When and How Can Data be Efficiently Released with Privacy?

Cynthia Dwork Moni Naor* Omer Reingold Guy N. Rothblum
Salil Vadhan

Abstract

We consider private data analysis in the setting in which a trusted and trustworthy curator, having obtained a large data set containing private information, releases to the public a "sanitization" of the data set that simultaneously protects the privacy of the individual contributors of data and offers utility to the data analyst. The sanitization may be in the form of an arbitrary data structure, accompanied by a computational procedure for determining approximate answers to queries on the original data set, or it may be a "synthetic data set" consisting of data items drawn from the same universe as items in the original data set; queries are carried out as if the synthetic data set were the actual input. In either case the process is non-interactive; once the sanitization has been released the original data and the curator play no further role.

Blum *et al.* (STOC '08) showed the remarkable result that, for any any set X of potential data items and any "concept" class $\mathcal{C}$ of functions $f : X \to \{0, 1\}$, the *exponential mechanism* of McSherry and Talwar (FOCS '07) can be used to (inefficiently) generate a synthetic data set that maintains approximately correct fractional counts for all concepts in $\mathcal{C}$, while ensuring a strong privacy guarantee.

In this work we investigate the computational complexity of non-interactive privacy mechanisms, mapping the boundary between feasibility and infeasibility. Let κ be a computation parameter. We show

1. When $|\mathcal{C}|$ and $|X|$ are both polynomial in κ, it is possible to efficiently (in κ) and privately construct synthetic data sets maintaining approximately correct counts, even when the original data set is very small (roughly, $O(2^{\sqrt{\log |\mathcal{C}|}} \log |X|)$).
2. When either $|\mathcal{C}|$ or $|X|$ is superpolynomial in κ there exist distributions on data sets and a choice of $\mathcal{C}$ for which, assuming the existence of one-way functions, there is no efficient private construction of a synthetic data set maintaining approximately correct counts.
3. Turning to the potentially easier problem of privately generating a data structure from which it is possible to approximate counts, there is a tight connection between hardness of sanitization and the existence of *traitor tracing* schemes, a method of content distribution in which (short) key strings are assigned to subscribers in a way that, given any useful "pirate" key string constructed by a coalition of malicious subscribers, it is possible to identify at least one colluder. Using known schemes, we obtain a distribution on databases and a concept class permitting inefficient sanitization (via Blum *et al.*), but for which no efficient sanitization is possible (under certain complexity assumptions).

Our algorithmic results give for the first time insight into the value of synthetic data sets beyond that of general data structures.

Keywords: privacy, differential privacy, exponential mechanism, traitor tracing, cryptography.

*E-mail: moni.naor@weizmann.ac.il

1 Introduction

We consider private data analysis in the setting in which a trusted and trustworthy *curator*, having obtained a large data set containing private information, releases to the public a "sanitization" of the data set that simultaneously protects the privacy of the individual contributors of data and offers utility to the data analyst. The literature refers to this as the *non-interactive* model, since once the curator has released the sanitization there is no further use for either the original data or the curator.

There has be a series of negative results concerning privacy, roughly saying that there is a class of queries with the property that it is blatantly non-private (allowing almost full reconstruction) if "too many" queries receive "overly accurate" responses [DN03, DMT07, DY08]. These results have been viewed as saying that one cannot answer a linear the database size, number of queries with small noise while preserving privacy. This motivates an interactive approach to private data analysis where the number of queries is limited to be small — sublinear in the size of the dataset X. Intuitively, in the interactive approach only the questions actually asked receive responses, while in the non-interactive approach, if there is no way to anticipate what will be of interest to the analyst, all questions must receive relatively accurate responses, which, by the aforementioned results, leads to blatant non-privacy.

Against this backdrop, Blum, Ligett, and Roth revisited the non-interactive case from a learning theory perspective [BLR08] and contradicted the above interpretation about the necessity of limiting the number of queries to be sublinear. Let X be a universe of data items and $\mathcal{C}$ be a "concept" class consisting of efficiently computable functions $f : X \to \{0, 1\}$. Given a database $x \in X^n$, Blum *et al.* employ the exponential mechanism of McSherry and Talwar [MT07] to (inefficiently) obtain a *synthetic database* that remarkably maintains approximately correct fractional counts for *all* concepts in $\mathcal{C}$, while ensuring a very strong privacy guarantee. That is, letting y denote the synthetic database produced, with high probability over the choices made by the curator, for every concept $c \in \mathcal{C}$, the fraction of elements in y that satisfy c is approximately the same as the fraction of elements in x that satisfy c.[1]

While the Blum *et al.* result is striking in its generality and its privacy/utility tradeoff, a direct implementation of the mechanism takes time superpolynomial in $|X|$ and $|\mathcal{C}|$, even though objects being manipulated are of bit-length only $\text{poly}(n, \log |X|, \log |\mathcal{C}|)$. In this paper, we investigate the computational feasibility of non-interactive privacy mechanisms, delineating the boundary between efficient and inefficient sanitization.

Synthetic Databases: Very roughly, we show that if either the universe X of data items or the concept class $\mathcal{C}$ is of size superpolynomial in a computation parameter κ, then there exists a distribution on databases and a concept class $\mathcal{C}$ for which there is no efficient (in κ) mechanism for privately generating synthetic databases. It follows that there is no general efficient implementation of the exponential mechanism. In contrast, if both the concept class and the data universe are of size polynomial in κ then not only is there an efficient mechanism, but the size of the input database can be surprisingly small, namely $|C|^{o(1)} \cdot \log |X|$ (or even $O(2^{\sqrt{\log |\mathcal{C}|}} \log |X|)$).

Non-synthetic case: What about the seemingly easier problem of efficiently and privately generating a *data structure* that for each $c \in \mathcal{C}$, yields an approximation to the fraction of items in

[1] This does not contradict the negative results because of the size of the error in the case of attacks using a polynomial number of queries, or the size of the input database in the case of attacks using an exponential number of queries.

the database that satisfy c? Here we show a tight connection between hardness of sanitization and the existence of *traitor tracing* schemes in cryptography [CFN94]. See Appendix B for full details. Informally, a traitor tracing scheme is a content distribution protocol that permits a set of subscribers, each receiving a "decoder" containing a key string, to decipher protected content (think movies), while simultaneously denying access to non-subscribers. Coalitions of subscribers are discouraged from distributing pirate decoders by the *tracing* property: by feeding carefully constructed but ordinary-looking enciphered content to the decoder and observing when it succeeds in correctly decoding it is possible to identify at least one member of the traitorous coalition. Traitor-tracing schemes exist under standard computational assumptions, making it hard to construct an untraceable pirate decoder.

The intuition behind the connection between traitor tracing and non-interactive sanitization is as follows. Think of each row in a database as containing the key string of a subscriber. The database can therefore be viewed as the set of key strings known to a traitorous coalition. The (huge) concept class will be the set of all possible ciphertexts. A key string k satisfies a concept c if the decryption of c under k ends in 1 (failure to decrypt results in 0). Still speaking intuitively, private sanitization corresponds to combining a collection of key strings to create an untraceable pirate decoder. Using known tracing schemes based on certain complexity assumptions, we obtain a distribution on databases and a concept class permitting an inefficient sanitization (via the Blum *et al.* [BLR08] methods), but for which no efficient sanitization is possible. With this intuition we show that the ability to privately create a synthetic database translates into the ability to create an untraceable pirate decoder.

Surprisingly we also show that the implication holds in the reverse case: hardness of sanitizing (of sorts) implies traitor tracing schemes (of sorts). To say that sanitizing is hard we need a distribution $\mathcal{D}$ on databases and a concept class $\mathcal{C}$. We think of a database as a source of key strings; from learning theory we know that if we choose a relatively small random subset z of the rows of the database then with high probability over this choice we will maintain approximate fractional counts with respect to $\mathcal{C}$. Thus, we will generate key strings by choosing small random subsets of the rows of the database. We will design an encryption scheme where decryption is equivalent to computing approximate counts on random concepts. Once we do this, the intuition is that a decryption box can be used to compute approximate counts, so viewing this box as a sanitization of the database we conclude (because sanitizing is hard) that the decryption box can be "traced" to the keys (database items) that were used to create it.

In the remainder of the Introduction we restrict our attention to the complexity of creating privacy-preserving *synthetic* databases.

Notation. We let X denote the universe of data items (database rows), $\mathcal{C}$ denote the concept class, and $\mathcal{D}$ denote a distribution on databases. The non-interactive privacy mechanism is denoted A; typically A will operate on an input database $x \in X^n$, producing a synthetic database $y \in X^m$. Let κ be a computation parameter.

1.1 Negative Results for Privacy-Preserving Synthetic Data Sets

We use cryptography to obtain our negative results. We do not define privacy yet; instead, we will argue that in each case considered below, any privacy mechanism producing synthetic databases that provides utility, here captured by approximately preserving fractional counts, is essentially forced to output at least one of the rows in the input database, thereby compromising the privacy of this row.

To get a taste of our hardness results, assume the existence of an existentially unforgeable signature scheme in which, given a set of signatures under a given key, it is infeasible to generate a new signature, either of one of the messages in the set or of any other message (known as strong unforgeability or super-secure, see Section 6.5.2 in [Gol04]). The class $\mathcal{C}$ for which it is hard to come up with privacy-preserving synthetic data contains one concept for each verification key vk. The data universe X consists of the set of all possible (message, signature) pairs. Assume messages and signatures have length polynomial in κ. A data item (m, σ) belongs to the concept c if $\mathit{Verify}(vk, m, \sigma) = 1$, *i.e.*, if σ is a valid signature for m according to verification key vk. Each database in $\mathcal{D}$ will be a set of valid (message, signature) pairs, where all signatures are under the *same* signing key. Let $x \in_R \mathcal{D}$ be a database, and let s be the signing key used, with corresponding verification key vk. Assuming that the sanitizer has produced y, it must be the case that almost all rows of y are valid signatures under vk (because the fractional count of x for the concept vk is 1). By the strong unforgeability of the signature scheme all of these must come from the input database x, contradicting (any reasonable notion of) privacy.

In the example, both $\mathcal{C}$ (the set of verification keys) and X (the set of (message, signature) pairs) are large. In Theorems 3.1 and 3.2 we show that we can obtain hardness results even if *either* of these is large while the other has size polynomial in κ. In other words, there is no general method for efficient generation of privacy-preserving synthetic data, assuming the existence of one-way functions if $|\mathcal{C}|$ of $|X|$ is $\exp(\kappa)$.

1.2 Positive Results for Privacy-Preserving Data Sets

Our principal positive result is an algorithm for privately generating synthetic data sets when both $\mathcal{C}$ and X are of size $\mathrm{poly}(\kappa)$. One measure of the quality of such an algorithm is the question of how large the input data set must be for the algorithm to work, as a function of the sizes of the concept class and the data universe, and the error and privacy parameters. One can also ask, for a given size n of input databases (and error and privacy parameters), for how large a concept class $\mathcal{C}$ does the algorithm work. Ignoring the error and privacy parameters, our algorithm requires only $n \geq O(2^{\sqrt{\log |\mathcal{C}|}} \log |X|)$, that is, it works whenever $|\mathcal{C}|$ is up to quasi-polynomial in n (but still polynomial in κ).

The significance of the smallness of n as a function of the size of the concept class becomes clear when we consider the variant of the interactive privacy mechanism of [DMNS06], in which noise is added to the true answer of each query according to the L_2 sensitivity of the sequence. Without going into details, for that technology to achieve constant fractional accuracy requires $|\mathcal{C}| < n^2$, as compared with our positive results for $|\mathcal{C}|$ quasi-polynomial in n (but $\mathrm{poly}(\kappa)$).

To put this in perspective, the techniques of [DN04, DMNS06] were designed for the *nouveau data riche*, where data are numerous and (absolute, not fractional) distortion that depends only on the query sequence, and not on n, essentially yields privacy for free. However, when n is smaller, having techniques yielding distortion only, say, $n^{2/3}$, for numbers of counting queries far exceeding n (but still polynomial in κ), yields usable analyses where previous techniques would be overwhelmed by noise.

The algorithm relies on several tools. Suppose we have a data structure that provides, for each concept $c \in \mathcal{C}$, an approximation to the fraction of the original data set x that satisfies c. The first tool is a "syntheticizer", i.e. a method for converting these approximate fractional counts to an actual synthetic data set (Theorem C.2). The algorithm uses linear programming, with a constraint for each concept; it also requires enumerating each data type in the universe, so it is only efficient (in κ) when $|\mathcal{C}|$ and $|X|$ are of size $\mathrm{poly}(\kappa)$. If the fractional counts provided by the data structure

	$\lvert C\rvert \in \exp(\kappa)$	$\lvert C\rvert \in \text{poly}(\kappa)$
$\lvert X\rvert \in \exp(\kappa)$	Average-case hard, Theorem 3.1	Average-case hard, Theorem 3.1
$\lvert X\rvert \in \text{poly}(\kappa)$	Worst-case hard for $n < \sqrt{\lvert X\rvert}$, Theorem 3.2 Data Structure for $n > \sqrt{\lvert X\rvert}$, Theorem C.1 Avg.-case Synth DB for $n > \log(\lvert X\rvert)$, Theorem C.7	Synth DB for $n > 2^{\sqrt{\log(\lvert C\rvert)}}$, Theorem 4.2

Table 1: (Roughly) computational complexity of sanitizing with synthetic output, database size n, constant utility and privacy.

are correct to within an additive α (that is, they all agree with the fractional counts in the original data set x to within α), the resulting synthetic data set will be correct to within an additive 4α.

The second tool is an average case result for *concepts* (Theorem C.3). Let $\mathcal{C}$ be our concept class. The result says that if we choose a small subset of concepts $\mathcal{C}' \subset \mathcal{C}$, then, with high probability over the choice of $\mathcal{C}'$, any sufficiently small(!) multiset $T \subset X$ that has good accuracy for the concepts in $\mathcal{C}'$ has, in fact, good accuracy for "almost all" of $\mathcal{C}$. Specifically, if the set T of data items with good accuracy for $\mathcal{C}'$ can be described by m bits, then the set of concepts on which T does poorly is of order $m|\mathcal{C}|/|\mathcal{C}'|$. (This is similar "Occam's razor" arguments in computational learning theory.)

Using the first tool, the "syntheticizer", we see that if we have a sanitizer that can produce general output (not necessarily a synthetic data set) for every subset $\mathcal{C}' \subset \mathcal{C}$ of size up to, say, s, then we can randomly choose an s-element $\mathcal{C}'$, run this sanitizer, apply the syntheticizer to obtain a synthetic data set with good accuracy on $\mathcal{C}'$, and finally *subsample this synthetic data set* to obtain a small synthetic data set, denoted y, still with good accuracy on $\mathcal{C}'$. We can then apply the second tool to observe that we have good accuracy on most of $\mathcal{C}$.

Unfortunately, we cannot improve this by repeatedly choosing different subsets $\mathcal{C}' \subset \mathcal{C}$ and combining the results: there may exist "underprivileged" concepts c for which good performance on no $\mathcal{C}'$ not containing c yields good performance on c. However, since the poorly-handled concepts are not numerous we can compute differentially private counts for each of them. Arguing that revealing *which* concepts are not accurately represented by the synthetic data set y does not disturb privacy, we obtain a privacy-preserving data structure for $\mathcal{C}$ with three elements: the synthetic data set y that accurately represents most concepts in $\mathcal{C}$, the approximate counts for those concepts poorly represented by y, and the list of which concepts are in the second category. Finally, we combine all of these via our first tool to obtain a synthetic data set that accurately represents all concepts in $\mathcal{C}$.

These two tools almost give us a recursive procedure for creating a synthetic data set whenever $|\mathcal{C}|$ and $|X|$ are of size $\text{poly}(\kappa)$. The recursive step has been described, and all we lack is a sanitizer for the basis of the construction, when the concept class in the recursion is very small. Thus, our third tool is a simple sanitizer that works whenever n, the size of the input data set, is just slightly larger than the size of the concept class (Claim C.6).

One interesting aspect of this construction is the apparently crucial role of synthetic data. This appears in the second tool: we use the fact that if a (sufficiently small) set of data items has good utility for a random subset $\mathcal{C}' \subset \mathcal{C}$ then it will have good utility for most of $\mathcal{C}$. Thus we can choose a random subset of concepts for the recursive call and get, from the obtained synthetic data set, a lot of mileage with respect to the *complete* set of concepts. If instead the recursive call produced an arbitrary data structure that accurately covered $\mathcal{C}'$, we would not know how to argue that it also gives information about $\mathcal{C} \setminus \mathcal{C}'$.

2 Preliminaries and Definitions

Let $[n]$ be the set $\{1, 2, \ldots n\}$. For $x, y \in \{0,1\}^n$ we use $x \circ y$ to denote the concatenation of x and y (a string in $\{0,1\}^{2n}$). For a (discrete) distribution D over a set X we denote by $x \sim D$ the experiment of selecting $x \in X$ by the distribution D. A function $f(n)$ is *negligible* if it is smaller than any (inverse) polynomial. We refer the reader to [Gol01, Gol04] for complete definitions of standard cryptographic objects used in this work such as one-way functions, distribution ensembles, signature schemes and pseudo-random functions.

2.1 Privacy Preserving Sanitizers

In this work we deal with (non-interactive) methods for releasing information about a database. For a given database x, a (randomized) non-interactive database access mechanism A computes an output $A(x)$ that can later be used to reconstruct information about x. We will be concerned with mechanisms A that are *private* and *useful*, both of these notions are defined formally below.

Definition 2.1 (Differential Privacy [DMNS06])**.** A randomized function A is (ε, δ)-*differentially private* if for any two databases $x \in \{0,1\}^n$ and $x' \in \{0,1\}^n$ such that $\Delta(x, x') = 1$, and for all sets S of possible outputs of A it holds that:

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

If A is $(\varepsilon, 0)$-differentially private (i.e. $\delta = 0$), then we say it is ε-*differentially private*.

See [Dwo08, KS08] for properties and results concerning differential privacy.

For a function f, the *(global) sensitivity of* f is the maximal difference in its values on neighboring databases: $\max_{x,x' \text{ neighbors}} |f(x) - f(x')|$. The *Laplace distribution* $Lap(t)$ has density function $h(y) \propto e^{-y/t}$, has mean 0 and standard deviation t. We usually refer to the Laplace distribution over integers. Dwork *et al.* [DMNS06] showed that if a function f has global sensitivity s then the function $f(x) + Lap(s/\varepsilon)$ is ε-differentially private.

Following [BLR08], we consider releasing information that can be used to answer a class of predicate counting queries. We call the class of queries that the information can be used to answer the *concept class*, usually denoted C. We call the set of values that elements in the database can take the *data universe*, usually denoted X. We sometimes work with ensembles of concept classes and data universes, one for every input length parameter. Predicate counting queries are usually referred to as *concepts*. Each such concept is defined by a predicate, and its output on a database x is a count: the number (or fraction) of elements in x that satisfy the predicate. For a concept c we say that $A(x)$ is α-*accurate for* c if the difference between the fractional count that $A(x)$ gives for c and the fractional count x gives for c (sometimes denoted $c(x)$) is at most α. We say that $A(x)$ is (α, γ)-*accurate for a concept class* C if $A(x)$ is α-accurate for a $1 - \gamma$ fraction of the concepts in C (if $\gamma = 0$ we sometimes refer to α-accuracy of $A(x)$ for a class). Our notion of a sanitizer A's utility w.r.t. a concept class is below.

Definition 2.2 ((α, β, γ)-Utility)**.** Let C be a concept class and X a data universe. A sanitizer A has (α, β, γ)-utility for n-item databases w.r.t. C and X if for any n-item database x:

$$\Pr_{A\text{'s coins}}[A(x) \text{ is } (\alpha, \gamma)\text{-accurate}] \leq \beta$$

We are mostly interested in coming up with sanitizers with $\gamma = 0$, however both as an intermediate result and as an exploration of the limitation of hardness we will consider the general case.

2.2 Hardness of Sanitizing

In this section we define what it means for a distribution on databases to be *computationally hard* to sanitize. Intuitively, for a hardness result to be convincing, we would like to say that it is hard to meet even a *weak* notion of privacy (let alone a notion as strong as differential privacy). In fact, we will say that sanitizing is hard if it is hard even to avoid leaking input items in their entirety: i.e. some item's privacy is always blatantly violated (such leakage is a very significant breach of privacy). Note that if leaking a few input items is allowed then sanitizing (with synthetic output) becomes easy: just output a randomly chosen subset of the input items; with high probably this subset will preserve utility even with respect to large sets of counting queries.

A distribution of databases is *hard to sanitize* (with respect to some concept class) if for any efficient alleged sanitizer, with high probability over a database drawn from the distribution, one of the database items can be extracted from the alleged sanitizer's output. Of course, to avoid triviality, we will also require that when this leaked item is excluded from the input database (and, say, replaced by a random different item), the probability that it can be extracted from the output is very small. This means that any efficient (alleged) sanitizer indeed compromises the privacy of input items in a strong sense. A formal definition follows.

Definition 2.3 ($(\mu, \alpha, \beta, \gamma, C)$-Hard-to-Sanitize Database Distribution)**.** Let C be a concept class ensemble, X a data universe ensemble, and $\mu, \alpha, \beta, \gamma \in [0, 1]$. Let n be a database size and D an ensemble of distributions, where D_n is over collections of $n + 1$ items from X_n, we often denote by $(x, x'_i) \sim D$ the experiment of choosing an n-element database and an additional element x'_i from the distribution D. We think of D as specifying a distribution on n-item databases (and their neighbors) that is hard to sanitize.

An algorithm is said to be "efficient" if its running time is $\text{poly}(n, \log(|C_n|), \log(|X_n|))$. We say that D is $(\mu, \alpha, \beta, \gamma, C)$*-hard-to-sanitize* if the following holds:

There exists an efficient algorithm T such that for any alleged efficient sanitizer A the following two conditions hold:

1. With probability $1 - \mu$ over choosing a database x from D and over A's and T's coins, if $A(x)$ maintains α-utility for a $1 - \gamma$ fraction of concepts, then T can recover one of x's items from $A(x)$:

$$\Pr_{(x,x'_i)\sim D,\ A\text{'s},\ T\text{'s coins}}[(A(x) \text{ maintains } (\alpha, \gamma)\text{-utility}) \text{ and } (x \cap T(A(x)) = \emptyset)] \leq \mu$$

2. For *every* efficient algorithm A, and for every $i \in [n]$, if we draw (x, x'_i) from D, and replace x_i with x'_i to form x', T cannot extract x_i from $A(x')$ except with small probability:

$$\Pr_{(x,x'_i)\sim D, A\text{'s},\ T\text{'s coins}}[x_i \in T(A(x'))] \leq \mu$$

We often drop C when it is clear from the context, and refer to distributions that are $(\mu, \alpha, \beta, \gamma)$-hard to sanitize. In most of our examples μ will be a negligible function, and so we use (α, β, γ)-hard to sanitize as shorthand for $(neg(), \alpha, \beta, \gamma)$-hard to sanitize ($C$ will be clear from the context).

Throughout this work we will sometimes place restrictions on the sanitizer, such as always outputting synthetic datasets, and then make claims about the hardness of sanitizers that obey these additional restrictions.

We conclude this section by observing that in particular, if a distribution is hard-to-sanitize a la Definition 2.3, then it is also hard to sanitize while achieving even weak differential privacy. In other words hard to sanitize implies hard to be differentially private (the other implication is not true).

Claim 2.1. *If a distribution D on databases is hard-to-sanitize with respect to $(\mu, \alpha, \beta, \gamma, C)$, where $\mu \leq \min(\beta, (1-8\beta)/(8n^{1+a}))$ for some $a > 0$, then no efficient sanitizer that achieves (α, β, γ)-utility with respect to C on databases drawn from D can achieve $(a\log(n), (1-8\beta)/2n^{1+a})$-differential privacy.*

3 Hardness of Sanitizing with Synthetic Data

We can now formally state our hardness theorems for generating synthetic databases when either $|C|$ or $|X|$ is large (as a function of κ). The proofs are given in Appendix A.

Theorem 3.1. *Let $f : \{0,1\}^\kappa \to \{0,1\}^\kappa$ be a one-way function. There exists a concept class C of size* $\text{poly}(\kappa)$*, a data universe X of size* $\exp(\text{poly}(\kappa))$*, and a distribution on databases of size* $\text{poly}(\kappa)$ *that is $(\mu = neg(\kappa), \alpha, \beta, \gamma, C)$-* hard-to-sanitize*, for any $4\gamma + 2\alpha \leq 1/4$ and any noticeable $1-\beta$.*

The construction builds on the construction for signatures outlined in the Introduction. There the concept class was of exponential size because we considered concepts corresponding to each of the possible verification keys. The new idea is to have just a single concept that verifies the validity of signatures: the verification algorithm *without* the verification key hard-wired in. The verification key will now be included as part of the data items. We construct hard to sanitize databases by choosing a *single* verification key and generating multiple valid signed messages. Now, as in the case above, no sanitizer can output private synthetic data that uses the same verification key as the input database. However, since the verification key is no longer hardwired into the (single) concept, the sanitizer may just generate synthetic data with valid signatures under *a different key* of its choosing. To prevent the adversary from doing so, we add more concepts, so that preserving utility for these concepts forces the sanitizer not to change the verification key used in the input database. This can be done by adding concepts that output the bits of the verification key (or rather the bits of its encoding under a high-distance error correcting code).

Finally, we show that when the data universe is of polynomial (in κ) size but the concept class is exponential in κ, sanitizing with synthetic output (and worst-case utility) is computationally hard.

Theorem 3.2. *Let $f : \{0,1\}^\kappa \to \{0,1\}^\kappa$ be a one-way function. For every $a > 0$, and for every integer $n = \text{poly}(\kappa)$, there exists a concept class C of size* $\exp(\text{poly}(\kappa))$*, a data universe X of size $O(n^{2+2a})$, and a distribution on databases of size n that is $(\mu, \alpha, \beta, 0, C)$-* hard-to-sanitize *(i.e. hard to sanitize for worst-case concepts) for $\alpha \leq 1/3$, $\beta \leq 1/10$ and $\mu = 1/40n^{1+a}$.*

The previous examples used signature schemes, where items in the data universe included a signature. Hardness of sanitizing came from the hardness of finding a new signature. However, when the data universe is of polynomial in κ size the sanitizer can enumerate over all possible data items and we cannot hope to include unforgeable signatures in the data items if the verification key is public (as it was in our previous construction). Instead, in this construction we will use pseudo-random functions [GGM86]: a family of efficiently computable functions, such that a random function from the family is indistinguishable (via black-box access) from truly random functions.

We will look at a family of functions with polynomial-size domain and range. The data items are input-output pairs and there is a concept for every function in the pseudo-random family that accepts an input-output pair iff it is consistent with the function. We generate a hard-to-sanitize database x by choosing a random function f_s in the family, n random inputs, computing the n outputs of f_s on those inputs, and putting the input-output pairs in the database. The intuition is that to maintain utility on the concept corresponding to f_s, most of the items in the output of a sanitizer with synthetic data should be input-output pairs consistent with f_s. But f_s is a pseudorandom function, and so, for inputs to f_s that were not in the initial database, the sanitizer's probability of "guessing" f_s's output is polynomially small. Thus (with high probability) many of the items that were in the input database must also be in the sanitizer's output.

We will see that the restrictions on the sanitizer in Theorem 3.2 are roughly tight by our positive results for exponential in κ concept-classes and polynomial data universes (see Section 4). We construct efficient sanitizers with average-case utility and synthetic output, and so Theorem 3.2 cannot be improved to rule out sanitizers with average-case utility. Interestingly, we construct sanitizers for databases of size at least roughly $\sqrt{|X|}$ with "pseudo-synthetic" output. See Theorem 4.1.

See Table 1 for a complete picture of the interaction between our positive and negative results.

4 Positive Results

We briefly describe our positive results for efficiently sanitizing databases. Details are given in Appendix C.

The following theorem requires $n > \sqrt{|X|} \in \text{poly}(\kappa)$ but has *only logarithmic dependence* on $|C|$. The output of this mechanism is not a synthetic database, but it is close, in the following sense. We think of a type (an element) in the universe X as a binary vector, with $x_{ij} = 1$ if $c_j(x_i) = 1$, and $x_{ij} = 0$ otherwise. For each type x_i we can define the corresponding *negative type* $-x_i$ consisting only of 0's and -1's: $(-x_i)_j = -x_{ij}$. The result below outputs a collection of counts, one per type in X; however, some of the counts may be negative. If we allow negative types, then we can create a "pseudo-synthetic" database with the types in X and negative types.

Theorem 4.1. *Let X be a data universe an C a concept class. There exists an ε-differentially private sanitizer A with $(\alpha, \beta, 0)$-utility that runs in time* $\text{poly}(|X|, n) \cdot \text{polylog}(|C|, 1/\beta, 1/\varepsilon)$ *and sanitizes databases of size $n \geq \lambda \cdot (\sqrt{|X|} \cdot \log(|X|) \cdot \log^2(1/\beta) \cdot \log|C|)/(\varepsilon \cdot \alpha)$ for a universal constant $\lambda > 0$.*

The sanitizer A computes a (differentially private) noisy collection of integer counts specifying how many times every item appears in its input database x. For each type $T \in X$, letting $count(T)$ denote the integer count for T, we place in the pseudo-synthetic database $count(T)$ copies of T if this count is positive, and $count(T)$ copies of $-T$ (that is T with a special notation that this is a negative item) if the count is negative. Let y denote the resulting pseudo-synthetic database.

Given y and a query a concept c, we enumerate over all the items (with multiplicities) in $i \in y$, and release the sum $\sum_{i \in y} c(i)$.

The main intuition for utility is that the errors in the counts of how many database items satisfy a concept, introduced by the noisiness of counts for each element in X, cancel out. This happens because we allow negative types. See Theorem C.1 in Appendix C.1 for details.

Main Positive Result. In the Introduction we described our main positive result: an algorithm for privately constructing a synthetic dataset that works for any polynomial size concept class C and data universe X, provided the database has size at least and $(\log|X|)2^{\Omega(\sqrt{\log|C|})}$ (see Appendix C.4).

Recall that the construction is recursive, that we assume we have a method (the "syntheticizer") for constructing m-bit privacy-preserving synthetic databases for any subset $C' \subset \mathcal{C}$ of size at most s, and that for a randomly chosen $\mathcal{C}' \subset C$, if we run the syntheticizer on $\mathcal{C}'$ then with high probabiltiy the resulting synthetic database gives accurate fractional counts for all but $s_{bad} = m|\mathcal{C}|/|\mathcal{C}'|$ concepts in $\mathcal{C}$. Let C_y denote these unfortunate concepts. We will create differentially private counts for these, each obtained by adding noise $Lap(s_{bad}/2\varepsilon)$. We therefore have a data structure for $\mathcal{C}$ with three elements: the synthetic data set y that privately and accurately represents most concepts in $\mathcal{C}$, the privacy-preserving approximate counts for those concepts poorly represented by y, and the list of which concepts are in the second category. We now discuss how to create this last – the list of concepts in C_y – in a privacy-preserving fashion.

Let w be the (true, not noisy) indicator vector describing which concepts are in $\mathcal{C}_y$. To create a (noisy) indicator bit vector v, we use the exponential mechanism, where the distance function $d_{c,y}$ for a concept c and an output y (of A_{small}) is:

$$d_{c,y}(x,0) = |c(x) - c(y)| - 1.5\alpha$$

$$d_{c,y}(x,1) = 1.5\alpha - |c(x) - c(y)|$$

This function has sensitivity $1/n$ (recall $c(x)$ and $c(y)$ are fractional counts). Now using the exponential mechanism, for each concept c we output bit b with probability proportional to $Pr[v_c = b] \propto e^{-\varepsilon \cdot n \cdot d_{c,y}(x,b)/2s_{bad}}$. The intuition is that if the difference between c's counts on x and y is smaller than 1.5α, then 0 becomes more likely (exponentially fast in the distance), whereas if the difference is larger than 1.5α, then 1 becomes more likely. Note that, since the sensitivity of the distance function $d_{c,y}$ is at most $1/n$, each of the bits v_c is (on its own) an $\varepsilon/2s_{bad}$-differentially private function of the input (for any possible output y and concept c). We first show (Claim C.5) that the bit v_c indeed provides a good indication of whether the output y accurately approximates the concept c's count. Specifically, for concepts c that y either "fits" very well (about α-close to their counts in x), or for those that y fits very poorly (about 2α-far), their indicator bit v_c is with overwhelming probability either 0 or 1. Moreover, this is even true for the given y even when we compute the v_c bits using a neighboring database x' of x.

Releasing v may, at first glance, seem problematic: there are $|C|$ concepts and when computing v_c for each such concept, we are only adding "noise" on the magnitude of $s_{bad}/2\varepsilon$ (rather than $|C|/\varepsilon$). A crucial observation, however, is that since for almost all the concepts v_c will very "strongly" tend to be 0 both for the input database x and for any neighboring database x', we can let the (negligible) δ term in (ε, δ)-differential privacy handle the extremely unlikely case that, for some good concept $c \notin \mathcal{C}_y$ the exponential mechanism assigns output 1 to v_c.

On the other hand, assuming $|\mathcal{C}_y| \leq s_{bad}$, for each of the concepts $c \in \mathcal{C}_y$, we have $\varepsilon/2s_{bad}$ privacy for the bit v_c, so over all in this case we have $\varepsilon/2$ privacy for this "bad" portion of v.

In summary, the intuition is as follows. Letting the "good" part of v be the entries of concepts not in $\mathcal{C}_y$, and the "bad" part of v be the entries of concepts in $\mathcal{C}_y$, in the "good" part of the vector v we are getting $(0, \delta)$-differential privacy, and on the "bad" part of v we are getting $(\varepsilon/2, 0)$-differential privacy. In fact, the δ term will also cover the probability that $\mathcal{C}_y$ is unexpectedly large.

Theorem 4.2. *Let C be a concept class and X a data universe. For any desired $\alpha, \varepsilon > 0$, parameter $k \geq |C|$, there is an there is an $(\varepsilon, neg(k))$-differentially private sanitizer A with $(\alpha, neg(k), 0)$-utility for databases of size n, provided that*

$$n \geq \lambda \cdot 2^{6\sqrt{\log(|C|)}} \cdot (\log^8(k) \cdot \log(|X|) \cdot \log(1/\alpha))/(\alpha^3 \cdot \varepsilon)$$

Where $\lambda > 0$ is a universal constant. A's runtime is $\mathrm{poly}(k, |X|, n, 1/\alpha, 1/\varepsilon)$.

Details appear in Appendix C.

5 Conclusions and Open Problems

We have provided an almost complete characterization of when general methods for efficiently manufacturing a synthetic database maintaining approximately correct fractional counts for *all* concepts in $\mathcal{C}$, while ensuring a strong privacy guarantee, exist. If either the data universe or the concept class is exponential (or even super-polynomial) in κ then we have given a very strong indication that no general construction exists; specifically, under standard computational assumptions there are cases for which the task is infeasible. On the other hand for the case in which the data universe and concept class are both polynomial in κ, we gave an algorithm that works provided the database is not too small, or put in another way, provided the number of concepts is no more than some super-polynomial in the data-base size.

This raises some obvious issues. A technical point is whether it is possible to lower the bound on the minimum database size needed for our positive results to polylogarithmic in the sizes of the universes of concepts and data. It is conceivable that a different type of recursive algorithm, perhaps with fewer calls to the linear programming procedure for synthesizing database (Section C.2) or a better method for creating the indicator vector v, would yield such a result.

Another issue is related to the amount of noise/accuracy needed and whether it can be made sufficiently small for interesting applications. In general, from the results about near full reconstruction [DN03, DMT07, DY08], we know that accuracy of $o(1/\sqrt{n})$ is out of the question; in this work and [BLR08] the bound is roughly $1/n^{1/3}$. The important question is in what scenarios is such noise acceptable. For instance what about a small scale medical experiment? What about a web based survey? Can we obtain better accuracy? Can we reach accuracy $1/\sqrt{n}$?

For the case when both the data universe and concept classe are of size $\mathrm{poly}(\kappa)$, we saw that there is no real difference between the ability to sanitize with arbitrary representation and with synthetic datasets. However, when either is large, the tasks are not equivalent.

When both are large, the hardness of sanitizing is tightly related to that of constructing traitor tracing schemes. The question here is whether there are tracing traitor schemes that are *fully* resilient and yet have keys and ciphertext lengths independent of the number of users. This would yield a relatively small concept class ensemble that cannot be sanitized. Another implication for such schemes would be for showing hardness of learning privately, an issue considered by [KLN$^+$08].

Finally, there are many domains where we do not know of an *efficient* privacy preserving sanitizer. One example is that of half-spaces, where the concepts are half-spaces and the data items are points. [BLR08] gave an efficient algorithm, but one that answers not necessarily for the given half-space query but for a near half space. There are many other geometric type queries where we do not know whether a good algorithm exists and it is conceivable that the tools and mechanism developed for our main positive result would be helpful, especially since they are not inherently expensive (e.g. sampling and rounding).

References

[BGP00] Mihir Bellare, Oded Goldreich, and Erez Petrank. Uniform generation of np-witnesses using an np-oracle. *Inf. Comput.*, 163(2):510–526, 2000.

[BLR08] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to non-interactive database privacy. In *STOC*, pages 609–618, 2008.

[BN08] Dan Boneh and Moni Naor. Traitor tracing with constant size ciphertext. In *ACM Conference on Computer and Communications Security*, pages 501–510, 2008.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, pages 573–592, 2006.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, pages 257–270, 1994.

[DL08] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. Manuscript submitted to this conference, 2008.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevy and Tal Rabin, editors, *First Theory of Cryptography Conference (TCC)*, volume 3876, pages 265–284. Springer-Verlag, 2006.

[DMT07] Cynthia Dwork, Frank McSherry, and Kunal Talwar. The price of privacy and the limits of lp decoding. In *STOC*, pages 85–94, 2007.

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

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

[DN08] Cynthia Dwork and Moni Naor. On the difficulties of disclosure prevention in statistical databases or the case for differential privacy. Unpublished manuscript, 2008.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *EUROCRYPT*, pages 523–540, 2004.

[Dwo08] Cynthia Dwork. Differential privacy: A survey of results. In *TAMC*, pages 1–19, 2008.

[DY08] Cynthia Dwork and Sergey Yekhanin. New efficient attacks on statistical disclosure control mechanisms. In *CRYPTO*, pages 469–480, 2008.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct pseudorandom functions. *Journal of the ACM*, 33(2):792–807, 1986.

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1.* Cambridge University Press, 2001.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2*. Cambridge University Press, 2004.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[IJKW08] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct product theorems: simplified, optimized, and derandomized. In *STOC*, pages 579–588, 2008.

[Jus72] Justesen. A class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18(5):652656, 1972.

[JVV86] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[KLN+08] Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? In *FOCS*, pages 1–19, 2008.

[KS08] Shiva Prasad Kasiviswanathan and Adam Smith. A note on differential privacy: Defining resistance to arbitrary side information. *CoRR*, abs/0803.3946, 2008.

[KY01] Aggelos Kiayias and Moti Yung. Self protecting pirates and black-box traitor tracing. In *CRYPTO*, pages 63–79, 2001.

[Mir08] Ilya Mironov. Personal communication, 2008.

[MT07] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103. IEEE Computer Society, 2007.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC*, pages 33–43, 1989.

[Rom90] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 387–394, 1990.

[Sto85] Larry J. Stockmeyer. On approximation algorithms for #p. *SIAM J. Comput.*, 14(4):849–861, 1985.

[SZ99] Michael E. Saks and Shiyu Zhou. Bp $_{\mathrm{h}}$space(s) subseteq dspace($\mathrm{s}^{3/2}$). *J. Comput. Syst. Sci.*, 58(2):376–403, 1999.

A Hardness Proofs for Synthetic Data

A.1 Hardness for Small Concept Classes

In this section we prove Theorem 3.1, showing a small concept class that is hard to sanitize (the data universe remains large). The construction builds on the "basic" construction outlined above. There the concept class was of exponential size because we considered all possible signature verification circuits. The new idea is to have just a single concept that verifies the validity of signatures: the verification algorithm *without* the verification key hard-wired in. The verification key will now be included as part of the data items. We construct hard to sanitize databases by choosing a *single* verification key and generating multiple valid signed messages. Now, as in the case above, no sanitizer can output private synthetic data that uses the same verification key as the input database. However, since the verification key is no longer hardwired in the (single) concept, the sanitizer may just generate synthetic data with valid signatures under *a different key* of its choosing. To prevent the adversary from doing so, we add more concepts, so that preserving utility for these concepts forces the sanitizer not to change the verification key used in the input database. These concepts will simply output the bits of the verification key (or rather the bits of its encoding under an error correcting code). The details are in the proof below.

Proof of Theorem 3.1. Let $(Gen, Sign, Verify)$ be a signature scheme with keys, messages and signatures of length $\text{poly}(\kappa)$. In fact we need a scheme where is is even hard to forge a new signature of a previously signed message. Such a signature scheme can be constructed from any one-way function by the results of [NY89, Rom90], see also Goldreich's book [Gol04], Section 6.5.2, for the details on modifying the scheme so that generating new signatures of signed messages is hard.

The data universe. The data universe consists of key-message-signature pairs $X = \{(vk, m, s) : vk, m, s \in \{0,1\}^{\text{poly}(\kappa)}\}$.

The concept class. C contains the verification circuit $Verify(vk, m, s)$ that outputs 1 iff s is a valid signature on m under key vk. In addition, we fix an explicit "good" binary error correcting code $E : \{0,1\}^{|vk|} \to \{0,1\}^{100|vk|}$ with fractional distance $1/4$ (say based on the Justesen Code [Jus72] with some amplification); we only need to be able to encode efficiently, not to to correct errors. In addition to $Verify$, the concept class C includes $100 \cdot |vk|$ concepts $\{c_1, \ldots, c_{8|vk|}\}$. The concept c_i on input (vk, m, s) outputs the i-th bit of the encoding of vk, i.e. the bit $E(vk)_i$. For technical reasons, to make arguments about on-average utility, we include the $Verify$ concept $8|vk|$ times in C (we use this to argue that on-average utility implies maintaining utility for $Verify$).

Hard-to-sanitize distribution. To sample from the distribution D on databases, start with the database x: generate a signature-verification key pair $(sk, vk) \leftarrow Gen(1^\kappa)$, and choose n messages $m_1, \ldots, m_n$ uniformly and at random from $\{0,1\}^{\text{poly}(\kappa)}$. Finally, output the database whose i-th entry is the verification key, the i-th message and its valid signature, i.e. the tuple $(vk, m_i, Sig(sk, m_i))$. For generating the $n+1$-th item x'_i, just generate a new message-signature pair (under the same key vk).

Intuition. Let A be some alleged sanitizer with high on-average utility, generate (x, x') as above with verification key vk, and examine the items in $A(x)$. Because A has high utility, for most of the bits of vk's encoding (corresponding to concepts c_i), most of the items in A's output should have verification keys whose encodings agree with $E(vk)$ on those bits. This also implies that many of the items in $A(x)$ will have verification keys whose encodings mostly agree with $E(vk)$. But since E is a high-distance error-correcting code, to agree with $E(vk)$ on most bits, the verification keys

of the items of $A(x)$ have to be *identical* to vk. So we conclude that with high probability, many of the items in $A(x)$ have verification key vk. Now, to maintain utility on the concept *Verify*, most of these items with key vk have to contain valid signatures. Since the signature scheme is unforgeable, *all* of these signatures must also appear in the input x, which intuitively contradicts privacy (see Claim A.1 below). In particular, with noticeable probability the item x_j, which appears in x but not in x', will be in $A(x)$. Again, because the signature scheme is unforgeable, x_j cannot appear in $A(x')$ (except with negligible probability, see claim A.2 below), and thus we obtain a direct contradiction to differential privacy. The formal proof follows.

Claim A.1. *For any alleged PPT sanitizer A with (α, β, γ)-utility, where $4\gamma + 2\alpha \leq 1/4$ and $1 - \beta$ is noticeable::*

$$\Pr_{(x,x'_i)\sim D, A\text{'s coins}}[(A \text{ maintains } (\alpha,\gamma)\text{-utility}) \text{ and } (x \cap A(x) = \emptyset)] = neg(\kappa)$$

Proof. We will prove that for *any* x sampled from D whenever A maintains (α, γ)-utility, the output $A(x)$ includes at least one item that is a valid signature under x's verification key vk. Once we prove this, the claim follows for a random x sampled from D. This is because the alleged sanitizer A only gets as input message-signature pairs (under the random key vk) that are in x, and cannot forge any new message-signature pairs (except with negligible probability).

Suppose A has at most an α fractional error on all but a γ-fraction of concepts (this happens with noticeable probability). In this case A preserves α-utility for at least one of the *Verify* concepts (since $\gamma < 1/2$)) and at least a $(1 - 2\gamma)$-fraction of the c_i-concepts (recall that "half" of the concepts are copies of the *Verify* concept). This means that for a $(1 - 2\gamma)$-fraction of the bits of vk's encoding, a $(1 - \alpha)$-fraction of the items in the output of $A(x)$ have a verification key vk' such that $E(vk)_i = E(vk')_i$. In particular, for a random item in $A(x)$ we expect all but a $2\gamma + \alpha$-fraction of the bits of its encoding to agree with $E(vk)$. So the probability (over items in $A(x)$) that more than $4\gamma + 2\alpha$ of the encoding bits *disagree* with $E(vk)$ is at most $1/2$. For at least half of the items in $A(x)$, the encoding of their verification key agrees with $E(vk)$ in all but $4\gamma + 2\alpha$ locations. But E is an error-correcting code with fractional distance $1/4$, and $4\gamma + 2\alpha < 1/4$, so all these verification keys must be *identical* to vk. Now, since we have α-utility for at least one of the $Verify$ concepts, we get that at least a $(1/2 - 2\alpha)$-fraction of the items in $A(x)$ are valid signatures under the verification key vk.

□

Claim A.2. *For any PPT A and any $i \in [n]$:*

$$\Pr_{(x,x'_i)\sim D, A\text{'s coins}}\left[\exists (vk, m, s) \in E(x') \setminus x' : Verify(vk, m, s) = 1\right] = neg(\kappa)$$

Proof. The claim follows directly from the unforgeability of the signature scheme. Note that the key pair (sk, vk) used to generate the database x' is generated randomly and x' consists only of the verification key vk and valid signatures.

□

Now combining the above two claims, we conclude that the distribution D is $(neg(), \alpha, \beta, \gamma, C)$-hard-to-sanitize.

□

The above theorem shows hardness of sanitizing with synthetic data. Note, however, that when the concept class is small one can always sanitizing by releasing noisy counts for every concept (see Section 4 for details). This, however, is not a sanitization with synthetic data. We conclude that sanitizing for small concepts class (with large data universes) is a task that separates efficient sanitization with synthetic data from efficient sanitization with arbitrary outputs.

A.2 Hardness for Small Data Domains

We now show a hardness result for sanitizing with synthetic data where the data universe is small (but the concept class is large). The previous examples used signature schemes, where items in the data universe included a signature. Hardness of sanitizing came from the hardness of finding a new signature. However, when the data universe is of polynomial size the sanitizer can enumerate over all possible data items and we cannot hope to include unforgeable signatures in the data items if the verification key is public (as it was in our previous construction). Instead, in this construction we will use pseudo-random functions [GGM86]: a family of efficiently computable functions, such that a random function from the family is indistinguishable (via black-box access) from truly random functions.

The result in this section only gives hardness of sanitization (with synthetic data) that maintains utility for worst-case concepts. This is with good reason, as in Section 4 we give positive results for efficiently sanitizing polynomial X and exponential C with utility for average-case concepts, when the data set is large.

Proof of Theorem 3.2. Let $\{f_s\}_{s\in\{0,1\}^{\mathrm{poly}(\kappa)}}$ be a family of pseudo-random functions from $[\ell]$ to $[\ell]$ (think of ℓ as polynomial in κ). Such a family pseudorandom functions can be constructed from any one-way function by the results of [GGM86, HILL99].

The data universe. Consider the data universe of input-output pairs $X = \{(a, b) : a, b \in [\ell]\}$.

The concept class. C will contain a single concept c_s for every function f_s in the family, where c_s accepts an input (a, b) iff $f_s(a) = b$

$$C = \{c_s : s \in \{0,1\}^{\mathrm{poly}(\kappa)}, c_s(a, b) = 1 \text{ iff } f_s(a) = b\}$$

Hard-to-sanitize Distribution. The distribution D on samples a key $s \in \{0,1\}^{\mathrm{poly}(\kappa)}$ and n distinct elements $a_1, \ldots, a_n \in [\ell]$. The i-th entry in the database x is $x_i = (a_i, f_s(a_i))$. To generate the additional element x'_i, choose a new random item $a'_i \in [\ell]$, and replace x_i with the new $x'_i = (a'_i, f_s(a'_i))$.

Intuition. The intuition is that all the items in x satisfy the concept c_s. Any sanitizer with reasonable utility (for worst-case concepts) needs to output many items that satisfy c_s, but it is computationally hard to find any items that satisfy c_s and are not in x (because the pseudorandom function's output is hard to predict). Thus the sanitizer must output many items that are in the input, and this violates differential privacy.

More formally, take T to be the algorithm that receives a database x over elements in X, and for each $a \in [\ell]$ outputs (a, b) for the unique b that a appears with most (taking the lexicographically first b in case of a tie). We prove the following two claims:

Claim A.3. *Assume $\{f_s\} : [\ell] \to [\ell]$ is a family of pseudorandom functions. Then for any alleged PPT sanitizer A:*

$$\Pr_{(x,x'_i)\sim D,\ A\text{'s coins}}[(A \textit{ has } 1/3\textit{-utility on all concepts}) \textit{ and } (x \cap T(A(x)) = \emptyset)] = O(1/\ell + neg(\kappa))$$

Proof. The function f_s is a pseudorandom function (with a randomly generated seed). We conclude that with overwhelming probability over the choice of seed, for any $a \in [\ell]$ that does not appear in x, the alleged sanitizer A cannot predict f_s's value on a any better than it could a random function, i.e. with probability noticeably greater than $1/\ell$. In particular, we expect no more than a $(1/\ell + neg(\kappa))$-fraction of the a's in $A(x)$ that are not in x to appear most frequently with the correct b. The probability that more than say a 1/10-fraction of these a's appear most frequently with the correct b is thus at most $O(1/\ell + neg(\kappa))$.

Suppose this event does not occur. Since all of the items in the input x satisfy the concept c_s (with the secret key used to generate x), if A preserves 1/3-utility (on all concepts), then all but an 1/3-fraction of the items in $A(x)$ satisfy c_s too. This means that 2/3 of them are of the form (a, b), where $b = f_s(a)$. Where can these "correct" (a, b) pairs be coming from? If for every (a, b) in the input most of its appearances in the output are not with the correct b, then many (more than 1/10) of the correct (a, b) pairs in the output must have b appearing with a in the majority of a's appearances in the output. This, again, cannot happen (except with probability $O(1/\ell + neg(\kappa))$ as above).

□

Strengthening Claim A.3. We can actually strengthen our negative example, as claimed in the introduction, so that the probability that any PPT A maintains α-utility on all concepts and $x \cap T(A(x)) = \emptyset$ is negligible. The construction as-is does not meet this stronger requirement, because A can choose one input $a \in [\ell]$, guess the pseudorandom function's output on that input, and output the guess as a synthetic dataset. This succeeds with noticeable probability $1/\ell$ (because the size of the function's output domain is ℓ).

To side-step this attack we change the concept class, adding concepts that ensure that the sanitizer's output must be diverse. In fact, we will ensure that no set T of few, say $\log(n)/100$, types of items (i.e. elements of X) has combined fractional weight more than 2/3 in the sanitizer's output. If we do this, then to maintain utility on the pseudorandom concept without outputting any of the input elements, the sanitizer has to predict the pseudorandom function's output on $\Omega(\log(n))$ different a's. No efficient sanitizer can do this with non-negligible probability.

It still remains to add concepts that ensure that no possible output containing a set of few $(\log(n/100))$ types with fractional weight 2/3 can maintain utility. This can be done by adding n completely random concepts. The input database is large and diverse, and so all its fractional counts on these concepts will be close to 1/2. On the other hand, for any fixed set T of $\log(n/100)$ types, with all but negligible in n probability, at least one of the concepts is 0 on *all* these types. Any output that gives weight 2/3 to the set T will have count at most 1/3 on that concept. Taking a union-bound over the less than $\binom{|X|}{\log(n)}$ possible sets T, with overwhelming probability none of them give good utility for the input database, and so the sanitizer's output must indeed be "diverse". This is a non-explicit construction, but it can be de-randomized (for example, using the pseudo-random function family at hand).

Claim A.4. *Assume $\{f_s\}$ is a family of pseudorandom functions on domain $[\ell]$. Then for any alleged PPT A that runs on databases of size n, and any i, when x is drawn as above using seed s, and $x_i = (a_i, b_i)$ is replaced by x'_i to form x':*

$$\Pr_{(x,x'_i)\sim D, A\text{'s coins}}\left[(a_i, b_i) \in T(A(x'))\right] = O(1/\ell + neg(\kappa))$$

Proof. The claim follows from $\{f_s\}$ being a pseudorandom family. Note that with probability $1 - 1/\ell$ the item (a_i, b_i) does not appear in x' (since the items in x were all distinct). If f_s were

truly random, then clearly no PPT can, on input x', predict its value on a_i with probability greater than $1/\ell$. Thus the probability that when s is chosen at random $A(x')$ predicts $f_s(a_i)$ (where a_i does not appear in x') by outputting it together with a_i in most occurrences is at most $1/\ell + neg(\kappa)$. The total probability of A outputting (a_i, b_i) is (by a union bound) at most $O(1/\ell + neg(\kappa))$.

□

Combining the above two claims, taking $\ell = n^{1+a}$ and taking $\mu = O(1/\ell)$, both of the conditions of Definition 2.3 are met.

□

A.3 NP-Hardness of Implementing the Exponential Mechanism

We conclude with a hardness result about using the exponential mechanism to privately release data, a la [BLR08]. We show a concept class and data universe for which implementing the algorithm of [BLR08] is NP-hard (under a probabilistic reduction). Note that this does not rule out sanitization with synthetic output that uses some other mechanism. The reduction uses ideas from the proof of Theorem 3.1 about hardness of sanitizing for small concept classes.

Theorem A.5. *There exists a concept class ensemble C of polynomial size, a data universe ensemble X of size exponential size, and a distribution D on databases for which it is NP-hard (under probabilistic reductions) to generate the distribution on sanitized synthetic datasets induced by the exponential mechanism. This hardness holds for utility parameters (α, β, γ) whenever $4\gamma + 2\alpha \leq 1/4$ and $1 - \beta$ is noticeable.*

Proof Sketch. We will use an alleged sanitizer A that implements the exponential mechanism to solve the NP-Complete problem SAT. Given a formula ϕ we want to know whether it has a satisfying assignment. For intuition, examine the data universe consisting of all possible assignments. Modify ϕ into a related formula ϕ' that has exactly one more (known) satisfying assignment u, and examine the concept class that has a single concept $c_{\phi'}$ that accepts an assignment if and only if it satisfies ϕ'. Note that this concept class depends on the SAT instance, whereas we will eventually construct a single concept class that we can use for all SAT instances.

Now construct a database x that has n copies of the satisfying assignment u and run the alleged sanitizer A on the database x. We claim that if ϕ is satisfiable, then one of its satisfying assignments will appear in the sanitizer's output with noticeable probability. This is because the distance function used by [BLR08], applied to a potential synthetic database output, will just count the number of satisfying assignments in the output and ignore *which* satisfying assignments are in the output. So for any satisfying assignment v of ϕ, when we use the exponential mechanism to sanitize, the probability that v is in the output is the same as the probability that u is in the output. The sanitizer's output must contain many satisfying assignments to maintain utility, and so we conclude that if ϕ was satisfiable then with noticeable probability the output of A on x will contain an assignment that satisfies ϕ.

Still, in the above we were using a concept class that depended on the SAT instance we wanted to solve. We can construct a single concept class for all instances using the ideas in the proof of Theorem 3.1. The idea is to change the data universe to consist of formula-assignment pairs, and to change the concept class to include the "universal" concept that output whether the given assignment satisfies the given formula, as well as concepts outputting the bits of an error-correcting encoding of the formula. Following the simpler construction above, given a formula ϕ we generate ϕ' with an extra satisfying assignment u. A hard to sanitize database contains multiple copies of the item (ϕ', u). Now, by the analysis in the proof of Theorem 3.1, most of the items in the

output of any sanitizer with synthetic data will have ϕ' as their formula together with a satisfying assignment. If the sanitizer implements the exponential mechanism, then if ϕ is satisfiable we get that with noticeable probability one of these assignment in the output satisfies ϕ.

□

For the other direction, the exponential mechanism can be implemented in the polynomial hierarchy via uniform sampling [Sto85, JVV86, BGP00]. To do this, generate a circuit and associate an exponential set of inputs with each item in the range of the exponential mechanism. For each item in the range, the fraction of the inputs corresponding to it that satisfy the circuit will be exponentially small in its distance, i.e. we want to run the exponential mechanism on input x and distance measure d, for an item y in the range, there is a set T_y of inputs, and the fraction of inputs that satisfy the circuit is proportional to $e^{\varepsilon \cdot d(x,y)}$. Sampling a uniform satisfying assignment to the circuit is exactly sampling an item in the range by the exponential mechanism's distribution.

B Hardness of Sanitizing with Arbitrary Output: The Tracing Traitors Connection

In this section we show a connection between the computational hardness of sanitization (with arbitrary output, i.e. not just with synthetic data) and a well known problem in cryptography: for a natural notion of hardness of sanitizing, constructing hard-to-sanitize datasets is essentially equivalent to constructing traitor-tracing schemes with good parameters. We begin by defining traitor tracing schemes, and then proceed to show both directions of the equivalence in the subsequent subsections.

Traitor Tracing. Traitor tracing schemes enable a publisher to trace a pirate decryption box to a secret key (of a treacherous user) that was used to create the box. For example, consider a content provider who wishes to broadcast his content to a group of subscribers. To do this, he gives each of the subscribers a decryption box that uses a secret key. Unfortunately, a subscriber might now build and distribute a pirate decoder. Traitor tracing schemes ensure that if such a pirate decoding box is found, the content distributer can run a *tracing* algorithm to recover at least one private key used to create the box, and perhaps try to take legal action against this subscriber. Tracing schemes were introduced by Chor, Fiat and Naor [CFN94] and there are many constructions with various choices of the parameters (see [BN08] for recent references).

A (private-key) traitor-tracing scheme consists of algorithms *Setup*, *Encrypt*, *Decrypt* and *Trace*. The *Setup* algorithm generates a key bk for the broadcaster and N subscriber keys $k_1, \ldots, k_N$. The *Encrypt* algorithm encrypts a given bit using the broadcaster's key bk. The *Decrypt* algorithm decrypts a given encryption using any of the subscriber keys. The tracing algorithm gets the key bk and oracle access to a (pirate) decryption box[2] and outputs the index $i \in \{1, \ldots, N\}$ of a key k_i that was used to create the pirate box. An important parameter of a traitor-tracing scheme is its *collusion-resistance*: a scheme is t-resilient if tracing is guaranteed to work as long as no more than t keys are used to create the pirate decoder. When $t = N$, i.e. tracing works even if all the subscribers join forces to try and create a pirate decoder, the scheme is said to be *fully resilient*. The other parameters that we will be especially interested in are the scheme's ciphertext and private key lengths. A fuller definition follows.

Definition B.1 (t-resilient Traitor-Tracing Scheme)**.** A scheme (*Setup*, *Encrypt*, *Decrypt*, *Trace*) as above is a *t-resilient traitor-tracing scheme* if (i) the ciphertexts it generates are semantically secure (see [GM84]), and (ii) no PPT adversary A can "win" in the following game with non-negligible probability (over the coins of *Setup*, A, *Trace*):

The adversary A receives the number of users N and a security parameter κ and (adaptively) requests the keys of up to t users $\{i_1, \ldots, i_t\}$. The adversary then outputs a pirate decoder *Dec*. The *Trace* algorithm is run with the tracing key and black-box access to *Dec*, it outputs the name $i \in [N]$ of a user or the error symbol $\perp$. We say that an adversary A "wins" if it is both the case that *Dec* has a non-negligible advantage in decrypting ciphertexts *and* the output of *Trace* is not in $\{i_1, \ldots, i_t\}$.

A *one-time* t-resilient traitor-tracing scheme is one where semantic security is only guaranteed to hold against adversaries that are given only a single ciphertext (as opposed to being given access to a sequence of encryptions or an encryption oracle).

[2] We assume that the box is stateless; this is without loss of generality by the work of Kiayias and Yung [KY01].

Traitor Tracing and Hardness of Sanitizing. We show that constructing traitor-tracing schemes with good private key size and ciphertext length is roughly equivalent to constructing hard-to-sanitize database distributions.

First we show that traitor tracing schemes can be used to build hard-to-sanitize distributions. We then instantiate this general transformation with a traitor-tracing scheme of Boneh, Waters and Sahai [BSW06], whose security relies on cryptographic hardness assumptions for bilinear groups of composite order. We obtain a concept class and database distribution that can be sanitized in exponential time (via the results of [BLR08]), but are hard to sanitize in polynomial time. These results are discussed and proven in Section B.1.

The connection between traitor-tracing and hardness of sanitization is not accidental; we actually show a strong relationship between these two notions. In fact, they are roughly *equivalent*. This is shown in Section B.2, where we prove that any hard-to-sanitize concept class and database distribution (with the proper parameters) can be used to construct a traitor-tracing scheme.

B.1 Traitor Tracing Implies Hard Sanitizing

In this section we show, in Theorem B.1, that traitor tracing schemes can be used to construct hard-to-sanitize distributions. The result is later used, together with the traitor-tracing scheme of [BSW06], to separate efficient and inefficient sanitization. We first present the theorem and its applications, and then close the section with a proof.

Theorem B.1 (Traitor Tracing Implies Hard Sanitizing)**.** *If there exists a fully resilient (i.e. n-resilient) traitor-tracing scheme with private-key size $ksize = ksize(n, \kappa)$ and cipher-text size $csize = csize(n, \kappa)$, then there exists a concept class ensemble C, where C_n has size 2^{csize}, and a data universe ensemble X, where X_n has size 2^{ksize}, and a database distribution ensemble D, such that for any noticeable function $f(n) = 1/poly(\kappa)$ the ensemble D is $(\mu = neg(n), \alpha = 1/2 - f(n), \beta = 1 - f(n), \gamma = 0, C)$-hard-to-sanitize.*

Separating Efficient and Inefficient Sanitization. Using this theorem, together with a traitor-tracing scheme of Boneh, Sahai and Waters [BSW06], we obtain a hardness-of-sanitizing result. BSW construct a fully resilient (for any coalition size) traitor-tracing scheme with private keys of size $O(\kappa)$ and ciphertexts of length $O(\sqrt{n} \cdot \kappa)$, where κ is a security parameter. The security of the scheme relies on several hardness assumptions over bilinear groups of composite order.[3]

Plugging this construction into Theorem B.1, we obtain for any n and security parameter κ, a data universe X of size $|X| = 2^\kappa$, a concept class of size $|C| = 2^{\sqrt{n} \cdot \kappa}$, and a $(f(n), 1/2 - f(n), 1 - f(n), 0)$-hard-to-sanitize distribution D over databases of n items from X. Compare this with the (exponential-time) sanitizer of BLR: their sanitizer is guaranteed to give differential privacy as long as the database size satisfies $n = \Theta(\log(|C|) \cdot \log(|B|))$ which is $\Theta(\sqrt{n} \cdot \kappa^2)$ (here we take $\alpha, \beta, \varepsilon$ to be constant). Taking the security parameter to be a small polynomial in n, we conclude that the task of sanitizing databases drawn from the distribution D is *information-theoretically possible* (for unbounded sanitizers), but *computationally intractable*:

Corollary B.2 (Separating Efficient and Unbounded Sanitizers, Informal)**.** *Under the appropriate hardness assumptions on bilinear groups of composite order, there exists a data universe, a concept class and a database distribution where sanitization is possible in exponential time, but impossible in polynomial time.*

[3] Specifically, they need to assume the Decision 3-Party Diffie-Hellman Assumption, the Subgroup Decision Assumption, and the Bilinear Subgroup Decision Assumption. See [BSW06] for a precise statement of these assumptions and the results.

Proof of Theorem B.1. Let $(Setup, Encrypt, Decrypt, Trace)$ be a traitor-tracing scheme as above. We construct a hard to sanitize concept class and database distribution.

The data universe. Consider the data universe of possibly keys $X = \{0,1\}^{ksize(n+1,\kappa)}$.

The concept class. C will contain a concept for every possible ciphertext. I.e. for every $m \in \{0,1\}^{csize(n+1,\kappa)}$. The concept c_m on input a key-string k outputs the decryption of m using the key k (if the messages are longer than one bit, output the least significant bit).

Hard-to-sanitize distribution. The distribution D on databases uses *Setup* to generate $n+1$ decryption keys for the users, n of these keys are used (at random) to generate the n entries of the database x. The $n+1$-th key is the extra item x'_i.

Intuition. The intuition is that for a concept corresponding to a valid encryption of a 0 or 1 message, all the keys in the database decrypt it correctly and output 0 or 1 respectively. So we can view any sanitizer that maintains $(1/2 - f(n), f(n), 0)$ utility as an adversary that with probability $f(n)$ outputs an object that decrypts encryptions of 0 or 1 correctly. We can use the traitor-tracing *Trace* algorithm on such a sanitizer to trace one of the keys in the input of the sanitizer. More formally, we prove the two claims below:

Claim B.3. *For any alleged PPT sanitizer A with $(\alpha = 1/2 - f(n), \beta = f(n), \gamma = 0)$-utility, where $f(n)$ is noticeable:*

$$\Pr_{(x,x'_i)\sim D, A\text{'s coins}}[(A \text{ maintains } \alpha\text{-utility}) \text{ and } (x \cap Trace(A(x)) = \emptyset)] = neg(\kappa)$$

Proof. We treat A as an adversary for the traitor-tracing scheme that gets the n keys in the database x and outputs a sanitization as a decryption box. For a ciphertext c corresponding to a concept, the decryption box uses the sanitization to compute c's count in the input database, outputting 1 if this count is at least $1/2$ and 0 otherwise. For a valid ciphertext of bit b, its fractional count in the input database $c(x)$ is exactly equal to b. If A maintains α-accuracy on all concepts (ciphertexts), where $\alpha < 1/2$, then the decryption will be correct and the *Trace* algorithm will (except with negligible probability) output one of the keys in the input database.

□

Claim B.4. *For any PPT A and any $i \in [n]$:*

$$\Pr_{(x,x'_i)\sim D,\ E\text{'s coins}}\left[x_i \in Trace(A(x'))\right] = neg(\kappa)$$

Proof. The database x' is a random collection of n of the $n+1$ user keys. If on input x' A can output the remaining key x_i with noticeable probability, then we can use A to build a pirate decoder: it takes n of the keys and outputs the $n+1$-th key as its output. Now *Trace* when run on A's output cannot trace any user except the innocent owner of the $n+1$-th key, a contradiction.

□

Combining these two claims, we conclude that the database distribution D is $(neg(\kappa), 1/2 - f(n), f(n), 0, C)$-hard-to-sanitize

□

B.2 Hard Sanitizing Implies Traitor Tracing

In this section we show that hard-to-sanitize database distributions can be used to construct traitor-tracing schemes. Note that we require that it is hard to sanitize even with a small (but noticeable) γ, i.e. it is hard to sanitize while maintaining utility for all but a small fraction of the concepts.

Theorem B.5 (Hard Sanitizing Implies Traitor Tracing)**.** *Let C be an ensemble of concept classes, X an ensemble of data universes, and D a distribution that for every noticeable function $f(n) = 1/\text{poly}(n)$ is $(\mu = neg(n), \alpha = O(1/\log(n)), \beta = 1 - f(n), \gamma = O(1/\log(n)), C)$-hard-to-sanitize.*

Then there exists an $n/\text{polylog}(n)$-resilient one-time traitor tracing scheme, with private keys of size $\log(|X|) \cdot \text{polylog}(n)$ and cipher-text length $\log(|C|) \cdot \text{polylog}(n)$. The scheme is secure against adversaries that run in time $\text{poly}(n, \log(|C|), \log(|X|))$.

Proof. **Intuition.** For a hard-to-sanitize distribution, we are guaranteed that for any sanitizer with good utility we can "trace" one of the items in its input. Given a hard-to-sanitize database distribution and concept class, the idea will be to use a randomly drawn n-item database as a "master key", where the secret used to decrypt messages is the counts of random concepts on this database. To give users private keys, we can randomly partition the database into $n/\text{polylog}(n)$ sets of $\text{polylog}(n)$ items each, each such set will be a "key". These sets are large enough that with overwhelming probability their counts on a random collection of say $\text{polylog}(n)$ concepts are *all* close to the counts of the original database. Our goal will be to design an encryption scheme where decryption is equivalent to computing approximate counts on random concepts. Once we do this, the intuition is that a decryption box can be used to compute approximate counts, viewing this box as a sanitization of the database we conclude (because sanitizing is hard) that the decryption box can be "traced" to the keys (database items) that were used to create it.

There are several challenges. First, the counts of different users on a random concept are close (w.h.p.), but they are not exactly the same, so it is not clear how to encrypt a message so that all the users' different (but close) counts can be used to decrypt it. A similar difficulty was encountered in the work of Dwork and Naor [DN08], where they used fuzzy extractors (see [DRS04]). Such a solution, however, requires publishing public information that depends on the counts, which in our setting is problematic. Instead, we *add noise and round* the counts in a way that ensures that as long as two sets of counts are close they will (with very high probability) be rounded to *exactly* the same rounded counts (in a different setting, a similar idea was used by Saks and Zhou [SZ99]). Now we can extract a hardcore bit of the noisy rounded counts and use it to encrypt a message, and all of the users will be able to decrypt a ciphertext (except with some low error probability). Using this solution (unlike fuzzy extractors) the only information that needs to be "published" in the ciphertext is the noise, which is independent of the database x.

Using the above scheme, any decryption box can be used to compute approximate counts for a large random collection of concepts with noticeable probability. We use a uniform direct-product theorem of [IJKW08] to show that this, in turn, means that a decryption box can be used to build a procedure that with noticeable probability computes approximate counts correctly for all but a small fraction of the concepts. We can treat such a procedure as an alleged sanitizer, and from the hardness of sanitizing the database distribution we can use the algorithm T (from the hardness of sanitizing guarantee) to "trace" a database item in the procedure's input. The user whose key contains that item must have colluded to build the decryption box. A full description and proof follow.

The Scheme. We now describe the *Setup*, *Encrypt* and *Decrypt* algorithms of the traitor tracing scheme. We first analyze these algorithms and then present the *Trace* algorithm and complete the

proof of the theorem. Throughout this section "efficient" algorithms are those that run in time $\text{poly}(n, \log(|C|), \log(|X|))$.

- The *Setup* algorithm chooses an n-item database x from the distribution D. Take $s = \text{polylog}(n)$ to be a size for subsets of items from x. The algorithm forms n/s keys by dividing at random the n items in x into n/s disjoint subsets, each of s items. The i-th player's key is just the i-th subset, and so keys are indeed of size $\log(|X|) \cdot \textit{polylog}(n)$. The broadcast key is simply the collection of all secret keys.

- The *Encrypt* algorithm receives a message bit b to encrypt and a private key which is just a set of s items.[4] It chooses uniformly and at random $\ell = \text{polylog}(n)$ concepts $c_1, \ldots, c_\ell \in C$. For each of these concepts, it computes the fraction of items in the key that satisfy this concept, and then chooses a uniformly random noise value $t \in \{-50/\log(n), -49/\log(n), \ldots, 50/\log(n)\}$ and adds t to all the fractional counts. The *Encrypt* algorithm then rounds all the (noisy) fractional counts down to the nearest multiple of $100/\log(n)$. Let the resulting vector of rounded noisy fractional counts be $v = (v_1, \ldots, v_\ell)$.

  The *Encrypt* algorithm extracts a hard-core bit of v and XORs this hard-core bit with the message bit b in order to encrypt it. To do this, it chooses a random bit vector r s.t. $|r| = |v|$ and outputs the encryption:

$$Enc_{key}(b) = (c_1, \ldots, c_\ell, t, r, \langle v, r \rangle \oplus b)$$

- The *Decrypt* algorithm gets as input a private key and a ciphertext $(c_1, \ldots, c_\ell, t, r, \textit{cipher})$. It proceeds similarly to the *Encrypt* algorithm and computes, using the items from its given private key and $(c_1, \ldots, c_\ell, t)$ from the ciphertext, a vector of rounded noisy fractional counts v'. It then outputs the decrypted message as: $\textit{cipher} \oplus \langle v', r \rangle$.

We begin by showing that the above scheme is correct (namely ciphertext are correctly decrypted when the right keys are used) and also a semantically-secure one-time encryption scheme. We begin with a claim about correctness.

Claim B.6. *For any message b, with probability 9/10 over the coins of the Setup and Encrypt algorithms, for any pair of private keys k_i, k_j we get that $\textit{Decrypt}(k_j, \textit{Encrypt}(k_i, b)) = b$.*

Proof. For every database x of n items, for every concept $c \in C$, for each of the private key subsets k_i (each of size $s = \text{polylog}(n) \cdot \ell^2 = \text{polylog}(n)$), with all but negligible probability, the fractional count of the number of items in the key's subset that satisfy c is within $1/(\ell \cdot \log(n))$ of the fraction of items in the original database x that satisfy c (by a Chernoff bound). Taking a union bound, for *any* collection of concepts $(c_1, \ldots, c_\ell)$, chosen by the *Encrypt* algorithm, with all but negligible probability, all the keys' fractional counts on *all* the concepts are within a range of size $2/(\ell \cdot \log(n))$.

For a given concept c, let us examine this (small) range. We would like to show that by adding uniform random noise t between $-50/\log(n)$ and $50/\log(n)$ and rounding to the nearest multiple of $100/\log(n)$ with high probability the entire range of the counts of the different keys get rounded to the same multiple. What is the probability that this does not occur? It is exactly the probability that there is a multiple of $100/\log(n)$ such that when we add t to it we "land" inside the small range (of size $2/(\ell \cdot \log(n))$) of counts. This probability is at most $1/50\ell$. Taking a union bound

[4] In traitor tracing schemes encryption is usually done using a broadcast key. We construct a scheme where any private key can be used for encryption.

over all ℓ concepts, we get that with probability 1/50 the rounded noisy fractional counts of all the keys are the same. When this occurs, the all the private keys will decrypt the ciphertext correctly. □

We now argue that the scheme is semantically secure. We begin with a helpful claim that will be used to argue both semantic security and traitor tracing (with good parameters). In both of these proofs we will need to transform an algorithm that computes correct approximate counts simultaneously on $\ell = \text{polylog}(n)$ randomly chosen concepts $(c_1, \ldots, c_\ell)$ with non-negligible probability, into a procedure that with noticeable probability over some initial coin tosses correctly computes approximate counts on all but a small fraction of the concepts. This is essentially a *hardness amplification* question, and we use the recent results of [IJKW08], which we restate here for the case of amplifying the hardness of computing approximate counts (usually hardness amplification is used to amplify the hardness of *precisely* computing a function).

Claim B.7 ([IJKW08] restated)**.** *Let M be an efficient procedure that with noticeable probability $1/q(n)$, given $\ell = \text{polylog}(n)$ random concepts $(c_1, \ldots, c_l)$, achieves α-utility on all of the concepts. Then M can be used to construct an efficient procedure M' such that with probability $\Omega(1/q(n))$ (over some initial coin tosses), the procedure M' achieves $3 \cdot \alpha$-utility for all but a $1/\log(n)$-fraction of the concepts.*

Proof Intuition. We do not repeat the proof of [IJKW08], rather we recall its fundamentals and clarify why it applies to the setting of hardness amplification for computing approximate counts.

The IJKW procedure M' (translated to our setting) is as follows: choose a random collection of $\ell/2$ concepts $A = (a_1, \ldots, a_{\ell/2})$ and say we are given a vector of correct α-approximate counts for these concepts $v = (v_1, \ldots, v_{\ell/2})$. Now, M' is given a concept c and wants to compute an approximate count. If $c \in A$ then M' can compute the approximate count, otherwise M' repeats the following process up to $\text{polylog}(n) \cdot q(n)$ times:

Choose $\ell/2 - 1$ additional random concepts to form a collection B of ℓ concepts that includes A and the concept c. Ask M to compute approximate counts on all the concepts in B, if for all the concepts in A these approximate counts are within a 2α-fractional accuracy of the known (and correct) approximate counts in v, then output the count that was given for c. If all iterations fail, output failure (an arbitrary count).

For the above procedure we needed the set v of approximate counts for A. IJKW get A and v by choosing a random collection B_0 of ℓ concepts, choose A to be a random subset of $\ell/2$ concepts in B_0, and take v to be the counts given by M on the collection of concepts B_0. IJKW show that with probability $\Omega(1/q(n))$ the choice of A and B_0 is (in a technical sense) "excellent", and all the following hold:

- M's answer on B_0 is correct and so v has the correct α-approximate counts for all the concepts in A.
- For a random B that contains A, the machine M correctly computes α-approximate counts on B with reasonably high ($\Omega(1/q(n))$) probability. These counts will be 2α-close to the counts in v and so these B's will pass the consistency test (and are numerous).
- For a random B for which M is 2α-close to the counts on v, w.h.p. the machine M's count for most concepts $c \in B - A$ in B will be a 3α-accurate fractional count. Here the intuition is that if many of the counts on B are not 3α-close, then with high probability one of these inaccurate counts will "land" inside A, will not be 2α-close to v's count for the concept, and

will be rejected by the consistency test. This is certainly true for a random $A \subset B$, but IJKW show it is also true conditioned on having accurate answers for A.

So with probability $\Omega(1/q(n))$ the initial A and B_0 that M' chooses are "excellent", and in this case M' achieves 3α-accuracy for all but a $O(1/\log(n))$-fraction of the concepts.

□

With this hardness amplification result in mind, we now turn to proving the (one-time) semantic security of the traitor-tracing scheme. We note that here semantic security actually follows from the traitor-tracing property, but we prove it here separately as a warm-up.

Claim B.8. *Under the conditions of Theorem B.5 there does not exists a PPT distinguisher B that distinguishes encryptions of 0's and 1's with non-negligible probability.*

Proof. Let $1/p(n)$ be B's distinguishing advantage. With probability $1/2p(n)$ over the selection of a random $(c_1, \ldots, c_\ell, t)$, B's distinguishing advantage (for a random r) is at least $1/2p(n)$. Call these the "good" $(c_1, \ldots, c_\ell, t)$-tuples.

For a "good" tuple, by the Goldreich-Levin hardcore-bit theorem [GL89], the distinguisher B can be used to output, with probability $\Omega(p^2(n))$ the "correct" rounded noisy fractional count. Note that with all but negligible probability this count is also a within an $O(1/\log(n))$ accuracy of the *correct* count on database x (see the proof of Claim B.6).

We have just used B to design a procedure that correctly estimates the fractional counts of $\ell = \text{polylog}(n)$ concepts with accuracy $O(1/\log(n))$ and with noticeable probability $\Omega(1/p^3(n))$. By Claim B.7 we can use this procedure to design a new procedure B' that with noticeable probability correctly estimates the fractional counts of all but a $O(1/\log(n))$ fraction of the counts. We can use this procedure B' as a sanitizer, but notice that it never needed to access the database x. Thus, it is trivially perfectly private and also achieves $(O(1/\log(n)), 1 - \text{poly}(1/p(n)), O(1/\log(n)))$-utility. This contradicts the hardness of sanitizing the distribution D.

□

Tracing. We are now ready to present the *Trace* algorithm. The idea is to transform an adversary A that creates a decryption box into an alleged sanitizer for the database x used to generate the keys. We can then use the algorithm T from the hardness-of-sanitization guarantee to trace one of the database items that was in A's collection of keys. Throughout the analysis we assume that for the x used to generate the keys (i.e. chosen according to the hard-to-sanitize distribution D), indeed the two conditions of hardness-of-sanitizing (Conditions 1 and 2 of Definition 2.3) hold. This is guaranteed to occur except with negligible probability.

Let A be an adversary that asks for some of the decryption keys and creates (with noticeable probability $1/p(n)$) a decryption box M that decrypts a random ciphertext correctly with probability $1/2 + \lambda$.

A decryption box can be used to predict the counts of a collection of ℓ concepts with noticeable probability. The *Trace* algorithm first uses the box M to generate a polynomial-size list of algorithms $\{M_1, \ldots, M_{\text{poly}}\}$, one of which correctly predicts the count of a random concept with very high probability. This is done as in Claim B.8. We treat each of these algorithms as a sanitization. *Trace* runs T on each of the algorithms M_j and outputs the index of a user (say the first) whose key contains an item that was in T's output when run on one of these algorithms (or $\perp$ if such a user does not exist).

The analysis will use the fact that one of the M_j's has good utility, and so from the hardness of sanitizing, with overwhelming probability T's output on the "good" M_j has an intersection with

one of the keys. We will also show that with high probability all the items in T's output that are also in x are ones that were in keys used by the traitor-tracing adversary A.

Claim B.9. *Let A be any efficient traitor-tracing adversary that with noticeable probability $1/p(n)$ outputs a decryption box M that has noticeable decryption advantage λ.*

With all but negligible probability, when A generates a decryption box M with decryption advantage λ, the Trace procedure run on the box M outputs one of the keys used by A.

Proof. We use the algorithm T as a traitor-tracing procedure on the decryption box M. As a first step, we view A (the traitor-tracing adversary) as an alleged sanitizer. We know that with probability $1/p(n)$ over the the selection of x and the adversary A's coins, the adversary outputs a "good" decryption box M. This means that for a random tuple $(c_1, \ldots, c_\ell, t)$, with probability at least $\lambda/2$ over the tuple, the distinguishing advantage of M on random ciphertexts created using that tuple is at least $\lambda/2$. By the Goldreich-Levin Theorem, we can use M to output a set of $O(1/\lambda^2)$ count vectors. With high probability, one of these count vectors will be the correct rounded noisy fractional count vector and is thus a $100/\log(n)$-approximation to the counts of the database x on *all of the concepts* $(c_1, \ldots, c_\ell)$.

We proceed using the argument in Claim B.8. The *Trace* algorithm can now use the box M to generate a list of $s = O(n/\lambda^2)$ boxes $M_1, \ldots M_s$ such that with overwhelming probability one of the boxes, say M_j, predicts the count of a random concept with good accuracy with very high probability (again, as in Claim B.8). We treat each of these boxes as a sanitization, and we treat A as a sanitizer that outputs one of the boxes at random. We can thus run the algorithm T on all of these alleged sanitizations.

With probability $1/(s \cdot p(n))$ over the selection of x and the coins of A, it outputs a "good" box M_j, which is a sanitization that is $O(1/\log(n))$-accurate for all but a $O(1/\log(n))$ fraction of the concepts. Viewing A as a sanitizer, by the $(\mu = neg(n), \alpha = O(1/\log(n)), \beta = 1 - f(n), \gamma = O(1/\log(n)))$-hardness of sanitizing D, and as long as $f(n) \geq p(n) \cdot n/\lambda^2$, we get that this alleged sanitizer falls within the hard-to-sanitize regime of parameters. This means that when M is a decryption box with advantage λ, when we run T on the "good" box M_j we constructed from M, with overwhelming probability it outputs some element $x_k \in x$. In particular, with overwhelming probability *Trace* will output the name of *some* user (who has item x_k in his key). It remains to show that with high probability this user was indeed a colluder, i.e. that all of the items in T's output that are also in x are ones that were in user keys used by the traitor-tracing adversary A.

First note that if A got *all* the keys as input then we are done. It remains to show that when A only asks for some (but not all) of the keys, T only outputs items of x that are in one of A's input keys (except with negligible probability).

Suppose this is not the case: A (adaptively) chooses a set S of not all the database items, and outputs a decryption box M with advantage λ. When we run *Trace* on this box, with noticeable probability it outputs an item $x_k \in x$ such that $x_k \notin S$. This will contradict Condition 2 of the hardness of sanitizing D. Let us re-examine that condition, it stipulates that if we replace any $x_i \in x$ with the (unique) item x'_i, the probability that, when we run T on the output of *any efficient procedure* on input x', the output contains x_i is negligible.

To see that A as above contradicts condition 2, consider running the *Trace* algorithm to generate the list of boxes $M_1, \ldots, M_s$ and running T on one of these boxes at random. We assumed (for contradiction), that with noticeable probability T's output contains x_k that was not in the input item set S. Now choose at random an item $i \in [n]$ and replace x_i with x'_i to obtain a new database x'. Whenever the above event occurred and T's output contained x_k, if we run the same procedure on the *modified* x' (where we choose i at random and replace x_i), the probability that $i = k$ and

the procedure's output still contains x_i is at least $1/n$ (because i was chosen uniformly at random). In total, the probability that the output of T on a random box from the collection $M_1, \ldots, M_s$ contains x_i is noticeable, contradicting Condition 2 of the hardness of sanitizing D.

□

□

Additional Remarks. We conclude by noting that the standard notion of traitor-tracing gives a hardness of sanitization result that satisfies the conditions of Theorem B.5. The connection between hardness of sanitizing and traitor tracing is, thus, relatively tight. Still, we find it interesting to examine more closely the parameters in this connection. It is especially interesting to ask whether weaker notions or parameters for hardness of sanitizing results still imply traitor tracing.

We only considered distributions that are hard to sanitize while maintaining utility with any non-negligible probability (roughly, $\beta = 1 - neg$). We could also consider distributions that are only hard to to sanitize while maintaining utility with higher probabilities. Using such a distribution, we could construct traitor-tracing schemes where it is hard for an adversary to generate a decryption box that has (large) noticeable advantage with some (large) noticeable probability for which tracing fails with some (again potentially large) noticeable probability. All three of these probabilities depend on the β for which sanitizing is hard. We note that the overhead incurred by the Goldreich-Levin list decoder and the IJKW hardness amplification becomes more significant in these setting, and reducing it is an interesting open question.

Other interesting directions are dealing with distributions where a database drawn from the distribution is only hard to sanitize with some small (but noticeable) probability (i.e. most databases are easy to sanitize, but a few are hard). In this case, one possible construction of a traitor-tracing scheme could sample many x's for the keys, and encrypt a bit b by secret-sharing it into many bits and encrypting each of the secret shares using a different key. This presents some difficulties because our encryption scheme has a non-negligible error probability that becomes amplified if it is used multiple times. Finally, it is interesting to get a scheme that is semantically secure for adversaries with an encryption oracle (i.e. schemes that can be used any polynomial number of times).

C Positive Results Details

C.1 Sanitizing with Arbitrary Outputs for Dense Datasets

We show that when the size of the data universe is polynomial, we can efficiently sanitize databases whose size is at least roughly $\sqrt{|X|}$. This sanitization *does not* produce a synthetic database, since it can have negative counts for elements. In terms of efficiency can be run in time that is only logarithmic in $|C|$ and polynomial in $|X|$.

Theorem C.1. *Let X be a data universe an C a concept class. There exists an ε-differentially private sanitizer A with $(\alpha, \beta, 0)$-utility that runs in time* $\mathrm{poly}(|X|, n) \cdot \mathrm{polylog}(|C|, 1/\beta, 1/\varepsilon)$ *and sanitizes databases of size $n \geq \lambda \cdot (\sqrt{|X|} \cdot \log(|X|) \cdot \log^2(1/\beta) \cdot \log|C|)/(\varepsilon \cdot \alpha)$ for a universal constant $\lambda > 0$.*

Proof. The sanitizer A will compute a (differentially private) noisy collection of integer counts specifying how many times every item $i \in X$ appears in its input database x. As long as the size n of x is not too small, the noise we add will not effect privacy too significantly.

The sanitizer A, for each data item $i \in X$, computes how many times i appears in the input database x and adds to this count noise according to the Laplace distribution $Lap(2/\varepsilon)$ and releases the vector u of all these counts. This is an ε-differentially private sanitizer because changing a single item in the input database can only change two of the counts by 1 (see [DMNS06]).

For utility, given a concept c we enumerate over all the items $i \in X$, and release the sum

$$\sum_{i \in X : c(i)=1} u_i$$

Note that the noisy data item counts can be negative, and so what we release is not a synthetic database.

What is the utility of the sanitizer? For a given concept c, the error is the sum of the noises we added to the counts of items that satisfy c. The error is thus the sum of up to $|X|$ independent samples from $Lap(2/\varepsilon)$. The probability that the sum of noises is greater than $(\lambda \cdot \sqrt{|X|} \cdot \log(|X|) \cdot \log^2(1/\beta) \cdot \log(|C|))/\varepsilon$ is (for large enough constant $\lambda > 0$) at most $\beta/|C|$. This is because the probability that one of the noise samples is larger than $z = \lambda' \cdot \log(|X|) \cdot \log(1/\beta)$ is (for large enough λ') at most $\beta/2|X|$, so taking a union the probability that *any* of the noise samples is large at above is at most $\beta/2$. Conditioned on none of the noise samples being so large, by Azuma's inequality the probability that the sum of the at most $|X|$ noise counts for concept c is $\lambda'' \cdot \sqrt{|X|} \cdot \log(1/\beta) \cdot \log(|C|))$ is (for large enough λ'') at most $\beta/2|C|$.

Taking a union bound over all the concepts and taking $n \geq \lambda \cdot (\sqrt{|X|} \cdot \log(|X|) \cdot \log^2(1/\beta) \cdot \log(|C|))/(\varepsilon \cdot \alpha)$, we get that with probability $1 - \beta$ the noisy counts are α accurate on all of the concepts. □

We think of a type (an element) in the universe X as a binary vector, with $x_{ij} = 1$ if $c_j(x_i) = 1$, and $x_{ij} = 0$ otherwise. For each type x_i we can define the corresponding *negative type* $-x_i$ consisting only of 0's and -1's. The result above outputs a collection of counts, one per type in X. The output is not a synthetic database because some of the counts can be negative, but if we allow negative types, then we can create a "pseudo-synthetic" database with the types in X and negative types. Also, using Theorem C.2 of Section C.2, it is possible to modify the above sanitizer and get truly synthetic output, but the running time of the new sanitizer will be polynomial in $|C|$.

Finally, contrast the above positive result to the negative result of Theorem 3.2. That theorem says that sanitizing with respect to an exponential concept class and with synthetic output is hard

when the database size is (roughly) *smaller than* $\sqrt{|X|}$. The negative result holds even for sanitizers with pseudo-synthetic output. The positive result we have just presented, on the other hand, only works when the database size is (roughly) *larger than* $\sqrt{|X|}$.

C.2 From Arbitrary Outputs to Synthetic Data

We begin by observing that sanitizing with arbitrary output and with synthetic datasets are more or less equivalent when the data universe and concept class are both of polynomial size.

Theorem C.2. *Let X be a data universe, C a concept class and a A an (ε, δ)-differentially private sanitizer with utility $(\alpha, \beta, 0)$ and arbitrary output.*

Then there exists a sanitizer A' that is (ε, δ)-differentially private and has utility $(4\alpha, 2\beta, 0)$. The new sanitizer A' outputs a synthetic database of size $\tilde{O}(\log(|C|) \cdot \log(1/\beta)/\alpha^2)$, its running time is polynomial in A's and in $(|X|, |C|, 1/\alpha, \log(1/\beta))$

Proof. The idea is to run the sanitizer A and then use it to get (differentially private) counts on all the concepts in C. We will then use linear programming to come up with a low-weight fractional database that approximates these counts. Finally, we transform this fractional database into a standard synthetic database by rounding the fractional counts.

The new sanitizer A' begins by running A on its input database x, and then for each concept $c \in C$ it uses A's output to compute a fractional count on that concept. The input database x is never accessed again and so A' is (ε, δ)-differentially private. Let v be the resulting vector of counts, i.e. v_c is the fractional count that A's output gives on concept c. With probability $1 - \beta$, all of the entries in v are α-accurate.

We now want to extract a "fractional" database y that approximates these counts. This will be done using linear programming. For every $i \in X$ we introduce a variable $a_i \geq 0$ that will "count" the (fractional) number of occurrences of i in the fractional database y. We represent the count of concept c in y as the sum of the count of items that satisfy c. We want all of these counts to be within an α-accuracy of the respective counts in v_c, writing this as a linear inequality we get:

$$(v_c - \alpha) \cdot \sum_{i \in X} a_i \quad \leq \quad \sum_{i \in X \text{ s.t. } c(i)=1} a_i \leq (v_c + \alpha) \cdot \sum_{i \in X} a_i$$

When the counts are all within an α-accuracy of the counts in v_c, it is also the case that (with probability $1 - \beta$) they are all within a 2α-accuracy of the correct counts.

We write a linear program with two such constraint for each concept (a total of $2|C|$ constraints). A' tries to find a fractional solution to this linear program. To see that such a solution exists, observe that the database x itself is α-close to the vector of counts v, and so there *exists* a solution to the linear program (in fact even an integer solution), and hence A' will find *some* fractional solution.

We conclude that A' can generate a fractional database with $(2\alpha, \beta, 0)$-utility, but we really want a synthetic (integer) database. To transform the fractional database into an integer one, we first scale the fractional database so that its total weight is 1, i.e. $\sum_{i \in X} a_i = 1$. Now round down each fractional point to closest multiple of $\alpha/|X|$, this changes each fractional count by at most a $\alpha/|X|$ additive factor, and so the rounded counts have $(3\alpha, \beta, 0)$ utility. Now we can treat the rounded fractional database (which has total weight 1), as an integer synthetic database of (polynomial) size at most $|X|/\alpha$. If this size is too large, we can reduce it by randomly sampling $m = \tilde{O}(\log(|C|) \cdot \log(1/\beta)/\alpha^2)$ items, as suggested by [BLR08]. With probability $1 - \beta$ the resulting database maintains 4α-utility on all of the concepts. Taking a union bound (over the probability of A failing and the probability of the random sub-sampling failing) we get $(4\alpha, 2\beta, 0)$-utility.

□

C.3 On-Average Utility for Large Concept Classes

In this section we show how to transform a sanitizer with synthetic data that gives worst-case utility for any small set of concepts from a class C into a sanitizer that gives average-case utility for *the entire class* C. This transformation works for any concept class C and data universe X as long as the output of the sanitizer is not too large.

The idea behind the proof is that if we have a sanitizer with synthetic data that works well for every small set of concepts, we can sub-sample a *random* small set of concepts C' from a large concept class C and activate the sanitizer on the small set of concepts. If the sanitizer's output is small enough (w.l.o.g. since we can randomly sub-sample from the output), then with overwhelming probability over the selection of C' any synthetic data output that is accurate for concepts in C' should also be accurate for almost all the concepts in the (much larger) C, and we get on-average utility. Variants of this idea will also prove useful in the subsequent sections.

Theorem C.3. *Let C be a concepts class that can be efficiently sampled and X a data universe, and let A be an (ε, δ)-differentially private sanitizer that, given a subclass $C' \subseteq C$ of up to s concepts, achieves $(\alpha, \beta, 0)$-utility for all the concepts in C'. Furthermore, A works for databases of size n or greater and its output is a synthetic data set of total size m bits.*

Then there is an (ε, δ)-differentially private sanitizer A' that achieves $(\alpha, 2\beta, (m + \log(1/\beta))/s)$ utility for the class C. This new sanitizer A' also works for databases of size n or more, its running time is polynomial in A's, and it also outputs a synthetic data set of total size m bits.

Proof. The sanitizer A' works by choosing *at random* a set C' of s concepts from C and running A on its input database x with the concepts in C', outputting a small m-bit sanitization $A(x)$. This sanitization is A''s output and so it is clearly (ε, δ)-differentially private.

For utility, we claim that with probability $1-\beta$ over the selection of the concept set C' there *does not exist* an m-bit output sanitization y and a collection C_y of concepts of size $|C|/s \cdot (m+2\log(1/\beta))$ such that y is α-accurate for *all* of the concepts in C', but bad for all the concepts in C_y. Once we prove this, we know that with high probability over C', whenever $A(x)$ is accurate for all of the concepts in C', it is also accurate for all but $|C|/s \cdot (m + 2\log(1/\beta))$ of the concepts in C. Taking a union bound, A''s total failure probability is at most 2β.

Examine a potential m-bit output y of A, and say that y is "bad" if it gives counts that are not α-accurate for a concept set C_y of total size $|C| \cdot (\log(1/\beta) + m)/s$ (within C). For such a bad y, what is the probability (over C') that y gives accurate counts for *every* concept in C'? This is exactly the probability that none of the concepts in C' are in C_y, or

$$(1 - |C_y|/|C|)^{|C'|} \leq (1 - (\log(1/\beta) + m)/s)^s \leq e^{-(\log(1/\beta)+m)} \leq \beta \cdot 2^{-m}$$

Now, taking a union bound, the probability that there *exists* an m-bit output y that is accurate on the concepts in C' but inaccurate on a set of size $|C| \cdot (\log(1/\beta) + m)/s$ is at most β.

□

Remark C.1. *If, instead of producing a synthetic data set, the sanitizer A used in Theorem C.3 produced an arbitrary data structure that accurately covered $\mathcal{C}'$, we would not know how to argue that it also gives information about $\mathcal{C} \setminus \mathcal{C}'$. This is all we need, but we do not in general know how to interpret an arbitrary data structure for $\mathcal{C}'$ on concepts not in $\mathcal{C}'$. In contrast, an item from the data universe always either satisifes or fails to satisfy an arbitrary concept in $\mathcal{C}$.*

We note that the condition in Theorem C.3 on the smallness of the sanitizer's output is especially reasonable for sanitizers with synthetic output. The reason is that we can always "shrink" the

output length of such a sanitizer by randomly sub-sampling a few of its output items. This comes at only a small cost in utility: for any sanitizer A with synthetic output and (α, β, γ)-utility, we can randomly sub-sample $\tilde{O}(\log(1/\beta) \cdot \log(|C|)/\alpha^2)$ items, to get an m-bit output where $m = \tilde{O}(\log(|X|) \cdot \log(1/\beta) \cdot \log(|C|)/\alpha^2)$. With probability $1 - \beta$ the counts of the small subsampled synthetic database are α-close to those of A's output, and are thus 2α-accurate for all of the concepts A was accurate on. Thus we get a sanitizer with small output and $(2\alpha, 2\beta, \gamma)$-utility and small output (there is no loss in privacy).

Average-Case Utility with Polynomial Size X. When X is of polynomial size, by Theorem C.2 we can transform any sanitizer A on small concept classes into one that outputs small synthetic databases. This means that if we permit sanitizers to run in time $\text{poly}(|X|)$, *any* worst-case sanitizer for small sets of concepts (with or without synthetic data) can be transformed into an average-case sanitizer for larger sets of concepts. Even for an exponential-size concept class C, we can choose a polynomial size collection of random concepts C', efficiently sanitize w.r.t. C' (with a small synthetic data output) and get utility for all but a small polynomial fraction of the concepts in the exponential class C. Using this idea, the size of the input database needs to be polynomial in the size of the class C'.

C.4 An Efficient Sanitizer for Polynomial Concept Classes

In this section we construct an efficient sanitizer for any polynomial size concept class C and data universe X. The advantage of this sanitizer is that it works even for very small databases: databases whose size is roughly $|C|^{o(1)} \cdot \log(|X|)$. We begin by stating the parameters of this sanitizer in Theorem 4.2, we outline the construction, prove two useful lemmas and finally prove Theorem 4.2.

Construction Overview. The construction is recursive: we start out wanting to construct a sanitizer for a large concept class of size $|C|$. We reduce this to the goal of constructing a sanitizer for any subset $C' \subseteq C$ of size $|C|/f$ where f is some factor by which we shrink the database. The reduction is stated as Lemma C.4 below, and the intuition follows the proof of Theorem C.3. Say we have a sanitizer for sets of $|C|/f$ concepts with small synthetic output: we sample a set of $|C|/f$ concepts uniformly from C and run this sanitizer on them. With overwhelming probability, the resulting synthetic database y gives accurate counts for all but a small set of concepts in C. We release (in a privacy-preserving manner) the names of the concepts for which y does not give accurate counts, and we also release these concepts' counts in the input database x (adding noise to make sure these counts also preserve privacy).

The sanitizer for sets of $|C|/f$ concepts used in the above reduction is constructed recursively via the same construction. The output size for the sanitizer constructed above is large (and it is not synthetic data), but we can reduce it to be a small synthetic output using Theorem C.2. We do this recursively d times until in the end we only need to construct a sanitizer on sets of $|C|/f^d$ concepts. In each of these recursive steps we pay additive losses in the privacy parameters and in β and a constant multiplicative factor in the α accuracy parameter (a factor of 4 comes from using Theorem C.2 to generate synthetic data, in the actual construction there will be additional losses in accuracy). The recursion terminates once the concept class size is small enough to allow us to sanitize using a very simple sanitizer.

To get a better (but rough) idea of the parameters, at the "bottom" of the recursion we need a sanitizer for databases of size roughly $|C|/f^d$ with accuracy smaller than $1/8^d$. We construct such

a sanitizer in Claim C.6 by simply outputting noisy counts for all the concepts. To get this small sanitizer we need the input database x to be of size n greater than $|C|/f^d + 8^d$. In each of the recursive steps we are also adding noise to sets of counts of size roughly f, so in total we need the input database size to be larger than $f + |C|/f^d + 8^d$. To optimize this size we choose $f = 2^{\sqrt{\log(|C|)}}$ and $d = \sqrt{\log(|C|)}$, which requires databases of size at least $2^{O(\sqrt{\log(|C|)})}$. We proceed to prove this result.

Lemma C.4. *Let C be a concept class and X a data universe. Let A_{small} be an $(\varepsilon_{small}, \delta_{small})$-differentially private sanitizer that, given any concept set $C_{small} \subseteq C$ of size s_{small}, achieves (α, β)-utility on the concepts in C_{small}. Say A_{small} works on databases of size at least n_{small} and has synthetic output of size m bits (these can both be a function of the other parameters).*

For any desired ε and parameter $k \geq |C|$, where $(\beta + neg(k)) \leq min(1/2, \varepsilon/8)$, we construct a sanitizer A for the entire class C. This sanitizer A is $(\varepsilon_{small} + \varepsilon, \delta_{small} + \beta + neg(k))$-differentially private and has $(2\alpha, neg(k))$-utility. It works for databases of size n where:

$$n \geq \lambda \cdot \max(n_{small}, (|C|/s_{small}) \cdot (\log^4(k) \cdot m)/(\varepsilon \cdot \alpha))$$

For some universal constant $\lambda > 0$. The running time of the new sanitizer is $\text{poly}(k, 1/\alpha, 1/\varepsilon)$ *plus a single call to A_{small}.*

Proof. We choose a random subset C_{small} of s_{small} concepts in C and run the sanitizer A_{small} on this set with parameters $\varepsilon_{small}, \delta_{small}, \alpha, \beta$ (we need an input database of size n_{small}). Let y be the m-bit output of A_{small}.

As in Theorem C.3, with overwhelming probability $1 - 2^{\log^2 k}$ over the choice of C_{small}, the set C_{bad} of concepts that y is not α-accurate on is of size smaller than

$$s_{bad} = |C| \cdot (\log^2(k) + m)/s_{small}$$

The sanitizer A outputs this (differentially private) y in the clear. There remains, however, the set C_{bad} of concepts for which y is not accurate. We would like to output which of the concepts are in C_{bad} and then separately release a differentially private sanitization that has good utility on this (much smaller) set of concepts. The problem is that the set of concepts that are in C_{bad} depends on the input database x, and so we must be careful to release it in a differentially private manner.

For each concept $c \in C$ we want to release a bit v_c that indicates whether y approximates it well: $v_c = 0$ indicates that y approximates the concept fairly well and $v_c = 1$ indicates that it does not. Our goal is to release a differentially private vector v with all the indicator bits for all the functions.

Indicator Vector. To release the indicator bit vector v, we use the exponential mechanism, where the distance function $d_{c,y}$ for a concept c and an output y (of A_{small}) is:

$$d_{c,y}(x, 0) = |c(x) - c(y)| - 1.5\alpha$$

$$d_{c,y}(x, 1) = 1.5\alpha - |c(x) - c(y)|$$

This function has sensitivity $1/n$ (recall $c(x)$ and $c(y)$ are fractional counts), it can alternatively be written as $d_{c,y}(x, b) = -1^b \cdot (|c(x) - c(y)| - 1.5\alpha)$. Now using the exponential mechanism, for each concept c we output bit b with probability proportional to

$$Pr[v_c = b] \propto e^{-\varepsilon \cdot n \cdot d_{c,y}(x,b)/2s_{bad}}$$

The intuition is that if the difference between c's counts on x and y is smaller than 1.5α, then 0 becomes more likely (exponentially fast in the distance), whereas if the difference is larger than 1.5α, then 1 becomes more likely. Note that, since the sensitivity of the distance function $d_{c,y}$ is at most $1/n$, each of the bits v_c is (on its own) an $\varepsilon/2s_{bad}$-differentially private function of the input (for any possible output y and concept c). We first claim that the bit v_c indeed provides a good indication of whether the output y accurately approximates the concept c's count.

Claim C.5. *For every concept c and output y and for every database x of size $n \geq 8/\alpha$:*

- *If the difference between x and y's counts on c is at most $\alpha + 2/n$ then v_c is 1 with probability at most $e^{-\varepsilon\alpha n/8s_{bad}}$.*
- *If the difference between x and y's counts on c is at least $2\alpha - 2/n$ then v_c is 0 with probability at most $e^{-\varepsilon\alpha n/8s_{bad}}$.*

The above claim shows that for concepts c that y either "fits" very well (about α-close to their counts in x), or for those that y fits very poorly (about 2α-far), their indicator bit v_c is with overwhelming probability either 0 or 1. Moreover, this is even true for the given y even when we compute the v_c bits using a neighboring database x' of x! This fact will be useful for analyzing privacy, as essentially the only indicator bits we need to "worry about" when moving form x to x' are the ones whose counts in y are not very accurate. By the smallness of s_{bad} these concepts are relatively few.

The sanitizer A releases the vector v indicating which concepts y is accurate for. Finally, for each of the concepts c for which $v_c = 1$, it computes c's count in the input x, adds noise by the distribution $Lap(4s_{bad}/\varepsilon)$, and releases the noisy count.

In summary, A's output includes the synthetic database y, the vector v indicating which concepts y "fits", and a collection of noisy counts for all the concepts that y does not fit. We now argue utility and differential privacy.

Utility. For any concept c for which y is not 2α-accurate, the probability that $v_c = 0$ is at most $e^{-\varepsilon\alpha n/8s_{bad}}$ (Claim C.5). Taking $n \geq \log^2(k) \cdot s_{bad}/(\varepsilon \cdot \alpha)$ we get that this probability is negligible in k. We conclude that with all but $neg(k)$ probability (over the sanitizer's coins), for the concepts for which $v_c = 0$, the output y indeed gives a 2α-accurate count. For the concepts where $v_c = 1$ we output the correct count with noise $Lap(4s_{bad}/\varepsilon)$. The probability that the noise is greater than $\log^2(k) \cdot s_{bad}/\varepsilon$ is negligible in k, and so taking $n \geq \log^2(k) \cdot s_{bad}/(\varepsilon \cdot \alpha)$ we also get α-accuracy for all of these concepts with all but $neg(k)$ probability.

Differential Privacy. The output y is differentially private by assumption on A_{small}. The main question we have to answer is whether the vector v we release is differentially private. If it is, then since with overwhelming probability only at most s_{bad} concepts have $v_c = 1$ (by Claim C.5), and for each concept with $v_c = 1$ we release an $\varepsilon/2s_{bad}$-differentially private noisy count. Releasing the s_{bad} noisy counts is thus $\varepsilon/4$-differentially private.

Releasing v may, at first glance, seem problematic: there are $|C|$ concepts and when computing v_c for each such concept, we are only adding "noise" on the magnitude of $s_{bad}/2\varepsilon$ (rather than $|C|/\varepsilon$). A crucial observation, however, is that almost all the concepts v_c will very "strongly" tend to be 0 both for the input database x and for any neighboring database x'. Thus (since we are only guaranteeing (ε, δ)-differential privacy), at least intuitively, we do not need to worry too much

about the concepts that are not in C_{bad}, and it is sufficient to add noise proportional to s_{bad}, the number of concepts that y does not fit well.

More formally, for any two neighboring databases x and x', examine the probability of any output event E consisting of a set of possible (y, v) output pairs (we ignore for now the noisy counts that are released for c's with $v_c = 1$, adding them to the analysis later). For now, assume that in both cases (for x and for x') we look at the event E when the synthetic database y is generated according to x. We will later use the differential privacy of A_{small} and the composition of differential privacy (see [DL08, Mir08]) to argue that generating y according to x or x' will not change the probabilities of events too much.

We use $P[Z]$ to denote the probability of an event Z when running the sanitizer A on input database x, and $P'[Z]$ to denote the probability when running the sanitizer A on the neighboring database x'. Recall that, for now, in both these cases we assume y is generated by running A_{small} on x. Consider the following two events:

- The event Y: the output vector y is *not* $\alpha + 1/n$-*accurate* w.r.t. x for a set of s_{bad} concepts.
- The event V: some concept c for which y is $\alpha + 1/n$-accurate w.r.t. x has $v_c \neq 0$.

By the utility of A_{small} and by Claim C.5 the probability of either of these events happening is at most $\beta + neg(k)$ both when running the sanitizer on input database x and on its neighbor x'.

$$\begin{aligned} P[E] &\leq P[Y \cup V] \cdot P[E|Y \cup V] + P[\bar{Y} \cap \bar{V}] \cdot P[E|\bar{Y} \cap \bar{V}] \\ &\leq \beta + neg(k) + P[E|\bar{Y} \cap \bar{V}] \end{aligned}$$

Let us now examine the relationship between $P[E|\bar{Y} \cap \bar{V}]$ and $P'[E|\bar{Y} \cap \bar{V}]$. These are both the probabilities of E conditioned on the events Y and V both not occurring. This means that for every distinct y, we can restrict our attention to a small set of s_{bad} locations in v, and it is given that v is 0 everywhere else. This set of locations is fixed per y, corresponding to the few concepts for which y and x do not strongly agree. We call this set $L(y)$:

$$L(y) = \{c \in C \text{ s.t. } |c(y) - c(x)| > \alpha + 2/n\}$$

Now we treat the event $(E|\bar{Y} \cap \bar{V})$ as a set of possible pairs $(y, v_{|L(y)})$. There is no need to consider locations in v outside of $L(y)$, as we have already conditioned on them all being 0.

For a given y and a given set of bit values $v_{L(y)}$, because we computed each location in v using a $\varepsilon/2s_{bad}$-differentially private function, and the size of $L(y)$ is at most s_{bad}, we get (because differential privacy composes) that:

$$P[(y, v_{|L(y)}|\bar{Y} \cap \bar{V}] \leq e^{\varepsilon/2} \cdot P'[(y, v_{|L(y)}|\bar{Y} \cap \bar{V}]$$

We conclude that

$$P[E|\bar{Y} \cap \bar{V}] \leq e^{\varepsilon/2} \cdot P'[E|\bar{Y} \cap \bar{V}]$$

And so

$$\begin{aligned} P[E] &\leq P[E|\bar{Y} \cap \bar{V} + \beta + neg(k) \\ &\leq e^{\varepsilon/2} \cdot P'[E|\bar{Y} \cap \bar{V}] + \beta + neg(k) \\ &\leq e^{\varepsilon/2} \cdot P'[E]/P'[\bar{Y} \cap \bar{V}] + \beta + neg(k) \\ &\leq (e^{\varepsilon/2}/(1 - \beta - neg(k))) \cdot P'[E] + \beta + neg(k) \\ &\leq e^{3\varepsilon/4} \cdot P'[E] + \beta + neg(k) \end{aligned}$$

Where the last inequality is because we assumed $(\beta + neg(k)) \leq min(1/2, \varepsilon/8)$, and for $x \in (0, 1/2]$ we know that $\ln(1-x) \geq -2x$.

In the case that Y and V do not occur, we also release noisy counts for the concepts for which $v_c = 1$. This is an $(\varepsilon/4)$-differentially private function of the input database, and by a similar argument we get $(\varepsilon, \beta + neg(k))$-differential privacy for the output of the sanitizer (y, v and the counts). Finally, throughout the analysis we assumed that y was generated using x. Now since y itself is an $(\varepsilon_{small}, \delta_{small})$ differentially private function of the input database, by composition of differential privacy (see [DL08, Mir08]), the output of the sanitizer A is $(\varepsilon + \varepsilon_{small}, \beta + neg(k) + \delta_{small})$-differentially private.

□

We now outline the (simple) sanitizer for small concept classes that we will use as the basis of our recursive construction.

Claim C.6. *Let C be a concept class and X a data universe. For any desired $\alpha, \varepsilon > 0$ and parameter $k \geq |C|$, there is an ε-differentially private sanitizer A with $(\alpha, neg(k))$-utility for databases of size n, provided that $n \geq 4\log^2(k) \cdot |C|/(\varepsilon \cdot \alpha)$. The sanitizer A runs in time* $\text{poly}(|C|, |X|, k, 1/\varepsilon, 1/\alpha)$ *and outputs synthetic data.*

Proof. The sanitizer A first computes for every concept c its count according to the input database x, and then adds noise from $Lap(|C|/\varepsilon)$. These counts are each $\varepsilon/|C|$-differentially private and so the collection of $|C|$ counts is ε-differentially private. For $n \geq 4\log^2(k) \cdot |C|/(\varepsilon \cdot \alpha)$, with all but $neg(k)$ probability they are all $\alpha/4$-accurate. Finally, we use the transformation of Theorem C.2 to transform this output into a (small) synthetic database.

□

We are now ready to put together Lemma C.4 and Claim C.6 to construct a sanitizer for polynomial concept classes and data universes and prove Theorem 4.2.

Proof of Theorem 4.2. The construction is recursive, using Lemma C.4 to reduce the size of the concept class until it reaches a concept class small enough to use the simple sanitizer of Claim C.6. We will construct a sequence of sanitizers on larger and larger concept sets. As outlined above, we recurse d times, each time reducing the size of the concept class by a factor of f. We specify d and f below.

We start with the "base" sanitizer A_1, of Claim C.6. Take $\alpha_1 = \alpha/8^d$ and $\varepsilon_1 = \varepsilon/d$. We want the sanitizer A_1 to have $(\varepsilon_1, neg(k))$-differential privacy and $(\alpha_1, neg(k))$-utility for concept classes of constant size (say 1). This is guaranteed by Claim C.6 as long as the input database size n is at least

$$n_1 = 4\log^2(k)/(\varepsilon_1 \cdot \alpha_1) = (4\log^2(k) \cdot 8^d \cdot d)/(\varepsilon \cdot \alpha)$$

Let i be an integer in $\{1, \ldots, d\}$. Take $\alpha_i = 8^i \cdot \alpha_1$ and $\varepsilon_i = i \cdot \varepsilon_1$. Suppose we have a sanitizer A_i with $(\alpha_i, neg(k))$-utility and $(\varepsilon_i, neg(k))$-differential privacy for concept classes of size $s_i = f^d$ and databases of size n_i specified below.

First, we use Theorem C.2 to transform A_i into a sanitizer with synthetic output of length $m_i = \tilde{O}((1/\alpha_i^2) \cdot \log^2(k) \cdot \log(|C|) \cdot \log(|X|))$. This new sanitizer has $(4\alpha_i, neg(k))$-utility.

Now, by Lemma C.4, we can use the "smaller" sanitizer A_i to construct a sanitizer A_{i+1} for larger concept classes of size $s_{i+1} = f^{i+1}$. By Lemma C.4, this new sanitizer has $(8\alpha_i, neg(k))$-utility and $(\varepsilon_i + \varepsilon_1, neg(k))$-differential privacy. It works for databases of size n_{i+1}, where:

$$n_{i+1} \geq \max(n_i, \lambda \cdot (f \cdot \log^7(k) \cdot \log(|X|) \cdot \log(1/\alpha_i))/(\alpha_i^3 \cdot \varepsilon_1))$$

for a universal constant $\lambda > 0$, and its running time is as in the theorem guarantee.

Starting with our initial sanitizer A_1, we get a final sanitizer A_d with utility $(\alpha = 8^d \cdot \alpha_1, neg(k))$ and $(\varepsilon = d \cdot \varepsilon_1, neg(k))$-differential privacy.

By our choice of parameters, we get that the final sanitizer works for concept classes of size f^d as long as the database size is at least:

$$n \quad = \quad \max((4\log^2(k) \cdot 8^d \cdot d)/(\varepsilon \cdot \alpha), \lambda \cdot (f \cdot \log^7(k) \cdot \log(|X|) \cdot d^2 \cdot 8^{3d} \cdot \log(1/\alpha))/(\alpha^3 \cdot \varepsilon)))$$

for some universal constant $\lambda > 0$. So it suffices to take:

$$n = \lambda \cdot 8^{3d} \cdot f \cdot d^2 \cdot (log^7 k \cdot \log(|X|) \cdot \log(1/\alpha))/(\alpha^2 \cdot \varepsilon)$$

Taking $d = \sqrt{\log(|C|)}/3$ and $f = 2^{3\sqrt{\log(|C|)}}$, we get a sanitizer for concept classes of size $|C|$ that works as long as the database is of size:

$$n = \lambda \cdot 2^{6\sqrt{\log(|C|)}} \cdot (\log^8(k) \cdot \log(|X|) \cdot \log(1/\alpha))/(\alpha^3 \cdot \varepsilon)$$

□

C.5 Very Small Databases

The result of Section C.4 cannot be used to sanitize databases of tiny (say polylog($|C|$)) size. In this section we construct an efficient sanitizer for tiny databases of size roughly $O(\log |X|)$ (for constant $\alpha, \beta, \gamma, \varepsilon$), with average-case utility. The sanitizer's running time is polynomial in $\log(|C|)$ (i.e. it is efficient even for exponential-size C), but while it is efficient for fixed constant α, γ, its running time grows polynomially in $|X|^{(\alpha, \log(1/\gamma))}$. This sanitizer uses the exponential mechanism and the (inefficient) sanitizer of [BLR08], relaxing to average-case utility enables us to reduce the running time.

Theorem C.7. *Let C be a concept class that can be efficiently sampled and X a data universe. For any desired $\alpha, \beta, \gamma, \varepsilon \in (0, 1)$, there is an there is an ε-differentially private sanitizer A with (α, β, γ)-utility that runs on databases of size n such that*

$$n \geq \lambda \cdot \big((\log(|X|) \cdot \log(1/\alpha) \cdot \log(1/\gamma))/(\alpha^3 \cdot \varepsilon) + (log(1/\beta)/(\alpha \cdot \varepsilon))\big)$$

for a universal constant $\lambda > 0$. A's runtime is $\text{poly}(\log(|C|), |X|^{\log(1/\alpha) \cdot \log(1/\gamma)/\alpha^2}, n, \log(1/\beta), 1/\varepsilon)$.

Proof. Recall the sanitizer of [BLR08]: they output each possible synthetic database y of small (but super-constant) size with probability exponentially small in its distance from the input x. The range of outputs they consider is thus of super-polynomial size (at least $|X|^{\log(|C|)}$). The distance measure they use is the worst case (over all concepts in C) difference in counts between the two databases, and so computing this distance may require enumerating all the concepts in C (whereas we want an efficient algorithm even for exponential size C). The [BLR08] sanitizer is thus inefficient for two reasons: (i) it runs the exponential mechanism on a super-polynomial sized range of outputs, (ii) computing the probability of a given item in the range may be hard. We show that if we relax the utility guarantee to average-case (over concepts) utility, then both of these obstacles can be overcome by .

To overcome the first difficulty, observe that for any database x and concept c, sub-sampling $m = O(\log(1/\alpha) \cdot \log(1/\gamma)/\alpha^2)$ of the items of x maintains $\alpha/2$-accuracy on the concept c with probability $1 - \gamma/4$. Thus there exists an m-item synthetic database that is $\alpha/2$-accurate on $1 - \gamma/4$

of the concepts. Our sanitizer will run the exponential mechanism as in [BLR08], but only on the collection of m-item synthetic databases. There are only $|X|^m = \text{poly}(|X|^{\log(1/\alpha)\cdot\log(1/\gamma)/\alpha^2})$ such databases (a polynomial number for constant α, γ), and this factors into the algorithm's running time.

To overcome the second difficulty, we change the distance function used when we invoke the exponential mechanism. All we want from our output is that it be accurate on a $(1-\gamma)$-fraction of the concepts. In the exponential mechanism, instead of using the distance of the worst-case (over all concepts) concept, we would like to use the distance of the $(|C| \cdot \gamma)$-th most distant (again over all the concepts) concept. This distance, unfortunately, is still not easy to compute. Instead of computing it precisely, the new sanitizer samples a set of S of $\text{poly}(m \cdot \log|X| \cdot \log(1/\beta)/\gamma)$ concepts from C and its distance function $d_{S,\gamma}$ will output the distance of the $(|S| \cdot \gamma/2)$-th most distance concept within this set S.

This final distance function is efficiently computable. Furthermore, its sensitivity is at most $1/n$, and so (as in [BLR08]) ε-differential privacy is guaranteed by the exponential mechanism. It remains to argue utility.

First, we claim that for any input x, with probability $1 - \beta/4$ over the choice of S, there exists an m-item synthetic database y such that $d_{S,\gamma}(x, y) \leq \alpha/2$. This is because we know there exists such a y that is $\alpha/2$-accurate on a $(1 - \gamma/4)$-fraction of the concepts in C, and so by a Chernoff bound the probability (over S) that it is not $\alpha/2$ accurate on a $(1 - \gamma/2)$-fraction of the concepts in S is at most $\beta/4$.

Second, for an arbitrary "bad" m-item synthetic database y that is at least α-inaccurate on at least a γ-fraction of the concepts in C, the probability that it is α-accurate on a $(1 - \gamma/2)$-fraction of the concepts in S is (again by a Chernoff Bound) at most $\beta/(4 \cdot |X|^m)$. Taking a union bound over the $|X|^m$ possible m-item synthetic databases, the probability that there *exists* a such "bad" y such that $d_{S,\gamma}(x, y) \leq \alpha$ is negligible.

Putting these two facts together, we get that with probability $1 - \beta/2$ there exists a "good" y with $d_{S,\gamma}(x, y) \leq \alpha/2$ and for *every* "bad" y it is the case that $d_{S,\gamma}(x, y) \geq \alpha$. By the analysis of [BLR08] this means that the exponential mechanism will output a y that is not "bad", i.e. has α-accuracy on all but a γ-fraction of the concepts, with probability at least $1 - \beta$.

□