, requiring databases of size only $\tilde{O}(k^{1.5}\log d)$.

The efficient interactive mechanism we give in section 3 is based on an analogy between iterative database construction (IDC) algorithms and online learning algorithms in the mistake bound model. We implement the multiplicative weights IDC of Hardt and Rothblum [HR10] analogously to how Winnow is implemented in the *infinite attribute model* of Blum [Blu90]. In our setting, it can be thought of as an *infinite universe model* that has no dependence on the universe size in either the running time or accuracy bounds. This involves running the multiplicative weights algorithm on a much smaller universe. Hardt and Rothblum [HR10] also gave a version of their algorithm which ran on a small subset of the universe to give efficient run-time guarantees. The main difference is that we select the subset of the universe that we run the multiplicative weights algorithm on adaptively, based on the queries that arrive, whereas [HR10] select the subset nonadaptively, independently of the queries. [HR10] give average case utility bounds for linear queries on randomly selected databases; in contrast, we give worst-case utility bounds that hold for all input databases, but only for sparse linear queries.

The efficient non-interactive mechanism we give in section 4 is based on random projections using families of limited independence hash functions, which have previously been used for space-bounded computations in the streaming model [CW09, KN10]. Limited independence hash functions have also previously been used for streaming algorithms in the context of differential privacy [DNP+10].

2 Preliminaries

A database $\mathcal{D}$ is a multiset of elements from some (possibly infinite) abstract universe $\mathcal{X}$. We write $|\mathcal{D}| = n$ to denote the cardinality of $\mathcal{D}$. For any $x \in \mathcal{X}$ we can also write $D[x]$ to denote: $\mathcal{D}[x] = |\{x' \in \mathcal{D} : x' = x\}|$ the number of elements of type x in the database. Viewed this way, a database $\mathcal{D} \in \mathbb{N}^{|\mathcal{X}|}$ is a vector with integer entries in the range $[0, n]$.

A linear query $Q : \mathcal{X} \to [0, 1]$ is a function mapping elements in the universe to values on the real unit interval. For notational convenience, we will define $Q(\emptyset) = 0$. We can also evaluate a linear query on a database. The value of a linear query Q on a database is simply the average value of Q on elements of the database:

$$Q(\mathcal{D}) = \frac{1}{n}\sum_{x\in\mathcal{D}} Q(x) = \frac{1}{n}\sum_{x\in\mathcal{X}} Q(x)D[x]$$

Similarly to how we can think of a database as a vector, we can think of a query as a vector $Q \in [0,1]^{|\mathcal{X}|}$ with $Q[x] = Q(x)$. Viewed this way, $Q(\mathcal{D}) = \frac{1}{n}\langle Q, \mathcal{D}\rangle$.

It will sometimes be convenient to think of normalized databases (with entries that sum to 1). For a database $\mathcal{D}$ of size n, we define the corresponding normalized database $\hat{\mathcal{D}}$ to be the database such that $\hat{\mathcal{D}}[x] = \mathcal{D}[x]/n$. We evaluate a linear query on a normalized database by computing $Q(\hat{\mathcal{D}}) = \sum_{x \in \mathcal{X}} Q(x)\hat{\mathcal{D}}[x] = \langle Q, \hat{\mathcal{D}}\rangle$. Note that $Q(\mathcal{D}) = Q(\hat{\mathcal{D}})$.

Definition 1 (Sparsity). *The* sparsity *of a linear query Q is $|\{x \in \mathcal{X} : Q(x) > 0\}|$, the number of elements in the universe on which it takes a non-zero value. We say that a query is m-sparse if its sparsity is at most m. We will also refer to the class of all m-sparse linear queries, denoted $\mathcal{Q}_m$.*

In this paper, we will assume that given an m-sparse query, we can quickly (in time polynomial in m) enumerate the elements $x \in \mathcal{X}$ on which $Q(x) > 0$.

Remark 1. While the assumption that we can quickly enumerate the non-zero values of a query may not always hold, it is indeed the case that for many natural classes of queries, we can enumerate the non-zero elements in time *linear* in m. For example, this holds for queries that are specified as lists of the universe elements on which the query is non-zero, as well as for many implicitly defined query classes such as conjunctions, disjunctions, parities, etc.[3] Of course, classes like conjunctions are typically not sparse, but conjunctions with $d - O(\log n)$ literals are, and their support can be quickly enumerated (even though there are superpolynomially many such conjunctions).

2.1 Utility

We will design algorithms which can accurately answer large numbers of sparse linear queries. We will be interested in both *interactive* mechanisms and *non-interactive* mechanisms. A non-interactive mechanism takes as input a database, runs one time, and outputs some data structure capable of answering many queries without further interaction with the data release mechanism. An interactive mechanism takes as input a stream of queries, and must provide a numeric answer to each query before the next one arrives.

Definition 2 (Accuracy for non-Interactive Mechanisms). *Let $\mathcal{Q}$ be a set of queries. A non-interactive mechanism $M : \mathcal{X}^* \rightarrow R$ for some abstract range R is (α, β)-*accurate for *$\mathcal{Q}$ if there exists a function* Eval $: \mathcal{Q} \times R \rightarrow \mathbb{R}$ *s.t. for every database $\mathcal{D} \in \mathcal{X}^*$, with probability at least $1 - \beta$ over the coins of M, $M(\mathcal{D})$ outputs $r \in R$ such that $\max_{Q \in \mathcal{Q}} |Q(\mathcal{D}) - \text{Eval}(Q, r)| \leq \alpha$. We will abuse notation and write $Q(r) = \text{Eval}(Q, r)$.*

[3] The set of conjunctions over the d-dimensional boolean hypercube with $d - log(n)$ literals are n-sparse. Even though there are superpolynomially many such conjunctions, it is simple to enumerate the entries on which these conjunctions take non-zero value in time linear in n. We can simply enumerate all of the $2^{\log n} = n$ values that the unassigned variables can take.

M is efficient *if both M and* Eval *run in time polynomial in the size of the database n.*

Definition 3 (Accuracy for Interactive Mechanisms). *Let $\mathcal{Q}$ be a set of queries. An interactive mechanism M takes as input an adaptively chosen stream of queries $Q_1, \ldots, Q_k \in \mathcal{Q}$ and for each query Q_i, outputs an answer $a_i \in \mathbb{R}$ before receiving Q_{i+1}. It is (α, β)-*accurate *if for every database $\mathcal{D} \in \mathcal{X}^*$, with probability at least $1 - \beta$ over the coins of M: $\max_i |Q_i - a_i| \leq \alpha$.*

M is efficient *if the update time for each query (i.e. the time to produce answer a_i after receiving query Q_i) is polynomial in the size of the database n.*

2.2 Differential Privacy

We will require that our algorithms satisfy *differential privacy*, defined as follows. We must first define the notion of *neighboring databases.*

Definition 4 (Neighboring Databases). *Two databases $\mathcal{D}, \mathcal{D}'$ are* neighbors *if they differ only in the data of a single individual: i.e. if their symmetric difference is $|\mathcal{D} \triangle \mathcal{D}'| \leq 1$.*

Definition 5 (Differential Privacy [DMNS06]). *A randomized algorithm M acting on databases and outputting elements from some abstract range R is (ϵ, δ)-differentially private if for all pairs of neighboring databases $\mathcal{D}, \mathcal{D}'$ and for all subsets of the range $S \subseteq R$ the following holds:*

$$\Pr[M(\mathcal{D}) \in S] \leq \exp(\epsilon) \Pr[M(\mathcal{D}') \in S] + \delta$$

Remark 2. For a non-interactive mechanism, R is simply the set of data-structures that the mechanism outputs. For an interactive mechanism, because the queries may be adaptively chosen by an adversary, R is the set of query/answer transcripts produced by the algorithm when interacting with an arbitrary adversary. For a detailed treatment of differential privacy and adaptive adversaries, see [DRV10].

Additional preliminaries can be found in the full version [BR11].

3 A Fast IDC Algorithm for Sparse Queries

In this section we use the abstraction of an *iterative database construction* that was introduced by Gupta, Roth, and Ullman [GRU12]. It was shown in [GRU12] that efficient IDC algorithms automatically reduce to efficient differentially private query release mechanisms in the interactive setting. Roughly, an IDC mechanism works by maintaining a sequence of data structures $\mathcal{D}_1, \mathcal{D}_2, \ldots$ that give increasingly good approximations to the input database $\mathcal{D}$ (in a sense that depends on the IDC). Moreover, these mechanisms produce the next data structure in the sequence by considering only one query Q that *distinguishes* the real database in the sense that $Q(\mathcal{D}_t)$ differs significantly from $Q(\mathcal{D})$.

Syntactically, we will consider functions of the form $\mathbf{U} : \mathcal{R}_{\mathbf{U}} \times \mathcal{Q} \times \mathbb{R} \to \mathcal{R}_{\mathbf{U}}$. The inputs to $\mathbf{U}$ are a data structure in $\mathcal{R}_{\mathbf{U}}$, which represents the current data structure $\mathcal{D}_t$; a query Q, which represents the distinguishing query, and may be restricted to a certain set $\mathcal{Q}$; and also a real number which estimates $Q(\mathcal{D})$. Formally, we define a *database update sequence* , to capture the sequence of inputs to $\mathbf{U}$ used to generate the database sequence $\mathcal{D}_1, \mathcal{D}_2, \ldots$.

Definition 6 (Database Update Sequence). *Let $\mathcal{D} \in \mathbb{N}^{|\mathcal{X}|}$ be any database and let $\left\{(\mathcal{D}_t, Q_t, \widehat{A}_t)\right\}_{t=1,\ldots,T} \in (\mathcal{R}_{\mathbf{U}} \times \mathcal{Q} \times \mathbb{R})^T$ be a sequence of tuples. We say the sequence is an* $(\mathbf{U}, \mathcal{D}, \mathcal{Q}, \alpha, T)$-database update sequence *if it satisfies the following properties:*

1. $\mathcal{D}_1 = \mathbf{U}(\emptyset, \cdot, \cdot)$,
2. *for every* $t = 1, 2, \ldots, T$, $|Q_t(\mathcal{D}) - Q_t(\mathcal{D}_t)| \geq \alpha$,
3. *for every* $t = 1, 2, \ldots, T$, $\left|Q_t(\mathcal{D}) - \widehat{A}_t\right| < \alpha$,
4. *and for every* $t = 1, 2, \ldots, T-1$, $\mathcal{D}_{t+1} = \mathbf{U}(\mathcal{D}_t, Q_t, \widehat{A}_t)$.

Definition 7 (Iterative Database Construction). *Let $\mathbf{U} : \mathcal{R}_{\mathbf{U}} \times \mathcal{Q} \times \mathbb{R} \to \mathcal{R}_{\mathbf{U}}$ be an update rule and let $B : \mathbb{R} \to \mathbb{R}$ be a function. We say $\mathbf{U}$ is a $B(\alpha)$-*iterative database construction for query class $\mathcal{Q}$ *if for every database $\mathcal{D} \in \mathbb{N}^{|\mathcal{X}|}$, every $(\mathbf{U}, \mathcal{D}, \mathcal{Q}, \alpha, T)$-database update sequence satisfies $T \leq B(\alpha)$.*

Note that the definition of an $B(\alpha)$-iterative database construction implies that if $\mathbf{U}$ is a $B(\alpha)$-iterative database construction, then given any maximal $(\mathbf{U}, \mathcal{D}, \mathcal{Q}, \alpha, T)$-database update sequence, the final database $\mathcal{D}_T$ must satisfy $\max_{Q \in \mathcal{Q}} |Q(\mathcal{D}) - Q(\mathcal{D}_T)| \leq \alpha$ or else there would exist another query satisfying property 2 of Definition 6, and thus there would exist a $(\mathbf{U}, \mathcal{D}, \mathcal{Q}, \alpha, T+1)$-database update sequence, contradicting maximality.

$B(\alpha)$-IDC algorithms generically reduce to (ϵ, δ)-differentially private (α, β)-accurate query release mechanisms in an efficiency preserving way. This framework was implicitly used by [RR10] and [HR10].

Theorem 3 ([GRU12]). *If there exists a $B(\alpha)$-IDC algorithm for a class of queries $\mathcal{Q}$ using a class of datastructures $\mathcal{R}_{\mathbf{U}}$ that take time at most $p(n, \alpha, |\mathcal{X}|)$ to update their hypotheses, and time at most $q(n, \alpha, |\mathcal{X}|)$ to evaluate a query on any $\mathcal{D} \in \mathcal{R}_{\mathbf{U}}$, then for any $0 < \epsilon, \delta, \beta < 1$ there exists an (ϵ, δ)-differentially private query release mechanism in the interactive setting that has update time at most $O(p(n, \alpha, \mathcal{X}) + q(n, \alpha, \mathcal{X}))$ and is (α, β)-accurate with respect to any adaptively chosen sequence of k queries from $\mathcal{Q}$ where α is the solution to the following equality: $\alpha = \frac{3000\sqrt{B(\alpha)}\log(4/\delta)\log(k/\beta)}{\epsilon n}$*

In this section we will give an efficient IDC algorithm for the class of m-sparse queries, and then call on Theorem 3 to reduce it to a differentially private query release mechanism in the interactive setting.

First we introduce the Sparse Multiplicative Weights data structure, which will be the class of datastructures $\mathcal{R}_{\mathbf{U}}$ that the Sparse Multiplicative Weights IDC algorithm uses.:

Definition 8 (Sparse Multiplicative Weights Data Structure). *The sparse multiplicative weights data structure $\mathcal{D}^{SMW}$ of size s is composed of three parts. We write $\mathcal{D}^{SMW} = (\mathcal{D}, h, ind)$.*

1. *$\mathcal{D}$ is a collection of s real valued variables $x_1, \ldots, x_s$, with $x_i \in [0,1]$ for all $i \in [s]$. Variable x_i for $i \in [s]$ is referenced by $\mathcal{D}[i]$. Initially $x_i = 1/s$ for all $i \in [s]$. We define $\mathcal{D}[i] = 0$ for all $i > s$.*
2. *h is a hash function $h : \mathcal{X} \to [s] \cup \emptyset$ mapping elements in the universe X to indices $i \in [s]$. Elements $x \in \mathcal{X}$ can also be unassigned in which case we write $h(x) = \emptyset$. Initially, $h(x) = \emptyset$ for all $x \in \mathcal{X}$ We write $h^{-1}(i) = x$ if $h(x) = i$, and $h^{-1}(i) = \emptyset$ if there does not exist any $x \in \mathcal{X}$ such that $h(x) = i$.*
3. *$ind \in [s+1]$ is a counter denoting the index of the first unassigned variable. For all $i < ind$, there exists some $x \in \mathcal{X}$ such that $h(x) = i$. For all $i \geq ind$, there does not exist any $x \in \mathcal{X}$ such that $h(x) = i$. Initially $ind = 1$.*

If $ind \leq s$, we can add *an unassigned element $x \in \mathcal{X}$ to $\mathcal{D}^{SMW}$. Adding an element $x \in \mathcal{X}$ to $\mathcal{D}^{SMW}$ sets $h(x) \leftarrow ind$ and increments $ind \leftarrow ind + 1$. If $ind = s + 1$, attempting to add an element causes the data structure to report **FAILURE**.*

A linear query Q is evaluated on a sparse MW data structure $\mathcal{D}^{SMW} = (\mathcal{D}, h)$ as follows.

$$Q(\mathcal{D}^{SMW}) = \sum_{x \in \mathcal{X} : Q(x) > 0 \wedge h(x) \neq \emptyset} Q(x) \cdot \mathcal{D}[h(x)] + \sum_{x \in \mathcal{X} : Q(x) > 0 \wedge h(x) = \emptyset} Q(x) \cdot \mathcal{D}[ind]$$

We now present Algorithm 3, the Sparse Multiplicative Weights (SMW) IDC algorithm for m-sparse queries. The algorithm is a version of the Hardt/Rothblum Multiplicative Weights IDC [HR10], modified to work without any dependence on the universe size. It will run multiplicative weights update steps over the variables of the SMW data structure, using the SMW data structure to delay assigning variables to particular universe elements $x \in \mathcal{X}$ until necessary. Note that it is not simply running the multiplicative weights algorithm from [HR10] implicitly: doing so would yield guarantees that depend on the cardinality of the universe $|\mathcal{X}|$. Instead, the guarantees we will get will depend only on m, and so will carry over even to the infinite-universe setting.

Theorem 4. *The Sparse Multiplicative Weights algorithm is a $B(\alpha)$-IDC for the class of m-sparse queries $\mathcal{Q}_m$, where:*

$$B(\alpha) = 4\frac{\log s + 1}{\alpha^2}$$

and s is the smallest integer such that $s/(\log(s) + 1) \geq 4m/\alpha^2$.

The analysis largely follows the Multiplicative Weights analysis given by Hardt and Rothblum [HR10]. The main difference is that rather than using one global potential function, we must use a different potential function for each database update sequence, defined as a function of the state of the hash table in the last SMW datastructure in the sequence. We must also argue that we never run

```
SMW($\mathcal{D}_t^{\mathrm{SMW}} = (\mathcal{D}_t, h_t, \mathrm{ind}_t), Q_t, \widehat{A}_t$):
1. If $\mathcal{D}_t^{\mathrm{SMW}} = \emptyset$
   (a) Let $s$ be the smallest integer such that $s/(\log(s)+1) \geq 4m/\alpha^2$.
   (b) Return a new Sparse MW data structure $\mathcal{D}_1^{\mathrm{SMW}} = (D_1, h_1, \mathrm{ind}_1)$ of size $s$
       with $h_1(x) = \emptyset$ for all $x \in \mathcal{X}$, $x_i = 1/s$ for all $i \in [s]$, and $\mathrm{ind}_1 = 1$.
2. Let $\mathcal{D}_{t+1}^{\mathrm{SMW}} = (\mathcal{D}_{t+1}, h_{t+1}, \mathrm{ind}_{t+1}) \leftarrow \mathcal{D}_t^{\mathrm{SMW}}$
3. Update: For all $x \in \mathcal{X}$ such that $Q_t(x) > 0$: If $h_{t+1}(x) = \emptyset$ then add $x$ to
   $\mathcal{D}_{t+1}^{\mathrm{SMW}}$.
4. If $\widehat{A}_t < Q_t(\mathcal{D}_t^{\mathrm{SMW}})$ Then Update: For all $x \in \mathcal{X}$ such that $Q_t(x) > 0$: Let
   $\mathcal{D}_{t+1}[h_{t+1}(x)] \leftarrow \mathcal{D}_{t+1}[h_{t+1}(x)] \cdot \exp(-\eta Q_t(x))$
5. Else Update: For all $x \in \mathcal{X}$ such that $Q_t(x) > 0$: Let $\mathcal{D}_{t+1}[h_{t+1}(x)] \leftarrow$
   $\mathcal{D}_{t+1}[h_{t+1}(x)] \cdot \exp(\eta Q_t(x))$
6. Normalize: For all $i \in [s]$: $\mathcal{D}_{t+1}[i] = \frac{\mathcal{D}_{t+1}[i]}{\sum_{j=1}^{s} \mathcal{D}_{t+1}[j]}$
7. Output $\mathcal{D}_{t+1}^{\mathrm{SMW}}$.
```

Fig. 1. The Sparse Multiplicative Weights (SMW) IDC Algorithm for m-sparse queries, adapted from the MW IDC of [HR10]. It is instantiated with an accuracy parameter $\eta = \alpha/2$. It takes as input a sparse MW datastructure $\mathcal{D}^{\mathrm{SMW}}$, an m-sparse query $Q \in \mathcal{Q}_m$, and an estimate of the query value $\widehat{A}$.

out of variables to assign in the SMW data structure, which would cause it to return **FAILURE**. To argue this, we apply the technique of Blum Hellerstein and Littlestone [BHL95], used to adapt Winnow to the infinite attribute model. The proof appears in the full version.

Finally, we may observe that both the update time for the SMW IDC and the time to evaluate a query on the SMW datatructure is $O(s) = \tilde{O}(m/\alpha^2)$. Therefore, we may instantiate Theorem 3 with the SMW IDC algorithm to obtain the main result of this section:

Theorem 5. *For any $0 < \epsilon, \delta, \beta < 1$ there exists an (ϵ, δ)-differentially private query release mechanism in the interactive setting, with running time per query $\tilde{O}(m/\alpha^2)$ that is (α, β)-accurate with respect to the set of all m-sparse linear queries $\mathcal{Q}_m$, with:*

$$\alpha = O\left(\frac{(\log m)^{1/4}\left(\log \frac{4}{\delta} \log \frac{k}{\beta}\right)^{1/2}}{(\epsilon \cdot n)^{1/2}}\right)$$

Proof. The proof follows by instantiating Theorem 3 with the SMW IDC algorithm, together with the bound $B(\alpha) = \frac{4(\log s+1)}{\alpha^2}$ proven in Theorem 4, and recalling that s is the smallest integer such that $s/(\log s + 1) \geq 4m/\alpha^2$.

3.1 Applications to Conjunctions

In this section, we briefly mention a simple application of this algorithm to the problem of releasing conjunctions with many literals. The algorithm given in this section leads to new results for releasing conjunctions on $d - k$ out of d literals. This complements the recent results of Hardt, Rothblum, and Servedio [HRS12] for releasing conjunctions on k out of d literals. The class of conjunctions are defined over the universe $\mathcal{X} = \{0,1\}^d$ equal to the d-dimensional boolean hypercube.

Definition 9. *A conjunction is a linear query specified by a subset of variables $S \subseteq [d]$, and defined by the predicate $Q_S : \{0,1\}^d \to \{0,1\}$ where $Q_S(x) = \prod_{i \in S} x_i$. We say that a conjunction Q_S has t literals if $|S| = t$.*

Remark 3. The set of all conjunctions of $d-k$ literals, denoted C_{d-k} is 2^k sparse, and of size $|C| \leq d^k$.

We can release the answers to all queries in C_{d-k} by running the sparse multiplicative weights algorithm on each query. We therefore get the following corollary:

Corollary 1. *There exists an (ϵ, δ)-differentially private algorithm in the non-interactive release setting with running time at most*

$$\tilde{O}\left(|C_{d-k}| \cdot \frac{2^k}{\alpha^2}\right) = \tilde{O}\left(\frac{(2d)^k}{\alpha^2}\right)$$

that is (α, β)-accurate for the set of all conjunctions on $d - k$ literals, which requires a database of size only:

$$n \geq \frac{k^{1.5} \log \frac{1}{\delta} \log \frac{d}{\beta}}{\epsilon \alpha^2}$$

We note that the running time of this algorithm is comparable to the running time of the algorithm of [HRS12] for releasing all conjunctions of k out of d literals to worst case error (time roughly $\tilde{O}(|C_k|) = \tilde{O}(d^k)$), but requires a database of size only roughly $k^{1.5} \log d$, rather than $d^{\tilde{O}(\sqrt{k})}$ as required by [HRS12]. Of course, conjunctions on k literals are a more natural class than conjunctions on $d - k$ literals, but the results are complementary.

Moreover, applying the sparse multiplicative weights algorithm in the interactive setting gives polynomially bounded running time per query for conjunctions on $d - k$ literals for any $k = O(\log n)$. Note that this is still a super-polynomially sized class of conjunctions, with $|C_{O(\log n)}| = d^{O(\log n)}$. This is the first interactive query release algorithm that we are aware of that is simultaneously privacy-efficient and computationally-efficient for a super-polynomially sized class of conjunctions (or any other family of queries with super-constant VC-dimension).

4 A Non-interactive Mechanism via Random Projection

In this section, we give a non-interactive query release mechanism for sparse queries based on releasing a perturbed random projection of the private database, together with the projection matrix. Note that when viewing the database $\mathcal{D}$ as a vector, it is an $|\mathcal{X}|$-dimensional object: $\mathcal{D} \in \mathbb{R}^{|\mathcal{X}|}$. A linear projection of $\mathcal{D}$ into T dimensions is obtained by multiplying it by a $|\mathcal{X}| \times T$ matrix, which cannot even be represented explicitly if we require algorithms that run in time polynomial in $n = |\mathcal{D}|$ for $n << |\mathcal{X}|$. It is therefore essential that we use projection matrices which can be represented concisely using hash functions drawn from limited-independence families.

We will use a limited-independence version of the Johnson-Lindenstrauss lemma presented in [KN10], first proven by [Ach01, CW09].

Theorem 6 (The Johnson-Lindenstrauss Lemma with Limited Independence [Ach01, CW09, KN10]). *For $d > 0$ an integer and any $0 < \varsigma, \tau < 1/2$, let A be a $T \times d$ random matrix with $\pm 1/\sqrt{T}$ entries that are r-wise independent for $T \geq 4 \cdot 64^2 \varsigma^{-2} \log(1/\tau)$ and $r \geq 2\log(1/\tau)$. Then for any $x \in \mathbb{R}^d$: $\Pr_A[|||Ax||_2^2 - ||x||_2^2| \geq \varsigma ||x||_2^2] \leq \tau$*

We will use the fact that random projections also preserve pairwise inner products. The following corollary is well known:

Corollary 2. *For $d > 0$ an integer and any $0 < \varsigma, \tau < 1/2$, let A be a $T \times d$ random matrix with $\pm 1/\sqrt{T}$ entries that are r-wise independent for $T \geq 4 \cdot 64^2 \varsigma^{-2} \log(1/\tau)$ and $r \geq 2\log(1/\tau)$. Then for any $x, y \in \mathbb{R}^d$: $\Pr_A[|\langle (Ax), (Ay) \rangle - \langle x, y \rangle| \geq \frac{\varsigma}{2}(||x||_2^2 + ||y||_2^2)] \leq 2\tau$*

Definition 10 (Random Projection Data Structure). *The random projection datastructure $\mathcal{D}_r$ of size T is composed of two parts: we write $\mathcal{D}_r = (u, f)$.*

1. *$u \in \mathbb{R}^T$ is a vector of length T.*
2. *$f : [|\mathcal{X}| \cdot T] \to \{-1/\sqrt{T}, 1/\sqrt{T}\}$ is a hash function implicitly representing a $T \times |\mathcal{X}|$ projection matrix $A \in \{-1/\sqrt{T}, 1/\sqrt{T}\}^{T \times |\mathcal{X}|}$. For any $(i, j) \in T \times |\mathcal{X}|$, we write $A[i, j]$ for $f(|\mathcal{X}| \cdot (i-1) + j)$.*

To evaluate a linear query Q on a random projection datastructure $\mathcal{D}_r = (u, f)$ we first project the query and then evaluate the projected query. To project the query we compute a vector $\widehat{Q} \in \mathbb{R}^T$ as follows. For each $i \in [T]$ $\widehat{Q}[i] = \sum_{x \in \mathcal{X}: Q(x) > 0} Q[x] \cdot A[i, x]$ Then we output: $Q(\mathcal{D}_r) = \frac{1}{n}\langle \widehat{Q}, u \rangle$.

Theorem 7. *SparseProject is (ϵ, δ)-differentially private.*

Theorem 8. *For any $0 < \epsilon, \delta < 1$, and any $\beta < 1$, and with respect to any class of m-sparse linear queries $\mathcal{Q} \subset \mathcal{Q}_m$ of cardinality $|\mathcal{Q}| \leq k$, SparseProject is (α, β)-accurate for: $\alpha = \tilde{O}\left(\log\left(\frac{k}{\beta}\right) \frac{\sqrt{m \log\left(\frac{1}{\delta}\right)}}{\epsilon n}\right)$ where the $\tilde{O}$ hides a term logarithmic in $(m + n)$.*

4.1 Applications to Conjunctions

In this section, we again briefly briefly mention a simple application of our non-interactive mechanism to the problem of releasing conjunctions with many literals. This gives the first polynomial time algorithm for non-interactively releasing a super-polynomially sized set of conjunctions.

```
**SparseProject**$(\mathcal{D}, \epsilon, \delta, \beta, m, k)$

1. **Let** $\tau \leftarrow \frac{\beta}{4k}$, $T \leftarrow 4 \cdot 64^2 \cdot \log\left(\frac{1}{\tau}\right)\left(\frac{m^{3/2}}{2} + \frac{n^4}{2\sqrt{m}} + \sqrt{m}n^2\right)$, $\sigma \leftarrow \frac{\epsilon}{\sqrt{8\ln(1/\delta)}}$
2. **Let** $f$ be a randomly chosen hash function from a family of $2\log(kT/2\beta)$-wise
   independent hash functions mapping $[T \times |\mathcal{X}|] \to \{-1/\sqrt{T}, 1/\sqrt{T}\}$. Write $A[i,j]$
   to denote $f(|\mathcal{X}| \cdot (i-1) + j)$.
3. **Let** $u, \nu \in \mathbb{R}^T$ be a vectors of length $T$.
4. **For** $i = 1$ to $T$
   (a) **Let** $u_i \leftarrow \sum_{x:\mathcal{D}[x]>0} \mathcal{D}[x] \cdot A[i,x]$
   (b) **Let** $\nu_i \leftarrow \mathrm{Lap}(1/\sigma)$
5. **Output** $\mathcal{D}_r = (u + \nu, f)$.
```

Fig. 2. SparseProject takes as input a private database $\mathcal{D}$ of size n, privacy parameters ϵ and δ, a confidence parameter β, a sparsity parameter m, and the size of the target query class k

Remark 4. The set of all conjunctions of $d-k$ literals, denoted C_{d-k} is 2^k sparse, and of size $|C_{d-k}| \leq d^k$.

Sparseproject therefore gives the following corollary:

Corollary 3. *There exists an (ϵ, δ)-differentially private algorithm in the non-interactive release setting with polynomially bounded running time, that is (α, β)-accurate for the class of conjunctions $C_{d-\log n}$ on $d - \log n$ literals for:* $\alpha = \tilde{O}\left(\left(\log n \log d + \log \frac{1}{\beta}\right) \frac{\sqrt{\log\left(\frac{1}{\delta}\right)}}{\epsilon\sqrt{n}}\right)$

Note that $C_{d-\log n}$ is a super-polynomially sized set of conjunctions. As far as we know, this represents the first algorithm in the non-interactive setting with non-trivial accuracy guarantees for a super-polynomially sized set of conjunctions that also achieves polynomial running time.

References

[Ach01] Achlioptas, D.: Database-friendly random projections. In: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, p. 281. ACM (2001)

[BHL95] Blum, A., Hellerstein, L., Littlestone, N.: Learning in the presence of finitely or infinitely many irrelevant attributes. JCSS 50(1), 32–40 (1995)

[BLR08] Blum, A., Ligett, K., Roth, A.: A learning theory approach to non-interactive database privacy. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, pp. 609–618. ACM (2008)

[Blu90] Blum, A.: Learning boolean functions in an infinite attribute space. In: Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing, pp. 64–72. ACM (1990)

[BR11] Blum, A., Roth, A.: Fast private data release algorithms for sparse queries. arXiv preprint arXiv:1111.6842 (2011)

[CKKL12] Cheraghchi, M., Klivans, A., Kothari, P., Lee, H.K.: Submodular functions are noise stable. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1586–1592. SIAM (2012)

[CW09] Clarkson, K.L., Woodruff, D.P.: Numerical linear algebra in the streaming model. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 205–214. ACM (2009)

[DMNS06] Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 265–284. Springer, Heidelberg (2006)

[DNP$^+$10] Dwork, C., Naor, M., Pitassi, T., Rothblum, G.N., Yekhanin, S.: Pan-private streaming algorithms. In: Proceedings of ICS (2010)

[DNR$^+$09] Dwork, C., Naor, M., Reingold, O., Rothblum, G.N., Vadhan, S.: On the complexity of differentially private data release: efficient algorithms and hardness results. In: Proceedings of the 41st Annual ACM Symposium on the Theory of Computing, pp. 381–390. ACM, New York (2009)

[DRV10] Dwork, C., Rothblum, G.N., Vadhan, S.: Boosting and differential privacy. In: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, pp. 51–60. IEEE (2010)

[GHRU11] Gupta, A., Hardt, M., Roth, A., Ullman, J.: Privately Releasing Conjunctions and the Statistical Query Barrier. In: Proceedings of the 43rd Annual ACM Symposium on the Theory of Computing. ACM, New York (2011)

[GRU12] Gupta, A., Roth, A., Ullman, J.: Iterative constructions and private data release. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 339–356. Springer, Heidelberg (2012)

[HR10] Hardt, M., Rothblum, G.N.: A multiplicative weights mechanism for privacy-preserving data analysis. In: 51st Annual IEEE Symposium on Foundations of Computer Science, pp. 61–70. IEEE (2010)

[HRS12] Hardt, M., Rothblum, G.N., Servedio, R.A.: Private data release via learning thresholds. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 168–187. SIAM (2012)

[HT10] Hardt, M., Talwar, K.: On the Geometry of Differential Privacy. In: The 42nd ACM Symposium on the Theory of Computing, STOC 2010 (2010)

[KN10] Kane, D.M., Nelson, J.: A derandomized sparse johnson-lindenstrauss transform. arXiv preprint arXiv:1006.3585 (2010)

[RR10] Roth, A., Roughgarden, T.: Interactive Privacy via the Median Mechanism. In: The 42nd ACM Symposium on the Theory of Computing, STOC 2010 (2010)

[UV11] Ullman, J., Vadhan, S.: PCPs and the hardness of generating private synthetic data. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 400–416. Springer, Heidelberg (2011)