all of the data points approximately as well as the best function in $\mathcal{Q}$, even if no function in $\mathcal{Q}$ can perfectly label them. Note that equivalently, an agnostic learning algorithm is one that maximizes the number of positive examples labeled 1 minus the number of negative examples labeled 1. Phrased in this way, we can see that a *distinguisher* as defined above is just an agnostic learning algorithm: just imagine that x contains all of the "positive" examples, and that y contains all of the "negative examples." (Note that it is ok of x and y are not disjoint — in the learning problem, the same example can occur with both a positive and a negative label, since agnostic learning does not require that any function perfectly label every example.) Finally, note also that for classes of linear queries $\mathcal{Q}$, a distinguisher is simply an optimization algorithm. Because for linear queries f, $f(x) - f(y) = f(x - y)$, a distinguisher simply seeks to find $\arg\max_{f \in \mathcal{Q}} |f(x - y)|$.

Note that, *a priori*, a differentially private distinguisher is a weaker object than a differentially private release algorithm: A distinguisher merely finds a query in a set $\mathcal{Q}$ with the approximately largest value, whereas a release algorithm must find the answer to every query in $\mathcal{Q}$. In the algorithm that follows, however, we reduce release to optimization.

We will first analyze the IC algorithm, and then instantiate it with a specific distinguisher and database update algorithm. What follows is a formal analysis, but the intuition for the mechanism is simple: we simply run the iterative database construction algorithm to construct a hypothesis that approximately matches x with respect to the queries $\mathcal{Q}$. If at each round our distinguisher succeeds in finding a query that has high discrepancy between the hypothesis database and the true database, then our database update algorithm will output a database that is β-accurate with respect to $\mathcal{Q}$. If the distinguisher ever fails to find such a query, then it must be that there are no such queries, and our database update algorithm has already learned an accurate hypothesis with respect to the queries of interest! This requires at most T iterations, and so we access the data only $2T$ times using $(\varepsilon_0, 0)$-differentially private methods (running the given distinguisher, and then checking its answer with the Laplace mechanism). Privacy will therefore follow from our composition theorems.

Algorithm 8 The Iterative Construction (IC) Mechanism. It takes as input a parameter ε_0, an $(F(\varepsilon_0), \gamma)$-Private Distinguisher Distinguish for $\mathcal{Q}$, together with an $T(\alpha)$-iterative database update algorithm U for $\mathcal{Q}$.

IC$(x, \alpha, \varepsilon_0, \text{Distinguish}, U)$:
  Let $D^0 = U(\perp, \cdot, \cdot)$.
  for $t = 1$ to $T(\alpha/2)$ **do**
    Let $f^{(t)} = \text{Distinguish}(x, D^{t-1})$
    Let $\hat{v}^{(t)} = f^{(t)}(x) + \text{Lap}\left(\frac{1}{\|x\|_1 \varepsilon_0}\right)$.
    if $|\hat{v}^{(t)} - f^{(t)}(D^{t-1})| < 3\alpha/4$ **then**
      Output $y = D^{t-1}$.
    else
      Let $D^t = U(D^{t-1}, f^{(t)}, \hat{v}^{(t)})$.
    end if
  end for
  Output $y = D^{T(\alpha/2)}$.

The analysis of this algorithm just involves checking the technical details of a simple intuition. Privacy will follow because the algorithm is just the composition of $2T(\alpha)$ steps, each of which is $(\varepsilon_0, 0)$-differentially private. Accuracy follows because we are always outputting the last database in a maximal database update sequence. If the algorithm has not yet formed a maximal Database Update Sequence, then the distinguishing algorithm will find a distinguishing query to add another step to the sequence.

Theorem 5.3. The IC algorithm is $(\varepsilon, 0)$-differentially private for $\varepsilon_0 \leq \varepsilon / 2T(\alpha/2)$. The IC algorithm is (ε, δ)-differentially private for $\varepsilon_0 \leq \frac{\varepsilon}{4\sqrt{T(\alpha/2)\log(1/\delta)}}$.

Proof. The algorithm runs at most $2T(\alpha/2)$ compositions of ε_0-differentially private algorithms. Recall from Theorem 3.20 that ε_0 differentially private algorithms are $2k\varepsilon_0$ differentially private under $2k$-fold composition, and are (ε', δ) private for $\varepsilon' = \sqrt{4k\ln(1/\delta')}\varepsilon_0 + 2k\varepsilon_0(e^{\varepsilon_0} - 1)$. Plugging in the stated values for ε_0 proves the claim. □

Theorem 5.4. Given an $(F(\varepsilon), \gamma)$-private distinguisher, a parameter ε_0, and a $T(\alpha)$-Database Update Algorithm, with probability at least $1-\beta$, the IC algorithm returns a database y such that: $\max_{f \in \mathcal{Q}} |f(x)-f(y)| \leq \alpha$ for any α such that where:

$$\alpha \geq \max\left[\frac{8\log(2T(\alpha/2)/\beta)}{\varepsilon_0\|x\|_1}, 8F\left(\varepsilon_0\right)\right]$$

so long as $\gamma \leq \beta/(2T(\alpha/2))$.

Proof. The analysis is straightforward.

Recall that if $Y_i \sim \mathrm{Lap}(1/(\varepsilon\|x\|_1))$, we have: $\Pr[|Y_i| \geq t/(\varepsilon\|x\|_1)] = \exp(-t)$. By a union bound, if $Y_1, \ldots, Y_k \sim \mathrm{Lap}(1/(\varepsilon\|x\|_1))$, then $\Pr[\max_i |Y_i| \geq t/(\varepsilon\|x\|_1)] \leq k\exp(-t)$. Therefore, because we make at most $T(\alpha/2)$ draws from $\mathrm{Lap}(1/(\varepsilon_0\|x\|_1))$, except with probability at most $\beta/2$, for all t:

$$|\hat{v}^{(t)} - f^{(t)}(x)| \leq \frac{1}{\varepsilon_0\|x\|_1}\log\frac{2T(\alpha/2)}{\beta} \leq \frac{\alpha}{8}.$$

Note that by assumption, $\gamma \leq \beta/(2T(\alpha/2))$, so we also have that except with probability $\beta/2$:

$$\begin{aligned}|f^{(t)}(x) - f^{(t)}(D^{t-1})| &\geq \max_{f\in\mathcal{Q}} |f(x) - f(D^{t-1})| - F(\varepsilon_0)\\ &\geq \max_{f\in\mathcal{Q}} |f(x) - f(D^{t-1})| - \frac{\alpha}{8}.\end{aligned}$$

For the rest of the argument, we will condition on both of these events occurring, which is the case except with probability β.

There are two cases. Either a data structure $D' = D^{T(\alpha/2)}$ is output, or data structure $D' = D^t$ for $t < T(\alpha/2)$ is output. First, suppose $D' = D^{T(\alpha/2)}$. Since for all $t < T(\alpha/2)$ it must have been the case that $|\hat{v}^{(t)} - f^{(t)}(D^{t-1})| \geq 3\alpha/4$ and by our conditioning, $|\hat{v}^{(t)} - f^{(t)}(x)| \leq \frac{\alpha}{8}$, we know for all t: $|f^{(t)}(x) - f^{(t)}(D^{t-1})| \geq \alpha/2$. Therefore, the sequence $(D^t, f^{(t)}, \hat{v}^{(t)})$, formed a maximal $(U, x, \mathcal{Q}, \alpha/2, T(\alpha/2))$-Database Update Sequence (recall Definition 5.3), and we have that $\max_{f\in\mathcal{Q}} |f(x) - f(x')| \leq \alpha/2$ as desired.

Next, suppose $D' = D^{t-1}$ for $t < T(\alpha/2)$. Then it must have been the case that for t, $|\hat{v}^{(t)} - f^{(t)}(D^{t-1})| < 3\alpha/4$. By our conditioning, in

this case it must be that $|f^{(t)}(x) - f^{(t)}(D^{t-1})| < \frac{7\alpha}{8}$, and that therefore by the properties of an $(F(\varepsilon_0), \gamma)$-distinguisher:

$$\max_{f \in \mathcal{Q}} |f(x) - f(D')| < \frac{7\alpha}{8} + F(\varepsilon_0) \leq \alpha$$

as desired. □

Note that we can use the exponential mechanism as a private distinguisher: take the domain to be $\mathcal{Q}$, and let the quality score be: $q(D, f) = |f(D) - f(D^t)|$, which has sensitivity $1/\|x\|_1$. Applying the exponential mechanism utility theorem, we get:

Theorem 5.5. The exponential mechanism is an $(F(\varepsilon), \gamma)$ distinguisher for:

$$F(\varepsilon) = \frac{2}{\|x\|_1 \varepsilon} \left(\log \frac{|\mathcal{Q}|}{\gamma} \right).$$

Therefore, using the exponential mechanism as a distinguisher, Theorem 5.4 gives:

Theorem 5.6. Given a $T(\alpha)$-Database Update Algorithm and a parameter ε_0 together with the exponential mechanism distinguisher, with probability at least $1 - \beta$, the IC algorithm returns a database y such that: $\max_{f \in \mathcal{Q}} |f(x) - f(y)| \leq \alpha$ where:

$$\alpha \leq \max \left[\frac{8 \log(2T(\alpha/2)/\beta)}{\varepsilon_0 \|x\|_1}, \frac{16}{\|x\|_1 \varepsilon_0} \left(\log \frac{|\mathcal{Q}|}{\gamma} \right) \right]$$

so long as $\gamma \leq \beta/(2T(\alpha/2))$.

Plugging in our values of ε_0:

Theorem 5.7. Given a $T(\alpha)$-Database Update Algorithm, together with the exponential mechanism distinguisher, the IC mechanism is ε-differentially private and with probability at least $1 - \beta$, the IC algorithm returns a database y such that: $\max_{f \in \mathcal{Q}} |f(x) - f(y)| \leq \alpha$ where:

$$\alpha \leq \frac{8T(\alpha/2)}{\|x\|_1 \varepsilon} \left(\log \frac{|\mathcal{Q}|}{\gamma} \right)$$

and (ε, δ)-differentially private for:

$$\alpha \leq \frac{16\sqrt{T(\alpha/2)\log(1/\delta)}}{\|x\|_1 \varepsilon}\left(\log\frac{|\mathcal{Q}|}{\gamma}\right)$$

so long as $\gamma \leq \beta/(2T(\alpha/2))$.

Note that in the language of this section, what we proved in Theorem 4.10 was exactly that the multiplicative weights algorithm is a $T(\alpha)$-Database Update Algorithm for $T(\alpha) = \frac{4\log|\mathcal{X}|}{\alpha^2}$. Plugging this bound into Theorem 5.7 recovers the bound we got for the online multiplicative weights algorithm. Note that now, however, we can plug in other database update algorithms as well.

5.2.1 Applications: other database update algorithms

Here we give several other database update algorithms. The first works directly from α-nets, and therefore can get non-trivial bounds even for nonlinear queries (unlike multiplicative weights, which only works for linear queries). The second is another database update algorithm for linear queries, but with bounds incomparable to multiplicative weights. (In general, it will yield improved bounds when the dataset has size close to the size of the data universe, whereas multiplicative weights will give better bounds when the dataset is much smaller than the data universe.)

We first discuss the median mechanism, which takes advantage of α-nets. The median mechanism does not operate on databases, but instead on median data structures:

Definition 5.6 (Median Data Structure)**.** A median data structure $\mathbf{D}$ is a collection of databases: $\mathbf{D} \subset \mathbb{N}^{|\mathcal{X}|}$. Any query f can be evaluated on a median data structure as follows: $f(\mathbf{D}) = \text{Median}(\{f(x) : x \in \mathbf{D}\})$.

In words, a median data structure is just a set of databases. To evaluate a query on it, we just evaluate the query on every database in the set, and then return the median value. Note that the answers given by the median data structure need not be consistent with *any* database! However, it will have the useful property that whenever it makes an

error, it will rule out at least half of the data sets in its collection as being inconsistent with the true data set.

The median mechanism is then very simple:

Algorithm 9 The Median Mechanism (MM) Update Rule. It inputs and outputs a median data structure. It is instantiated with an α-net $\mathcal{N}_\alpha(\mathcal{Q})$ for a query class $\mathcal{Q}$, and its initial state is $\mathbf{D} = \mathcal{N}_\alpha(\mathcal{Q})$

$MM_{\alpha,\mathcal{Q}}(\mathbf{D}^t, f_t, v_t)$:
  if $\mathbf{D}^t = \perp$ **then**
    Output $\mathbf{D}^0 \leftarrow N_\alpha(\mathcal{Q})$.
  end if
  if $v_t < f_t(\mathbf{D}^t)$ **then**
    Output $\mathbf{D}^{t+1} \leftarrow \mathbf{D}^t \setminus \{x \in \mathbf{D} : f_t(x) \geq f_t(\mathbf{D^t})\}$.
  else
    Output $\mathbf{D}^{t+1} \leftarrow \mathbf{D}^t \setminus \{x \in \mathbf{D} : f_t(x) \leq f_t(\mathbf{D^t})\}$.
  end if

The intuition for the median mechanism is as follows. It maintains a set of databases that are consistent with the answers to the distinguishing queries it has seen so far. Whenever it receives a query and answer that differ substantially from the real database, it updates itself to remove all of the databases that are inconsistent with the new information. Because it always chooses its answer as the median database among the set of consistent databases it is maintaining, every update step removes at least half of the consistent databases! Moreover, because the set of databases that it chooses initially is an α-net with respect to $\mathcal{Q}$, there is always some database that is never removed, because it remains consistent on all queries. This limits how many update rounds the mechanism can perform. How does the median mechanism do?

Theorem 5.8. For any class of queries $\mathcal{Q}$, The Median Mechanism is a $T(\alpha)$-database update algorithm for $T(\alpha) = \log |N_\alpha(\mathcal{Q})|$.

Proof. We must show that any sequence $\{(D^t, f_t, v_t)\}_{t=1,\ldots,L}$ with the property that $|f^t(\mathbf{D}^t) - f^t(x)| > \alpha$ and $|v_t - f^t(x)| < \alpha$ cannot have $L > \log |\mathcal{N}_\alpha(\mathcal{Q})|$. First observe that because $\mathbf{D}^0 = \mathcal{N}_\alpha(\mathcal{Q})$ is an α-net

for $\mathcal{Q}$, by definition there is at least one y such that $y \in \mathbf{D}^t$ for all t (Recall that the update rule is only invoked on queries with error at least α. Since there is guaranteed to be a database y that has error less than α on all queries, it is never removed by an update step). Thus, we can always answer queries with $\mathbf{D}^t$, and for all t, $|\mathbf{D}^t| \geq 1$. Next observe that for each t, $|\mathbf{D}^t| \leq |\mathbf{D}^{t-1}|/2$. This is because each update step removes at least half of the elements: all of the elements at least as large as, or at most as large as the median element in $\mathbf{D}^t$ with respect to query f_t. Therefore, after L update steps, $|\mathbf{D}^L| \leq 1/2^L \cdot |\mathcal{N}_\alpha(\mathcal{Q})|$. Setting $L > \log |\mathcal{N}_\alpha(\mathcal{Q})|$ gives $|\mathbf{D}^L| < 1$, a contradiction. □

Remark 5.2. For classes of linear queries $\mathcal{Q}$, we may refer to the upper bound on $\mathcal{N}_\alpha(\mathcal{Q})$ given in Theorem 4.2 to see that the Median Mechanism is a $T(\alpha)$-database update algorithm for $T(\alpha) = \log |\mathcal{Q}| \log |\mathcal{X}|/\alpha^2$. This is worse than the bound we gave for the Multiplicative Weights algorithm by a factor of $\log |\mathcal{Q}|$. On the other hand, nothing about the Median Mechanism algorithm is specific to linear queries — it works just as well for any class of queries that admits a small net. We can take advantage of this fact for nonlinear low sensitivity queries.

Note that if we want a mechanism which promises (ε, δ)-privacy for $\delta > 0$, we do not even need a particularly small net. In fact, the trivial net that simply includes every database of size $\|x\|_1$ will be sufficient:

Theorem 5.9. For every class of queries $\mathcal{Q}$ and every $\alpha \geq 0$, there is an α-net for databases of size $\|x\|_1 = n$ of size $\mathcal{N}_\alpha(\mathcal{Q}) \leq |\mathcal{X}|^n$.

Proof. We can simply let $\mathcal{N}_\alpha(\mathcal{Q})$ be the set of all $|\mathcal{X}|^n$ databases y of size $\|y\|_1 = n$. Then, for every x such that $\|x\|_1 = n$, we have $x \in \mathcal{N}_\alpha(\mathcal{Q})$, and so clearly: $\min_{y \in \mathcal{N}_\alpha(\mathcal{Q})} \max_{f \in \mathcal{Q}} |f(x) - f(y)| = 0$. □

We can use this fact to get query release algorithms for *arbitrary* low sensitivity queries, not just linear queries. Applying Theorem 5.7 to the above bound, we find:

Theorem 5.10. Using the median mechanism, together with the exponential mechanism distinguisher, the IC mechanism is (ε, δ)-differentially private and with probability at least $1 - \beta$, the IC algorithm returns a database y such that: $\max_{f \in \mathcal{Q}} |f(x) - f(y)| \leq \alpha$ where:

$$\alpha \leq \frac{16\sqrt{\log|\mathcal{X}| \log \frac{1}{\delta}} \log\left(\frac{2|\mathcal{Q}|n \log|\mathcal{X}|}{\beta}\right)}{\sqrt{n}\varepsilon},$$

where $\mathcal{Q}$ can be *any* family of sensitivity $1/n$ queries, not necessarily linear.

Proof. This follows simply by combining Theorems 5.8 and 5.9 to find that the Median Mechanism is a $T(\alpha)$-Database Update Algorithm for $T(\alpha) = n \log|\mathcal{X}|$ for databases of size $\|x\|_1 = n$ for every $\alpha > 0$ and every class of queries $\mathcal{Q}$. Plugging this into Theorem 5.7 gives the desired bound. □

Note that this bound is almost as good as we were able to achieve for the special case of linear queries in Theorem 4.15! However, unlike in the case of linear queries, because arbitrary queries may not have α-nets which are significantly smaller than the trivial net used here, we are not able to get nontrivial accuracy guarantees if we want $(\varepsilon, 0)$-differential privacy.

The next database update algorithm we present is again for linear queries, but achieves incomparable bounds to those of the multiplicative weights database update algorithm. It is based on the *Perceptron* algorithm from online learning (just as multiplicative weights is derived from the *hedge* algorithm from online learning). Since the algorithm is for linear queries, we treat each query $f_t \in \mathcal{Q}$ as being a vector $f_t \in [0,1]^{|\mathcal{X}|}$. Note that rather than doing a multiplicative update,

Algorithm 10 The Perceptron update rule

Perceptron$_{\alpha,\mathcal{Q}}(x^t, f_t, v_t)$:

If: $x^t = \perp$ **then:** output $x^{t+1} = 0^{|\mathcal{X}|}$

Else if: $f_t(x^t) > v_t$ **then:** output $x^{t+1} = x^t - \frac{\alpha}{|\mathcal{X}|} \cdot f_t$

Else if: $f_t(x^t) \leq v_t$ **then:** output $x^{t+1} = x^t + \frac{\alpha}{|\mathcal{X}|} \cdot f_t$

as in the MW database update algorithm, here we do an additive update. In the analysis, we will see that this database update algorithm has an exponentially worse dependence (as compared to multiplicative weights) on the size of the universe, but a superior dependence on the size of the database. Thus, it will achieve better performance for databases that are large compared to the size of the data universe, and worse performance for databases that are small compared to the size of the data universe.

Theorem 5.11. Perceptron is a $T(\alpha)$-database update algorithm for:

$$T(\alpha) = \left(\frac{\|x\|_2}{\|x\|_1}\right)^2 \cdot \frac{|\mathcal{X}|}{\alpha^2}.$$

Proof. Unlike for multiplicative weights, it will be more convenient to analyze the Perceptron algorithm without normalizing the database to be a probability distribution, and then prove that it is a $T(\alpha')$ database update algorithm for $T(\alpha') = \frac{\|x\|_2^2|\mathcal{X}|}{\alpha'^2}$. Plugging in $\alpha' = \alpha\|x\|_1$ will then complete the proof. Recall that since each query f_t is linear, we can view $f_t \in [0,1]^{|\mathcal{X}|}$ as a vector with the evaluation of $f_t(x)$ being equal to $\langle f_t, x\rangle$.

We must show that any sequence $\{(x^t, f_t, v_t)\}_{t=1,\ldots,L}$ with the property that $|f_t(x^t) - f_t(x)| > \alpha'$ and $|v_t - f_t(x)| < \alpha'$ cannot have $L > \frac{\|x\|_2^2|\mathcal{X}|}{\alpha'^2}$.

We use a potential argument to show that for every $t = 1, 2, \ldots, L$, x^{t+1} is significantly closer to x than x^t. Specifically, our potential function is the L_2^2 norm of the database $x - x^t$, defined as

$$\|x\|_2^2 = \sum_{i \in \mathcal{X}} x(i)^2.$$

Observe that $\|x - x^1\|_2^2 = \|x\|_2^2$ since $x^1 = 0$, and $\|x\|_2^2 \geq 0$. Thus it suffices to show that in every step, the potential decreases by $\alpha'^2/|\mathcal{X}|$. We analyze the case where $f_t(x^t) > v_t$, the analysis for the opposite case will be similar. Let $R^t = x^t - x$. Observe that in this case we have

$$f_t(R^t) = f_t(x^t) - f_t(x) \geq \alpha'.$$

Now we can analyze the drop in potential.

$$\begin{aligned}
\|R^t\|_2^2 - \|R^{t+1}\|_2^2 &= \|R^t\|_2^2 - \|R^t - (\alpha'/|\mathcal{X}|)\cdot f_t\|_2^2 \\
&= \sum_{i\in\mathcal{X}} ((R^t(i))^2 - (R^t(i) - (\alpha'/|\mathcal{X}|)\cdot f_t(i))^2) \\
&= \sum_{i\in\mathcal{X}} \left(\frac{2\alpha'}{|\mathcal{X}|}\cdot R^t(i) f_t(i) - \frac{\alpha'^2}{|\mathcal{X}|^2} f_t(i)^2 \right) \\
&= \frac{2\alpha'}{|\mathcal{X}|} f_t(R^t) - \frac{\alpha'^2}{|\mathcal{X}|^2} \sum_{i\in\mathcal{X}} f_t(i)^2 \\
&\geq \frac{2\alpha'}{|\mathcal{X}|} f_t(R^t) - \frac{\alpha'^2}{|\mathcal{X}|^2} |\mathcal{X}| \\
&\geq \frac{2\alpha'^2}{|\mathcal{X}|} - \frac{\alpha'^2}{|\mathcal{X}|} = \frac{\alpha'^2}{|\mathcal{X}|}.
\end{aligned}$$

This bounds the number of steps by $\|x\|_2^2|\mathcal{X}|/\alpha'^2$, and completes the proof. □

We may now plug this bound into Theorem 5.7 to obtain the following bound on the iterative construction mechanism:

Theorem 5.12. Using the perceptron database update algorithm, together with the exponential mechanism distinguisher, the IC mechanism is (ε, δ)-differentially private and with probability at least $1-\beta$, the IC algorithm returns a database y such that: $\max_{f\in\mathcal{Q}} |f(x) - f(y)| \leq \alpha$ where:

$$\alpha \leq \frac{4\sqrt{4}\sqrt{\|x\|_2}\,(4|\mathcal{X}|\ln(1/\delta))^{1/4}\sqrt{\frac{\log(2|\mathcal{Q}\|\mathcal{X}|\cdot\|x\|_2^2)}{\beta}}}{\sqrt{\epsilon}\|x\|_1},$$

where $\mathcal{Q}$ is a class of linear queries.

If the database x represents the edge set of a graph, for example, we will have $x_i \in [0,1]$ for all i, and so:

$$\frac{\sqrt{\|x\|_2}}{\|x\|_1} \leq \left(\frac{1}{\|x\|_1}\right)^{3/4}.$$

Therefore, the perceptron database update algorithm will outperform the multiplicative weights database update algorithm on dense graphs.

5.2.2 Iterative construction mechanisms and online algorithms

In this section, we generalize the iterative construction framework to the online setting by using the NumericSparse algorithm. The online multiplicative weights algorithm which saw in the last chapter is an instantiation of this approach. One way of viewing the online algorithm is that the NumericSparse algorithm is serving as the private distinguisher in the IC framework, but that the "hard work" of distinguishing is being foisted upon the unsuspecting user. That is: if the user asks a query that does not serve as a good distinguishing query, this is a good case. We cannot use the database update algorithm to update our hypothesis, but we don't need to! By definition, the current hypothesis is a good approximation to the private database with respect to this query. On the other hand, if the user asks a query for which our current hypothesis is not a good approximation to the true database, then by definition the user has found a good distinguishing query, and we are again in a good case — we can run the database update algorithm to update our hypothesis!

The idea of this algorithm is very simple. We will use a database update algorithm to publicly maintain a hypothesis database. Every time a query arrives, we will classify it as either a hard query, or an easy query. An easy query is one for which the answer given by the hypothesis database is approximately correct, and no update step is needed: if we know that a given query is easy, we can simply compute its answer on the publicly known hypothesis database rather than on the private database, and incur no privacy loss. If we know that a query is hard, we can compute and release its answer using the Laplace mechanism, and update our hypothesis using the database update algorithm. This way, our total privacy loss is not proportional to the number of queries asked, but instead proportional to the number of *hard* queries asked. Because the database update algorithm guarantees that there will not need to be many update steps, we can be guaranteed that the total privacy loss will be small.

Theorem 5.13. OnlineIC is (ε, δ)-differentially private.

Algorithm 11 The Online Iterative Construction Mechanism parameterized by a $T(\alpha)$-database update algorithm U. It takes as input a private database x, privacy parameters ε, δ, accuracy parameters α and β, and a stream of queries $\{f_i\}$ that may be chosen adaptively from a class of queries $\mathcal{Q}$. It outputs a stream of answers $\{a_i\}$.

OnlineIC$_U(x, \{f_i\}, \varepsilon, \delta, \alpha, \beta)$

Let $c \leftarrow T(\alpha)$,
if $\delta = 0$ **then**
  Let $T \leftarrow \frac{18c(\log(2|\mathcal{Q}|)+\log(4c/\beta))}{\epsilon||x||_1}$
else
  Let $T \leftarrow \frac{(2+32\sqrt{2})\sqrt{c\log\frac{2}{\delta}}(\log k+\log\frac{4c}{\beta})}{\epsilon||x||_1}$
end if
Initialize NumericSparse$(x, \{f'_i\}, T, c, \varepsilon, \delta)$ with a stream of queries $\{f'_i\}$, outputting a stream of answers a'_i.
Let $t \leftarrow 0$, $D^0 \in x$ be such that $D^0_i = 1/|\mathcal{X}|$ for all $i \in [|\mathcal{X}|]$.
for each query f_i **do**
  Let $f'_{2i-1}(\cdot) = f_i(\cdot) - f_i(D^t)$.
  Let $f'_{2i}(\cdot) = f_i(D^t) - f_i(\cdot)$
  if $a'_{2i-1} = \perp$ and $a'_{2i} = \perp$ **then**
    Let $a_i = f_i(D^t)$
  else
    if $a'_{2i-1} \in \mathbb{R}$ **then**
      Let $a_i = f_i(D^t) + a'_{2i-1}$
    else
      Let $a_i = f_i(D^t) - a'_{2i}$
    end if
    Let $D^{t+1} = U(D^t, f_i, a_i)$
    Let $t \leftarrow t + 1$.
  end if
end for

Proof. This follows directly from the privacy analysis of NumericSparse, because the OnlineIC algorithm accesses the database only through NumericSparse. □

Theorem 5.14. For $\delta = 0$, With probability at least $1 - \beta$, for all queries f_i, OnlineIC returns an answer a_i such that $|f_i(x) - a_i| \leq 3\alpha$ for any α such that:

$$\alpha \geq \frac{9T(\alpha)(\log(2|\mathcal{Q}|) + \log(4T(\alpha)/\beta))}{\epsilon||x||_1}.$$

Proof. Recall that by Theorem 3.28 that given k queries and a maximum number of above-threshold queries of c, Sparse Vector is (α, β)-accurate for:

$$\alpha = \frac{9c(\log k + \log(4c/\beta))}{\epsilon||x||_1}.$$

Here, we have $c = T(\alpha)$ and $k = 2|\mathcal{Q}|$. Note that we have set the threshold $T = 2\alpha$ in the algorithm. First let us assume that the sparse vector algorithm does not halt prematurely. In this case, by the utility theorem, except with probability at most β, we have for all i such that $a_i = f_i(D^t)$: $|f_i(D) - f_i(D^t)| \leq T + \alpha = 3\alpha$, as we wanted. Additionally, for all i such that $a_i = a'_{2i-1}$ or $a_i = a'_{2i}$, we have $|f_i(D) - a'_i| \leq \alpha$.

Note that we also have for all i such that $a_i = a'_{2i-1}$ or $a_i = a'_{2i}$: $|f_i(D) - f_i(D')| \geq T - \alpha = \alpha$, since $T = 2\alpha$. Therefore, f_i, a_i form a valid step in a database update sequence. Therefore, there can be at most $c = T(\alpha)$ such update steps, and so the Sparse vector algorithm does not halt prematurely. □

Similarly, we can prove a corresponding bound for (ε, δ)-privacy.

Theorem 5.15. For $\delta > 0$, With probability at least $1 - \beta$, for all queries f_i, OnlineIC returns an answer a_i such that $|f_i(x) - a_i| \leq 3\alpha$ for any α such that:

$$\alpha \geq \frac{(\sqrt{512} + 1)(\ln(2|\mathcal{Q}|) + \ln \frac{4T(\alpha)}{\beta})\sqrt{T(\alpha) \ln \frac{2}{\delta}}}{\epsilon||x||_1}$$

We can recover the bounds we proved for online multiplicative weights by recalling that the MW database update algorithm is a $T(\alpha)$-database update algorithm for $T(\alpha) = \frac{4 \log |\mathcal{X}|}{\alpha^2}$. More generally, we have that *any* algorithm in the iterative construction framework can be converted into an algorithm which works in the interactive setting without loss in accuracy. (i.e., we could equally well plug in

the median mechanism database update algorithm or the Perceptron database update algorithm, or any other). Tantalizingly, this means that (at least in the iterative construction framework), there is no gap in the accuracy achievable in the online vs. the offline query release models, despite the fact that the online model seems like it should be more difficult.

5.3 Connections

5.3.1 Iterative construction mechanism and α-nets

The Iterative Construction mechanism is implemented differently than the Net mechanism, but at its heart, its analysis is still based on the existence of small α-nets for the queries C. This connection is explicit for the median mechanism, which is parameterized by a net, but it holds for all database update algorithms. Note that the database output by the iterative database construction algorithm is entirely determined by the at most T functions $f_1, \ldots, f_T \in \mathcal{Q}$ fed into it, as selected by the distinguisher while the algorithm is running. Each of these functions can be indexed by at most $\log |\mathcal{Q}|$ bits, and so every database output by the mechanism can be described using only $T \log |\mathcal{Q}|$ bits. In other words, the IC algorithm itself describes an α-net for $\mathcal{Q}$ of size at most $\mathcal{N}_\alpha(\mathcal{Q}) \leq |\mathcal{Q}|^T$. To obtain error α using the Multiplicative Weights algorithm as an iterative database constructor, it suffices by Theorem 4.10 to take $T = 4 \log |\mathcal{X}|/\alpha^2$, which gives us $\mathcal{N}_\alpha(\mathcal{Q}) \leq |\mathcal{Q}|^{4 \log |\mathcal{X}|/\alpha^2} = |\mathcal{X}|^{4 \log |\mathcal{Q}|/\alpha^2}$. Note that up to the factor of 4 in the exponent, this is exactly the bound we gave using a different α-net in Theorem 4.2! There, we constructed an α-net by considering all collections of $\log |\mathcal{Q}|/\alpha^2$ data points, each of which could be indexed by $\log |\mathcal{X}|$ bits. Here, we considered all collections of $\log |\mathcal{X}|/\alpha^2$ functions in $\mathcal{Q}$, each of which could be indexed by $\log |\mathcal{Q}|$ bits. Both ways, we got α-nets of the same size! Indeed, we could just as well run the Net mechanism using the α-net defined by the IC mechanism, to obtain the same utility bounds. In some sense, one net is the "dual" of the other: one is constructed of databases, the other is constructed of queries, yet both nets are of the same size. We will see the same phenomenon in the

"boosting for queries" algorithm in the next section — it too answers a large number of linear queries using a data structure that is entirely determined by a small "net" of queries.

5.3.2 Agnostic learning

One way of viewing what the IC mechanism is doing is that it is reducing the seemingly (information theoretically) more difficult problem of *query release* to the easier problem of *query distinguishing* or *learning*. Recall that the distinguishing problem is to find the query $f \in \mathcal{Q}$ which varies the most between two databases x and y. Recall that in *learning*, the learner is given a collection of labeled examples $(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \{0, 1\}$, where $y_i \in \{0, 1\}$ is the *label* of x_i. If we view x as representing the *positive examples* in some large data set, and y as representing the *negative examples* in the same data set, then we can see that the problem of distinguishing is exactly the problem of *agnostic learning*. That is, a distinguisher finds the query that best labels the positive examples, even when there is no query in the class that is guaranteed to perfectly label them (Note that in this setting, the same example can appear with both a positive and a negative label — so the reduction still makes sense even when x and y are not disjoint). Intuitively, learning should be an information-theoretically easier problem than query release. The query release problem requires that we release the approximate value of every query f in some class $\mathcal{Q}$, evaluated on the database. In contrast, the agnostic learning problem asks only that we return the evaluation and identity of a single query: the query that best labels the dataset. It is clear that information theoretically, the learning problem is no harder than the query release problem. If we can solve the query release problem on databases x and y, then we can solve the distinguishing problem without any further access to the true private dataset, merely by checking the approximate evaluations of every query $f \in \mathcal{Q}$ on x and y that are made available to us with our query release algorithm. What we have shown in this section is that the reverse is true as well: given access to a private distinguishing or agnostic learning algorithm, we can solve the query release problem by making a small (i.e., only $\log |\mathcal{X}|/\alpha^2$) number of calls to the

private distinguishing algorithm, *with no further access to the private dataset*.

What are the implications of this? It tells us that up to small factors, the information complexity of agnostic learning is equal to the information complexity of query release. Computationally, the reduction is only as efficient as our database update algorithm, which, depending on our setting and algorithm, may or may not be efficient. But it tells us that any sort of information theoretic bound we may prove for the one problem can be ported over to the other problem, and vice versa. For example, most of the algorithms that we have seen (and most of the algorithms that we know about!) ultimately access the dataset by making linear queries via the Laplace mechanism. It turns out that any such algorithm can be seen as operating within the so-called *statistical query* model of data access, defined by Kearns in the context of machine learning. But agnostic learning is very hard in the statistical query model: even ignoring computational considerations, there is no algorithm which can make only a polynomial number of queries to the dataset and agnostically learn conjunctions to subconstant error. For query release this means that, *in the statistical query model*, there is no algorithm for *releasing* conjunctions (i.e., contingency tables) that runs in time polynomial in $1/\alpha$, where α is the desired accuracy level. If there is a privacy preserving query release algorithm with this run-time guarantee, it must operate outside of the SQ model, and therefore look very different from the currently known algorithms.

Because privacy guarantees compose linearly, this also tells us that (up to the possible factor of $\log |\mathcal{X}|/\alpha^2$) we should not expect to be able to privately learn to significantly higher accuracy than we can privately perform query release, and vice versa: an accurate algorithm for the one problem automatically gives us an accurate algorithm for the other.

5.3.3 A game theoretic view of query release

In this section, we take a brief sojourn into game theory to interpret some of the query release algorithms we have (and will see). Let us consider an interaction between two adversarial players, Alice and Bob.

Alice has some set of actions she might take, $\mathcal{A}$, and Bob has a set of actions $\mathcal{B}$. The game is played as follows: simultaneously, Alice picks some action $a \in \mathcal{A}$ (possibly at random), and Bob picks some action $b \in \mathcal{B}$ (possibly at random). Alice experiences a cost $c(a, b) \in [-1, 1]$. Alice wishes to play so as to minimize this cost, and since he is adversarial, Bob wishes to play so as to *maximize* this cost. This is what is called a *zero sum game*.

So how should Alice play? First, we consider an easier question. Suppose we handicap Alice and require that she announce her randomized strategy to Bob before she play it, and allow Bob to respond optimally using this information? If Alice announces that she will draw some action $a \in \mathcal{A}$ according to a probability distribution $\mathcal{D}_A$, then Bob will respond optimally so as to maximize Alice's expected cost. That is, Bob will play:

$$b^* = \arg\max_{b \in \mathcal{B}} \mathbb{E}_{a \sim \mathcal{D}_A}[c(a, b)].$$

Hence, once Alice announces her strategy, she knows what her cost will be, since Bob will be able to respond optimally. Therefore, Alice will wish to play a distribution over actions which *minimizes her cost once Bob responds*. That is, Alice will wish to play the distribution $\mathcal{D}_A$ defined as:

$$\mathcal{D}_A = \arg\min_{\mathcal{D} \in \Delta\mathcal{A}} \max_{b \in \mathcal{B}} \mathbb{E}_{a \sim \mathcal{D}}[c(a, b)].$$

If she plays $\mathcal{D}_A$ (and Bob responds optimally), Alice will experience the lowest possible cost that she can guarantee, with the handicap that she must announce her strategy ahead of time. Such a strategy for Alice is called a *min-max* strategy. Let us call the cost that Alice achieves when playing a min-max strategy Alice's *value* for the game, denoted v^A:

$$v^A = \min_{\mathcal{D} \in \Delta\mathcal{A}} \max_{b \in \mathcal{B}} \mathbb{E}_{a \sim \mathcal{D}}[c(a, b)].$$

We can similarly ask what Bob should play if we instead place *him* at the disadvantage and force him to announce his strategy first to Alice. If he does this, he will play the distribution $\mathcal{D}_B$ over actions $b \in \mathcal{B}$ that *maximizes* Alice's expected cost when Alice responds optimally. We call such a strategy $\mathcal{D}_B$ for Bob a *max-min* strategy. We can define

Bob's value for the game, v^B, as the maximum cost he can ensure by any strategy he might announce:

$$v^B = \max_{\mathcal{D} \in \Delta\mathcal{B}} \min_{a \in \mathcal{A}} \mathbb{E}_{b \sim \mathcal{D}}[c(a, b)].$$

Clearly, $v^B \leq v^A$, since announcing one's strategy is only a handicap.

One of the foundational results of game theory is Von-Neumann's min-max Theorem, which states that in any zero sum game, $v^A = v^B$.[2] In other words, there is no disadvantage to "going first" in a zero sum game, and if players play optimally, we can predict exactly Alice's cost: it will be $v^A = v^b \equiv v$, which we refer to as the value of the game.

Definition 5.7. In a zero sum game defined by action sets $\mathcal{A}, \mathcal{B}$ and a cost function $c : \mathcal{A} \times \mathcal{B} \to [-1, 1]$, let v be the value of the game. An α-approximate min-max strategy is a distribution $\mathcal{D}_A$ such that:

$$\max_{b \in \mathcal{B}} \mathbb{E}_{a \sim \mathcal{D}_A}[c(a, b)] \leq v + \alpha$$

Similarly, an α-approximate max-min strategy is a distribution $\mathcal{D}_B$ such that:

$$\min_{a \in \mathcal{A}} \mathbb{E}_{b \sim \mathcal{D}_B}[c(a, b)] \geq v - \alpha$$

If $\mathcal{D}_A$ and $\mathcal{D}_B$ are both α-approximate min-max and max-min strategies respectively, then we say that the pair $(\mathcal{D}_A, \mathcal{D}_B)$ is an α-approximate Nash equilibrium of the zero sum game.

So how does this relate to query release?

Consider a particular zero sum-game tailored to the problem of releasing a set of linear queries $\mathcal{Q}$ over a data universe $\mathcal{X}$. First, assume without loss of generality that for every $f \in \mathcal{Q}$, there is a query $\hat{f} \in \mathcal{Q}$ such that $\hat{f} = 1 - f$ (i.e., for each $\chi \in \mathcal{X}$, $\hat{f}(\chi) = 1 - f(\chi)$). Define Alice's action set to be $\mathcal{A} = \mathcal{X}$ and define Bob's action set to be $\mathcal{B} = \mathcal{Q}$. We will refer to Alice as the *database player*, and to Bob as the *query player*. Finally, fixing a true private database x normalized to be a probability distribution (i.e., $\|x\|_1 = 1$), define the cost function $c : \mathcal{A} \times \mathcal{B} \to [-1, 1]$

[2] Von Neumann is quoted as saying "As far as I can see, there could be no theory of games ... without that theorem ... I thought there was nothing worth publishing until the Minimax Theorem was proved" [10].

to be: $c(\chi, f) = f(\chi) - f(x)$. Let us call this game the "Query Release Game."

We begin with a simple observation:

Proposition 5.16. The value of the query release game is $v = 0$.

Proof. We first show that $v^A = v \leq 0$. Consider what happens if we let the database player's strategy correspond to the true database: $\mathcal{D}_A = x$. Then we have:

$$\begin{aligned} v^A &\leq \max_{f \in \mathcal{B}} \mathbb{E}_{\chi \sim \mathcal{D}_A}[c(\chi, f)] \\ &= \max_{f \in \mathcal{B}} \sum_{i=1}^{|\mathcal{X}|} f(\chi_i) \cdot x_i - f(x) \\ &= f(x) - f(x) \\ &= 0. \end{aligned}$$

Next we observe that $v = v^B \geq 0$. For point of contradiction, assume that $v < 0$. In other words, that there exists a distribution $\mathcal{D}_A$ such that *for all* $f \in \mathcal{Q}$

$$\mathbb{E}_{\chi \sim \mathcal{D}_A} c(\chi, f) < 0.$$

Here, we simply note that by definition, if $\mathbb{E}_{\chi \sim \mathcal{D}_A} c(\chi, f) = c < 0$ then $\mathbb{E}_{\chi \sim \mathcal{D}_A} c(\chi, \hat{f}) = -c > 0$, which is a contradiction since $\hat{f} \in \mathcal{Q}$. □

What we have established implies that for any distribution $\mathcal{D}_A$ that is an α-approximate min-max strategy for the database player, we have that for all queries $f \in \mathcal{Q}$: $|\mathbb{E}_{\chi \sim \mathcal{D}_A} f(\chi) - f(x)| \leq \alpha$. In other words, the distribution $\mathcal{D}_A$ can be viewed as a synthetic database that answers every query in $\mathcal{Q}$ with α-accuracy.

How about for nonlinear queries? We can repeat the same argument above if we change the query release game slightly. Rather than letting the database player have strategies corresponding to universe elements $\chi \in \mathcal{X}$, we let the database player have strategies corresponding to *databases* themselves! Then, $c(f, y) = |f(x) - f(y)|$. Its not hard to see that this game still has value 0 and that α-approximate min-max strategies correspond to synthetic data which give α-accurate answers to queries in $\mathcal{Q}$.

So how do we compute approximate min-max strategies in zero sum games? There are many ways! It is well known that if Alice plays the game repeatedly, updating her distribution on actions using an online-learning algorithm with a no-regret guarantee (defined in Section 11.2), and Bob responds at each round with an approximately-cost-maximizing response, then Alice's distribution will quickly converge to an approximate min-max strategy. Multiplicative weights is such an algorithm, and one way of understanding the multiplicative weights mechanism is as a strategy for Alice to play in the query release game defined in this section. (The private distinguisher is playing the role of Bob here, picking at each round the query that corresponds to approximately maximizing Alice's cost). The median mechanism is another such algorithm, for the game in which Alice's strategies correspond to databases, rather than universe elements, and so is also computing an approximate min-max solution to the query release game.

However, there are other ways to compute approximate equilibria as well! For example, *Bob*, the query player, could play the game using a no-regret learning algorithm (such as multiplicative weights), and *Alice* could repeatedly respond at each round with an approximately-cost-minimizing database! In this case, the *average* over the databases that Alice plays over the course of this experiment will converge to an approximate min-max solution as well. This is exactly what is being done in Section 6, in which the private base-sanitizer plays the role of Alice, at each round playing an approximately cost-minimizing database given Bob's distribution over queries.

In fact, a third way of computing an approximate equilibrium of a zero-sum game is to have *both* Alice and Bob play according to no-regret learning algorithms. We won't cover this approach here, but this approach has applications in guaranteeing privacy not just to the database, but also to the set of queries being asked, and to privately solving certain types of linear programs.

5.4 Bibliographical notes

The Iterative Construction Mechanism abstraction (together with the perception based database update algorithm) was formalized by

Gupta et al. [39], generalizing the median mechanism of Roth and Roughgarden [74] (initially presented as an online algorithm), the online private multiplicative weights mechanism of Hardt and Rothblum [44], and its offline variant of Gupta et al. [38]; see also Hardt et al. [41]. All these algorithm can be seen to be instantiations. The connection between query release and agnostic learning was observed in [38]. The observation that the median mechanism, when analyzed using the composition theorems of Dwork et al. [32] for (ε, δ) privacy, can be used to answer arbitrary low sensitivity queries is due to Hardt and Rothblum. The game theoretic view of query release, along with its applications to analyst privacy, is due to Hsu, Roth, and Ullman [48].

6

Boosting for Queries

In the previous sections, we have focused on the problem of private query release in which we insist on bounding the worst-case error over all queries. Would our problem be easier if we instead asked only for low error on average, given some distribution over the queries? In this section, we see that the answer is no: given a mechanism which is able to solve the query release problem with low average error given any distribution on queries, we can "boost" it into a mechanism which solves the query release problem to worst-case error. This both sheds light on the difficulty of private query release, and gives us a new tool for designing private query release algorithms.

Boosting is a general and widely used method for improving the accuracy of learning algorithms. Given a set of labeled training examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\},$$

where each x_i is drawn from an underlying distribution $\mathcal{D}$ on a universe $\mathcal{U}$, and each $y_i \in \{+1, -1\}$, a learning algorithm produces a hypothesis $h : \mathcal{U} \to \{+1, -1\}$. Ideally, h will not just "describe" the labeling on the given samples, but will also *generalize*, providing a reasonably accurate method of classifying other elements drawn from the underlying

distribution. The goal of boosting is to convert a weak *base* learner, which produces a hypothesis that may do just a little better than random guessing, into a strong learner, which yields a very accurate predictor for samples drawn according to $\mathcal{D}$. Many boosting algorithms share the following basic structure. First, an initial (typically uniform) probability distribution is imposed on the sample set. Computation then proceeds in rounds. In each round t:

1. The base learner is run on the current distribution, denoted $\mathcal{D}_t$, producing a classification hypothesis h_t; and
2. The hypotheses $h_1, \ldots, h_t$ are used to re-weight the samples, defining a new distribution $\mathcal{D}_{t+1}$.

The process halts either after a predetermined number of rounds or when an appropriate combining of the hypotheses is determined to be sufficiently accurate. Thus, given a base learner, the design decisions for a boosting algorithm are (1) are how to modify the probability distribution from one round to the next, and (2) how to combine the hypotheses $\{h_t\}_{t=1,\ldots,T}$ to form a final output hypothesis.

In this section we will use boosting on queries — that is, for the purposes of the boosting algorithm the universe $\mathcal{U}$ is a set of queries $\mathcal{Q}$ — to obtain an offline algorithm for answering large numbers of arbitrary low-sensitivity queries. This algorithm requires less space than the median mechanism, and, depending on the base learner, is potentially more time efficient as well.

The algorithm revolves around a somewhat magical fact (Lemma 6.5): if we can find a synopsis that provides accurate answers on a few selected queries, then in fact this synopsis provides accurate answers on *most* queries! We apply this fact to the base learner, which samples from a distribution on $\mathcal{Q}$ and produces as output a "weak" synopsis that yields "good" answers for a majority of the weight in $\mathcal{Q}$, boosting, in a differentially private fashion, to obtain a synopsis that is good for all of $\mathcal{Q}$.

Although the boosting is performed over the queries, the privacy is still for the rows of the database. The privacy challenge in boosting for queries comes from the fact that each row in the database affects the

answers to all the queries. This will manifest in the reweighting of the queries: adjacent databases could cause radically different reweightings, which will be observable in the generated h_t that, collectively, will form the synopsis.

The running time of the boosting procedure depends quasi-linearly on the number $|\mathcal{Q}|$ of queries and on the running time of the base synopsis generator, independent of the data universe size $|\mathcal{X}|$. This yields a new avenue for constructing efficient and accurate privacy-preserving mechanisms, analogous to the approach enabled by boosting in the machine learning literature: an algorithm designer can tackle the (potentially much easier) task of constructing a weak privacy-preserving base synopsis generator, and automatically obtain a stronger mechanism.

6.1 The boosting for queries algorithm

We will use the *row representation* for databases, outlined in Section 2, where we think of the database as a multiset of rows, or elements of $\mathcal{X}$. Fix a database size n, a data universe $\mathcal{X}$, and a query set $\mathcal{Q} = \{q : \mathcal{X}^* \to \mathbb{R}\}$ of real-valued queries of sensitivity at most ρ.

We assume the existence of a *base synopsis generator* (in Section 6.2 we will see how to construct these). The property we will need of the base generator, formulated next, is that, for any distribution $\mathcal{D}$ on the query set $\mathcal{Q}$, the output of base generator can be used for computing accurate answers for a *large fraction* of the queries, where the "large fraction" is defined in terms of the weights given by $\mathcal{D}$. The base generator is parameterized by k, the number of queries to be sampled; λ, an accuracy requirement for its outputs; η, a measurement of "large" describing what we mean by a large fraction of the queries, and β, a failure probability.

Definition 6.1 ((k, λ, η, β)-base synopsis generator)**.** For a fixed database size n, data universe $\mathcal{X}$ and query set $\mathcal{Q}$, consider a synopsis generator $\mathcal{M}$, that samples k queries independently from a distribution $\mathcal{D}$ on $\mathcal{Q}$ and outputs a synopsis. We say that $\mathcal{M}$ is a $(k, \lambda, \eta, \beta)$-*base synopsis generator* if for any distribution $\mathcal{D}$ on $\mathcal{Q}$, with all but β probability

over the coin flips of $\mathcal{M}$, the synopsis $\mathcal{S}$ that $\mathcal{M}$ outputs is λ-accurate for a $(1/2+\eta)$-fraction of the mass of $\mathcal{Q}$ as weighted by $\mathcal{D}$:

$$\Pr_{q\sim\mathcal{D}}[|q(\mathcal{S})-q(x)|\leq\lambda]\geq 1/2+\eta. \tag{6.1}$$

The query-boosting algorithm can be used for any class of queries and any differentially private base synopsis generator. The running time is inherited from the base synopsis generator. The booster invests additional time that is quasi-linear in $|\mathcal{Q}|$, and in particular its running time does not depend directly on the size of the data universe.

To specify the boosting algorithm we will need to specify a stopping condition, an aggregation mechanism, and an algorithm for updating the current distribution on $\mathcal{Q}$.

Stopping Condition. We will run the algorithm for a fixed number T of rounds — this will be our stopping condition. T will be selected so as to ensure sufficient accuracy (with very high probability); as we will see, $\log|\mathcal{Q}|/\eta^2$ rounds will suffice.

Updating the Distribution. Although the distributions are never directly revealed in the outputs, the base synopses $\mathcal{A}_1,\mathcal{A}_2,\ldots,\mathcal{A}_T$ are revealed, and each $\mathcal{A}_i$ can in principle leak information about the queries chosen, from $\mathcal{D}_i$, in constructing $\mathcal{A}_i$. We therefore need to constrain the max-divergence between the probability distributions obtained on neighboring databases. This is technically challenging because, given $\mathcal{A}_i$, the database is very heavily involved in constructing $\mathcal{D}_{i+1}$.

The initial distribution, $\mathcal{D}_1$, will be uniform over $\mathcal{Q}$. A standard method for updating $\mathcal{D}_t$ is to increase the weight of poorly handled elements, in our case, queries for which $|q(x)-q(A_t)|>\lambda$, by a fixed factor, say, e, and decrease the weight of well-handled elements by the same factor. (The weights are then normalized so as to sum to 1.) To get a feel for the difficulty, let $x=y\cup\{\xi\}$, and suppose that all queries q are handled well by $\mathcal{A}_t$ when the database is y, but the addition of ξ causes this to fail for, say, a $1/10$ fraction of the queries; that is, $|q(y)-q(A_t)|\leq\lambda$ for all queries q, but $|q(x)-q(A_t)|>\lambda$ for some $|\mathcal{Q}|/10$ queries. Note that, since $\mathcal{A}_t$ "does well" on $9/10$ of the queries even

when the database is x, it could be returned from the base sanitizer no matter which of x, y is the true data set. Our concern is with the effects of the updating: when the database is y all queries are well handled and there is no reweighting (after normalization), but when the database is x there is a reweighting: one tenth of the queries have their weights increased, the remaining nine tenths have their weights decreased. This difference in reweighting may be detected in the next iteration via $\mathcal{A}_{t+1}$, which is observable, and which will be built from samples drawn from rather different distributions depending on whether the database is x or y.

For example, suppose we start from the uniform distribution $\mathcal{D}_1$. Then $\mathcal{D}_2^{(y)} = \mathcal{D}_1^{(y)}$, where by $\mathcal{D}_i^{(z)}$ we mean the distribution at round i when the database is z. This is because the weight of every query is decreased by a factor of e, which disappears in the normalization. So each $q \in \mathcal{Q}$ is assigned weight $1/|\mathcal{Q}|$ in $\mathcal{D}_2^{(y)}$. In contrast, when the database is x the "unhappy" queries have normalized weight

$$\frac{\frac{e}{|\mathcal{Q}|}}{\frac{9}{10}\frac{1}{|\mathcal{Q}|}\frac{1}{e} + \frac{1}{10}\frac{e}{|\mathcal{Q}|}}.$$

Consider any such unhappy query q. The *ratio* $\mathcal{D}_2^{(x)}(q)/\mathcal{D}_2^{(y)}(q)$ is given by

$$\begin{aligned}\frac{\mathcal{D}_2^{(x)}(q)}{\mathcal{D}_2^{(y)}(q)} &= \frac{\frac{\frac{e}{|\mathcal{Q}|}}{\frac{9}{10}\frac{1}{|\mathcal{Q}|}\frac{1}{e} + \frac{1}{10}\frac{e}{|\mathcal{Q}|}}}{\frac{1}{|\mathcal{Q}|}} \\ &= \frac{10}{1+\frac{9}{e^2}} \stackrel{\text{def}}{=} F \approx 4.5085.\end{aligned}$$

Now, $\ln F \approx 1.506$, and even though the choice of queries used in round 2 by the base generator are not explicitly made public, they may be detectable from the resulting $\mathcal{A}_2$, which is made public. Thus, there is a potential privacy loss of up to 1.506 per query (of course, we expect cancellations; we are simply trying to explain the source of the difficulty). This is partially addressed by ensuring that the number of samples used by the base generator is relatively small, although we still have the problem that, over multiple iterations, the distributions $\mathcal{D}_t$ may evolve very differently even on neighboring databases.

The solution will be to attenuate the re-weighting procedure. Instead of always using a fixed ratio either for increasing the weight (when the answer is "accurate") or decreasing it (when it is not), we set separate thresholds for "accuracy" (λ) and "inaccuracy" ($\lambda + \mu$, for an appropriately chosen μ that scales with the *bit size* of the output of the base generator; see Lemma 6.5 below). Queries for which the error is below or above these thresholds have their weight decreased or increased, respectively, by a factor of e. For queries whose error lies between these two thresholds, we scale the natural logarithm of the weight change linearly: $1 - 2(|q(x) - q(A_t)| - \lambda)/\mu$, so queries with errors of magnitude exceeding $\lambda + \mu/2$ increase in weight, and those with errors of magnitude less than $\lambda + \mu/2$ decrease in weight.

The attenuated scaling reduces the effect of any individual on the re-weighting of any query. This is because an individual can only affect the true answer to a query — and thus also the accuracy of the base synopsis generator's output $q(A_t)$ — by a small amount, and the attenuation divides this amount by a parameter μ which will be chosen to compensate for the kT samples chosen (total) from the T distributions obtained over the course of the execution of the boosting algorithm. This helps to ensure privacy. Intuitively, we view each of these kT samples as a "mini-mechanism." We first bound the privacy loss of sampling at any round (Claim 6.4) and then bound the cumulative loss via the composition theorem.

The larger the gap (μ) between the thresholds for "accurate" and "inaccurate," the smaller the effect of each individual on a query's weight can be. This means that larger gaps are better for privacy. For accuracy, however, large gaps are bad. If the inaccuracy threshold is large, we can only guarantee that queries for which the base synopsis generator is very inaccurate have their weight substantially increased during re-weighting. This degrades the accuracy guarantee of the boosting algorithm: the errors are roughly equal to the "inaccuracy" threshold ($\lambda + \mu$).

Aggregation. For $t \in [T]$ we will run the base generator to obtain a synopsis $\mathcal{A}_t$. The synopses will be aggregated by taking the median: given $\mathcal{A}_1, \ldots, \mathcal{A}_T$, the quantity $q(x)$ is estimated by taking the T

approximate values for $q(x)$ computed using each of the $\mathcal{A}_i$, and then computing their median. With this aggregation method we can show accuracy for query q by arguing that a majority of the $\mathcal{A}_i$, $1 \leq i \leq T$ provide $\lambda + \mu$ accuracy (or better) for q. This implies that the median value of the T approximations to $q(x)$ will be within $\lambda + \mu$ of the true value.

Notation.

1. Throughout the algorithm's operation, we keep track of several variables (explicitly or implicitly). Variables indexed by $q \in \mathcal{Q}$ hold information pertaining to query q in the query set. Variables indexed by $t \in [T]$, usually computed in round t, will be used to construct the distribution $\mathcal{D}_{t+1}$ used for sampling in time period $t + 1$.
2. For a predicate P we use $[[P]]$ to denote 1 if the predicate is true and 0 if it is false.
3. There is a final tuning parameter α used in the algorithm. It will be chosen (see Corollary 6.3 below) to have value

$$\alpha = \alpha(\eta) = (1/2) \ln \left(\frac{1 + 2\eta}{1 - 2\eta} \right).$$

The algorithm appears in Figure 6.1. The quantity $u_{t,q}$ in Step 2(2b) is the new, un-normalized, weight of the query. For the moment, let us set $\alpha = 1$ (just so that we can ignore any α factors). Letting $a_{j,q}$ be the natural logarithm of the weight change in round j, $1 \leq j \leq t$, the new weight is given by:

$$u_{t,q} \leftarrow \exp \left(- \sum_{j=1}^{t} a_{j,q} \right).$$

Thus, at the end of the previous step the un-normalized weight was $u_{t-1,q} = \exp(-\sum_{j=1}^{t-1} a_{j,q})$ and the update corresponds to multiplication by $e^{-a_{j,t}}$. When the sum $\sum_{j=1}^{t} a_{j,q}$ is large, the weight is small. Every time a synopsis gives a very good approximation to $q(x)$, we add 1 to this sum; if the approximation is only moderately good (between λ and

Boosting for Queries$(k, \lambda, \eta, \rho, \mu, T)$
Given: database $x \in \mathcal{X}^n$, query set $\mathcal{Q}$, where each $q \in \mathcal{Q}$ is a function $q : X^n \to \mathbb{R}$ with sensitivity at most ρ.
Initialize $\mathcal{D}_1$ to be the uniform distribution over $\mathcal{Q}$.
For $t = 1, \ldots, T$:

1. Sample a sequence $S_t \subseteq \mathcal{Q}$ of k samples chosen independently and at random from $\mathcal{D}_t$.
   Run the base synopsis generator to compute a synopsis $\mathcal{A}_t : \mathcal{Q} \to \mathbb{R}$ that is w.h.p. accurate for at least $1/2 + \eta$ of the mass of $\mathcal{D}_t$.
2. Reweight the queries. For each $q \in \mathcal{Q}$:
   (a) If $\mathcal{A}_t$ is λ-accurate, then $a_{t,q} \leftarrow 1$
   If $\mathcal{A}_t$ is $(\lambda + \mu)$-inaccurate, then $a_{t,q} \leftarrow -1$
   Otherwise, let $d_{q,t} = |q(x) - \mathcal{A}_t(q)|$ be the error of $\mathcal{A}_t$ (between λ and $\lambda + \mu$) on q:
   $$a_{t,q} \leftarrow 1 - 2(d_{t,q} - \lambda)/\mu$$
   (b) $u_{t,q} \leftarrow \exp(-\alpha \cdot \sum_{j=1}^{t} a_{j,q})$, where $\alpha = (1/2)\ln((1+2\eta)/(1-2\eta))$.
3. Renormalize:
   $$Z_t \leftarrow \sum_{q \in \mathcal{Q}} u_{t,q}$$
   $$D_{t+1}[q] = u_{t,q}/Z_t$$

Output the final answer data structure $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_T)$. For $q \in \mathcal{Q}$:
$$\mathcal{A}(q) = median\{\mathcal{A}_1(q), \ldots, \mathcal{A}_T(q)\}$$

Figure 6.1: Boosting for queries.

$\lambda + \mu/2$), we add a positive amount, but less than 1. Conversely, when the synopsis is very bad (worse than $\lambda + \mu$ accuracy), we subtract 1; when it is barely acceptable (between $\lambda + \mu/2$ and $\lambda + \mu$), we subtract a smaller amount.

In the theorem below we see an inverse relationship between privacy loss due to sampling, captured by $\varepsilon_{\text{sample}}$, and the gap μ between the thresholds for accurate and inaccurate.

Theorem 6.1. Let $\mathcal{Q}$ be a query family with sensitivity at most ρ. For an appropriate setting of parameters, and with $T = \log|\mathcal{Q}|/\eta^2$ rounds, the algorithm of Figure 6.1 is an accurate and differentially private query-boosting algorithm:

1. When instantiated with a $(k, \lambda, \eta, \beta)$-base synopsis generator, the output of the boosting algorithm gives $(\lambda + \mu)$-accurate answers to *all* the queries in $\mathcal{Q}$ with probability at least $1 - T\beta$, where
   $$\mu \in O(((\log^{3/2}|Q|)\sqrt{k}\sqrt{\log(1/\beta)}\rho)/(\varepsilon_{\text{sample}} \cdot \eta^3)). \quad (6.2)$$

2. If the base synopsis generator is $(\varepsilon_{\text{base}}, \delta_{\text{base}})$-differentially private, then the boosting algorithm is $(\varepsilon_{\text{sample}} + T \cdot \varepsilon_{\text{base}}, \delta_{\text{sample}} + T\delta_{\text{base}})$-differentially private.

Allowing the constant η to be swallowed up into the big-O notation, and taking $\rho = 1$ for simplicity, we get $\mu = O(((\log^{3/2}|Q|)\sqrt{k}\sqrt{\log(1/\beta)})/\varepsilon_{\text{sample}})$. Thus we see that reducing the number k of input queries needed by the base sanitizer improves the quality of the output. Similarly, from the full statement of the theorem, we see that improving the generalization power of the base sanitizer, which corresponds to having a larger value of η (a bigger "strong majority"), also improves the accuracy.

Proof of Theorem 6.1. We first prove accuracy, then privacy.

We introduce the notation $a^-_{t,q}$ and $a^+_{t,q}$, satisfying

1. $a^-_{t,q}, a^+_{t,q} \in \{-1, 1\}$; and
2. $a^-_{t,q} \leq a_{t,q} \leq a^+_{t,q}$.

Recall that a larger $a_{t,q}$ indicates a higher quality of the approximation of the synopsis $\mathcal{A}_t$ for $q(x)$.

1. $a^-_{t,q}$ is 1 if $\mathcal{A}_t$ is λ-accurate on q, and -1 otherwise. To check that $a^-_{t,q} \leq a_{t,q}$, note that if $a^-_{t,q} = 1$ then $\mathcal{A}_t$ is λ-accurate for q, and so by definition $a_{t,q} = 1$ as well. If instead we have $a^-_{t,q} = -1$ then since we always have $a_{t,q} \in [-1, 1]$, we are done.

   We will use the $a^-_{t,q}$ to lower bound a measure of the quality of the output of the base generator. By the promise of the base generator, $\mathcal{A}_t$ is λ-accurate for at least a $1/2 + \eta$ fraction of the mass of $\mathcal{D}_t$. Thus,

$$r_t \triangleq \sum_{q \in \mathcal{Q}} \mathcal{D}_t[q] \cdot a^-_{t,q} \geq (1/2 + \eta) - (1/2 - \eta) = 2\eta. \tag{6.3}$$

2. $a^+_{t,q}$ is -1 if $\mathcal{A}_t$ is $(\lambda + \mu)$-*inaccurate* for q, and 1 otherwise. To check that $a_{t,q} \leq a^+_{t,q}$, note that if $a^+_{t,q} = -1$ then $\mathcal{A}_t$ is $(\lambda + \mu)$-*inaccurate* for q, so by definition $a_{t,q} = -1$ as well. If instead $a^+_{t,q} = 1$ then since we always have $a_{t,q} \in [-1, 1]$, we are done.

   Thus $a^+_{t,q}$ is positive if and only if $\mathcal{A}_t$ is at least minimally adequately accurate for q. We will use the $a^+_{t,q}$ to prove accuracy

of the aggregation. When we sum the values $a^+_{t,q}$, we get a positive number if and only if the majority of the $\mathcal{A}_t$ are providing passable — that is, within $\lambda + \mu$ — approximations to $q(x)$. In this case the median value will be within $\lambda + \mu$.

Lemma 6.2. After T rounds of boosting, with all but $T\beta$ probability, the answers to all but an $\exp(-\eta^2 T)$-fraction of the queries are $(\lambda+\mu)$-accurate.

Proof. In the last round of boosting, we have:

$$\mathcal{D}_{T+1}[q] = \frac{u_{T,q}}{Z_T}. \tag{6.4}$$

Since $a_{t,q} \leq a^+_{t,q}$ we have:

$$u^+_{T,q} \triangleq e^{-\alpha \sum_{t=1}^{T} a^+_{t,q}} \leq e^{-\alpha \sum_{t=1}^{T} a_{t,q}} = u_{T,q}. \tag{6.5}$$

(The superscript "+" reminds us that this unweighted value was computed using the terms $a^+_{t,q}$.) Note that we always have $u^+_{T,q} \geq 0$. Combining Equations (6.4) and (6.5), for all $q \in \mathcal{Q}$:

$$\mathcal{D}_{T+1}[q] \geq \frac{u^+_{T,q}}{Z_T}. \tag{6.6}$$

Recalling that $[[P]]$ denotes the boolean variable that has value 1 if and only if the predicate P is true, we turn to examining the value $[[\mathcal{A}$ is $(\lambda+\mu)$-inaccurate for $q]]$. If this predicate is 1, then it must be the case that the majority of $\{\mathcal{A}_j\}_{j=1}^T$ are $(\lambda+\mu)$-inaccurate, as otherwise their median would be $(\lambda+\mu)$-accurate.

From our discussion of the significance of the sign of $\sum_{t=1}^{T} a^+_{t,q}$, we have:

$$\begin{aligned} \mathcal{A} \text{ is } (\lambda+\mu)\text{-inaccurate for } q &\Rightarrow \sum_{t=1}^{T} a^+_{t,q} \leq 0 \\ &\Leftrightarrow e^{-\alpha \sum_{t=1}^{T} a^+_{t,q}} \geq 1 \\ &\Leftrightarrow u^+_{T,q} \geq 1 \end{aligned}$$

Since $u^+_{T,q} \geq 0$, We conclude that:

$$[[\mathcal{A} \text{ is } (\lambda+\mu)\text{-inaccurate for } q]] \leq u^+_{T,q}$$

Using this together with Equation (6.6) yields:

$$\begin{aligned}\frac{1}{|\mathcal{Q}|}\cdot\sum_{q\in\mathcal{Q}}[[\mathcal{A}\text{ is }(\lambda+\mu)\text{-inaccurate for }q]] &\leq \frac{1}{|\mathcal{Q}|}\cdot\sum_{q\in\mathcal{Q}}u^{+}_{T,q}\\ &\leq \frac{1}{|\mathcal{Q}|}\cdot\sum_{q\in\mathcal{Q}}\mathcal{D}_{T+1}[q]\cdot Z_T\\ &= \frac{Z_T}{|\mathcal{Q}|}.\end{aligned}$$

Thus the following claim completes the proof:

Claim 6.3. In round t of boosting, with all but $t\beta$ probability:

$$Z_t \leq \exp(-\eta^2\cdot t)\cdot|\mathcal{Q}|$$

Proof. By definition of a base synopsis generator, with all but β probability, the synopsis generated is λ-accurate for at least a $(1/2+\eta)$-fraction of the mass of the distribution $\mathcal{D}_t$. Recall that $a^{-}_{t,q}\in\{-1,1\}$ is 1 if and only if $\mathcal{A}_t$ is λ-accurate on q, and that $a^{-}_{t,q}\leq a_{t,q}$ and recall further the quantity $r_t \triangleq \sum_{q\in\mathcal{Q}}\mathcal{D}_t[q]\cdot a^{-}_{t,q}$ defined in Equation (6.3). As discussed above, r_t measures the "success" of the base synopsis generator in round t, where by "success" we mean the stricter notion of λ-accuracy. As summarized in Equation (6.3), if a $(1/2+\eta)$-fraction of the mass of $\mathcal{D}_t$ is computed with λ-accuracy, then $r_t\geq 2\eta$. Now observe also that for $t\in[T]$, assuming the base sanitizer did not fail in round t:

$$\begin{aligned}Z_t &= \sum_{q\in\mathcal{Q}}u_{t,q}\\ &= \sum_{q\in\mathcal{Q}}u_{t-1,q}\cdot e^{-\alpha\cdot a_{t,q}}\\ &= \sum_{q\in\mathcal{Q}}Z_{t-1}\cdot\mathcal{D}_t[q]\cdot e^{-\alpha\cdot a_{t,q}}\\ &\leq \sum_{q\in\mathcal{Q}}Z_{t-1}\cdot\mathcal{D}_t[q]\cdot e^{-\alpha\cdot a^{-}_{t,q}}\\ &= Z_{t-1}\cdot\sum_{q\in\mathcal{Q}}\mathcal{D}_t[q]\cdot\left(\left(\frac{1+a^{-}_{t,q}}{2}\right)\cdot e^{-\alpha}+\left(\frac{1-a^{-}_{t,q}}{2}\right)\cdot e^{\alpha}\right)\end{aligned}$$

(case analysis)

$$= \frac{Z_{t-1}}{2}\left[(e^{\alpha}+e^{-\alpha})+r_t(e^{-\alpha}-e^{\alpha})\right]$$
$$\leq \frac{Z_{t-1}}{2}\left[(e^{\alpha}+e^{-\alpha})+2\eta(e^{-\alpha}-e^{\alpha})\right] \ (r_t \geq 2\eta \text{ and } (e^{-\alpha}-e^{\alpha}) \leq 0)$$

By simple calculus we see that $(e^{\alpha}+e^{-\alpha})+2\eta(e^{-\alpha}-e^{\alpha})$ is minimized when

$$\alpha = (1/2)\ln\left(\frac{1+2\eta}{1-2\eta}\right).$$

Plugging this into the recurrence, we get

$$Z_t \leq (\sqrt{1-4\eta^2})^t|\mathcal{Q}| \leq \exp(-2\eta^2 t)|\mathcal{Q}|.$$ □

This completes the proof of Lemma 6.2. □

The lemma implies that accuracy for *all* queries simultaneously can be achieved by setting

$$T > \frac{\ln|\mathcal{Q}|}{\eta^2}.$$

Privacy. We will show that the entire sequence $(S_1, \mathcal{A}_1, \ldots, S_T, \mathcal{A}_T)$ can be output while preserving differential privacy. Note that this is stronger than we need — we do not actually output the sets $S_1, \ldots, S_T$. By our adaptive composition theorems, the privacy of each $\mathcal{A}_i$ will be guaranteed by the privacy guarantees of the base synopsis generator, together with the fact that S_{i-1} was computed in a differentially private way. Therefore, it suffices to prove that given that $(S_1, \mathcal{A}_1, \ldots, S_i, \mathcal{A}_i)$ is differentially private, S_{i+1} is as well. We can then combine the privacy parameters using our composition theorems to compute a final guarantee.

Lemma 6.4. Let $\varepsilon^* = \frac{4\alpha T\rho}{\mu}$. For all $i \in [T]$, once $(S_1, \mathcal{A}_1, \ldots, S_i, \mathcal{A}_i)$ is fixed, the computation of each element of S_{i+1} is $(\varepsilon^*, 0)$-differentially private.

Proof. Fixing $\mathcal{A}_1, \ldots, \mathcal{A}_i$, for every $j \leq i$, the quantity $d_{q,j}$ has sensitivity ρ, since $\mathcal{A}_j(q)$ is database independent (because $\mathcal{A}_j$ is fixed), and

every $q \in \mathcal{Q}$ has sensitivity bounded by ρ. Therefore, for every $j \leq i$, $a_{j,q}$ is $2\rho/\mu$ sensitive by construction, and so

$$g_i(q) \stackrel{\text{def}}{=} \sum_{j=1}^{i} a_{j,q}$$

has sensitivity at most $2i\rho/\mu \leq 2T\rho/\mu$. Then $\Delta g_i \stackrel{\text{def}}{=} 2T\rho/\mu$ is an upper bound on the sensitivity of g_i.

To argue privacy, we will show that the selection of queries for S_{i+1} is an instance of the exponential mechanism. Think of $-g_i(q)$ as the utility of a query q during the selection process at round $i+1$. The exponential mechanism says that to achieve $(\varepsilon^*, 0)$-differential privacy we should choose q with probability proportional to

$$\exp\left(-g_i(q)\frac{\varepsilon^*}{2\Delta g_i}\right).$$

Since $\varepsilon^*/2\Delta g_i = \alpha$ and the algorithm selects q with probability proportional to $e^{-\alpha g_i(q)}$, we see that this is exactly what the algorithm does! □

We bound the privacy loss of releasing the S_is by treating each selection of a query as a "mini-mechanism" that, over the course of T rounds of boosting, is invoked kT times. By Lemma 6.4 each mini-mechanism is $(4\alpha T\rho/\mu, 0)$-differentially private. By Theorem 3.20, for all $\beta > 0$ the composition of kT mechanisms, each of which is $(\alpha 4T\rho/\mu, 0)$-differentially private, is $(\varepsilon_{\text{sample}}, \delta_{\text{sample}})$-differentially private, where

$$\varepsilon_{\text{sample}} \stackrel{\text{def}}{=} \sqrt{2kT\log(1/\delta_{\text{sample}})}(\alpha 4T\rho/\mu) + kT\left(\frac{\alpha 4T\rho}{\mu}\right)^2. \quad (6.7)$$

Our total privacy loss comes from the composition of T calls to the base sanitizer and the cumulative loss from the kT samples. We conclude that the boosting algorithm in its entirety is: $(\varepsilon_{\text{boost}}, \delta_{\text{boost}})$-differentially private, where

$$\begin{aligned}\varepsilon_{\text{boost}} &= T\varepsilon_{\text{base}} + \varepsilon_{\text{sample}}\\ \delta_{\text{boost}} &= T\delta_{\text{base}} + \delta_{\text{sample}}\end{aligned}$$

To get the parameters claimed in the statement of the theorem, we can take:

$$\mu \in O((T^{3/2}\sqrt{k}\sqrt{\log(1/\beta)}\alpha\rho)/\varepsilon_{\text{sample}}). \tag{6.8}$$

□

6.2 Base synopsis generators

Algorithm SmallDB (Section 4) is based on the insight that a small randomly selected subset of database rows provides good answers to large sets of fractional counting queries. The base synopsis generators described in the current section have an analogous insight: a small synopsis that gives good approximations to the answers to a small subset of queries also yields good approximations to most queries. Both of these are instances of *generalization bounds*. In the remainder of this section we firs prove a generalization bound and then use it to construct differentially base synopsis generators.

6.2.1 A generalization bound

We have a distribution $\mathcal{D}$ over a large set $\mathcal{Q}$ of queries to be approximated. The lemma below says that a sufficiently small synopsis that gives sufficiently good approximations to the answers of a *randomly selected* subset $S \subset \mathcal{Q}$ of queries, sampled according to the distribution $\mathcal{D}$ on $\mathcal{Q}$, will, with high probability over the choice of S, also give good approximations to the answers to *most* queries in $\mathcal{Q}$ (that is, to most of the mass of $\mathcal{Q}$, weighted by D). Of course, to make any sense the synopsis must include a method of providing an answer to all queries in $\mathcal{Q}$, not just the subset $S \subseteq \mathcal{Q}$ received as input. Our particular generators, described in Sections 6.2.2 and Theorem 6.6 will produce synthetic databases; to answer any query one can simply apply the query to the synthetic database, but the lemma will be stated in full generality.

Let $R(y, q)$ denote the answer given by the synopsis y (when used as input for the reconstruction procedure) on query q. A synopsis y λ-*fits* a database x w.r.t a set S of queries if $\max_{q\in S} |R(y, q) - q(x)| \leq \lambda$. Let $|y|$

denote the number of bits needed to represent y. Since our synopses will be synthetic databases, $|y| = N \log_2 |\mathcal{X}|$ for some appropriately chosen number N of universe elements. The generalization bound shows that if y λ-fits x with respect to a large enough (larger than $|y|$) randomly chosen set S of queries sampled from a distribution $\mathcal{D}$, then with high probability y λ-fits x for *most* of the mass of $\mathcal{D}$.

Lemma 6.5. Let $\mathcal{D}$ be an arbitrary distribution on a query set $\mathcal{Q} = \{q : \mathcal{X}^* \to \mathbb{R}\}$. For all $m \in \mathcal{N}$, $\gamma \in (0,1)$, $\eta \in [0, 1/2)$, let $a = 2(\log(1/\gamma) + m)/(m(1-2\eta))$. Then with probability at least $1-\gamma$ over the choice of $S \sim \mathcal{D}^{a \cdot m}$, every synopsis y of size at most m bits that λ-fits x with respect to the query set S, also λ-fits x with respect to at least a $(1/2 + \eta)$-fraction of $\mathcal{D}$.

Before proving the lemma we observe that a is a compression factor: we are squeezing the answers to am queries into an m-bit output, so larger a corresponds to more compression. Typically, this means better generalization, and indeed we see that if a is larger then, keeping m and γ fixed, we would be able to have larger η. The lemma also says that, for any given output size m, the number of queries needed as input to obtain an output that does well on a majority ($1/2 + \eta$ fraction) of $\mathcal{D}$ is only $O(\log(1/\gamma) + m)$. This is interesting because a smaller number of queries k needed by the base generator leads, via the privacy loss $\varepsilon_{\text{sample}}$ due to sampling of kT queries and its inverse relationship to the slackness μ (Equation 6.7), to improved accuracy of the output of the boosting algorithm.

Proof of Lemma 6.5. Fix a set of queries $S \subset \mathcal{Q}$ chosen independently according to $\mathcal{D}^{a \cdot m}$. Examine an arbitrary m-bit synopsis y. Note that y is described by an m-bit string. Let us say y is *bad* if $|R(y,q) - q(x)| > \lambda$ for at least a $(\log(1/\gamma) + m)/(a \cdot m)$ fraction of $\mathcal{D}$, meaning that $\Pr_{q \sim \mathcal{D}}[|R(y,q) - q(x)| > \lambda] \geq (\log(1/\gamma) + m)/(a \cdot m)$.

In other words, y is bad if there exists a set $Q_y \subset \mathcal{Q}$ of fractional weight at least $(\log(1/\gamma) + m)/(a \cdot m)$ such that $|R(y,q) - q(x)| > \lambda$ for $q \in Q_y$. For such a y, what is the probability that y gives λ-accurate answers for *every* $q \in S$? This is exactly the probability that none of

the queries in S is in Q_y, or

$$(1 - (\log(1/\gamma) + m)/(a \cdot m))^{a \cdot m} \leq e^{-(\log(1/\gamma)+m)} \leq \gamma \cdot 2^{-m}$$

Taking a union bound over all 2^m possible choices for y, the probability that there exists an m-bit synopsis y that is accurate on all the queries in S but inaccurate on a set of fractional weight $(\log(1/\beta) + m)/(a \cdot m)$ is at most γ. Letting $k = am = |S|$ we see that it is sufficient to have

$$a > \frac{2(\log(1/\gamma) + m)}{m \cdot (1 - 2\eta)}. \tag{6.9}$$

□

This simple lemma is extremely powerful. It tells us that when constructing a base generator at round t, we only need to worry about ensuring good answers for the small set of random queries sampled from $\mathcal{D}_t$; doing well for most of $\mathcal{D}_t$ will happen automatically!

6.2.2 The base generator

Our first generator works by brute force. After sampling a set S of k queries independently according to a distribution $\mathcal{D}$, the base generator will produce noisy answers for all queries in S via the Laplace mechanism. Then, making no further use of the actual database, the algorithm searches for *any* database of size n for which these noisy answers are sufficiently close, and outputs this database. Privacy will be immediate because everything after the k invocations of the Laplace mechanism is in post-processing. Thus the only source of privacy loss is the cumulative loss from these k invocations of the Laplace mechanism, which we know how to analyze via the composition theorem. Utility will follow from the utility of the Laplace mechanism — which says that we are unlikely to have "very large" error on even one query — coupled with the fact that the true database x is an n-element database that fits these noisy responses.[1]

[1]This argument assumes the size n of the database is known. Alternatively we can include a noisy query of the form "How many rows are in the database?" and exhaustively search all databases of size close to the response to this query.

Theorem 6.6 (Base Synopsis Generator for Arbitrary Queries)**.** For any data universe $\mathcal{X}$, database size n, and class $\mathcal{Q} : \{\mathcal{X}^* \to \mathbb{R}\}$ of queries of sensitivity at most ρ, for any $\varepsilon_{\text{base}}, \delta_{\text{base}} > 0$, there exists an $(\varepsilon_{\text{base}}, \delta_{\text{base}})$-differentially private $(k, \lambda, \eta = 1/3, \beta)$-base synopsis generator for $\mathcal{Q}$, where $k = am > 6(m + \log(2/\beta)) = 6(n \log |\mathcal{X}| + \log(2/\beta))$ and $\lambda > 2b(\log k + log(2/\beta))$, where $b = \rho\sqrt{am \log(1/\delta_{\text{base}})}/\varepsilon_{\text{base}}$.

The running time of the generator is

$$|\mathcal{X}|^n \cdot \text{poly}(n, \log(1/\beta), \log(1/\varepsilon_{\text{base}}), \log(1/\delta_{\text{base}})).$$

Proof. We first describe the base generator at a high level, then determine the values for k and λ. The synopsis y produced by the base generator will be a synthetic database of size n. Thus $m = |y| = n \cdot \log |\mathcal{X}|$. The generator begins by choosing a set S of k queries, sampled independently according to $\mathcal{D}$. It computes a noisy answer for each query $q \in S$ using the Laplace mechanism, adding to each true answer an independent draw from $\text{Lap}(b)$ for an appropriate b to be determined later. Let $\{\widehat{q(x)}\}_{q \in \mathcal{Q}}$ be the collection of noisy answers. The generator enumerates over all $|\mathcal{X}|^n$ databases of size n, and outputs the lexicographically first database y such that for every $q \in S$ we have $|q(y) - \widehat{q(x)}| \leq \lambda/2$. If no such database is found, it outputs $\perp$ instead, and we say it *fails*. Note that if $|\widehat{q(x)} - q(x)| < \lambda/2$ and $|q(y) - \widehat{q(x)}| < \lambda/2$, then $|q(y) - q(x)| < \lambda$.

There are two potential sources of failure for our particular generator. One possibility is that y fails to generalize, or is *bad* as defined in the proof of Lemma 6.5. A second possibility is that one of the samples from the Laplace distribution is of excessively large magnitude, which might cause the generator to fail. We will choose our parameters so as to bound the probability of each of these events individually by at most $\beta/2$.

Substituting $\eta = 1/3$ and $m = n \log |X|$ into Equation 6.9 shows that taking $a > 6(1 + \log(2/\beta)/m)$ suffices in order for the probability of failure due to the choice of S to be bounded by $\beta/2$. Thus, taking $k = am > 6(m + \log(2/\beta)) = 6(n \log |\mathcal{X}| + \log(2/\beta))$ suffices.

We have k queries of sensitivity at most ρ. Using the Laplace mechanism with parameter $b = 2\sqrt{2k \log(1/\delta_{\text{base}})}\rho/\varepsilon_{\text{base}}$, ensures that each query incurs privacy loss at most $\varepsilon_{\text{base}}/\sqrt{2k \ln(1/\delta_{\text{base}})}$, which by

Corollary 3.21 ensures that the entire procedure will be $(\varepsilon_{\text{base}}, \delta_{\text{base}})$-differentially private.

We will choose λ so that the probability that any draw from $\text{Lap}(b)$ has magnitude exceeding $\lambda/2$ is at most $\beta/2$. Conditioned on the event that all k draws have magnitude at most λ we know that the input database itself will λ-fit our noisy answers, so the procedure will not fail.

Recall that the concentration properties of the Laplace distribution ensure that with probability at least $1-e^t$ a draw from $\text{Lap}(b)$ will have magnitude bounded by tb. Setting $\lambda/2 = tb$, the probability that a given draw will have magnitude exceeding $\lambda/2$ is bounded by $e^{-t} = e^{-\lambda/2b}$. To ensure that none of the k draws has magnitude exceeding $\lambda/2$ it suffices, by a union bound, to have

$$\begin{aligned} & ke^{-\lambda/2b} < \beta/2 \\ \Leftrightarrow\ & e^{\lambda/2b} > k\frac{2}{\beta} \\ \Leftrightarrow\ & \lambda/2 > b(\log k + \log(2/\beta)) \\ \Leftrightarrow\ & \lambda > 2b(\log k + \log(2/\beta)). \end{aligned}$$

□

The Special Case of Linear Queries. For the special case of linear queries it is possible to avoid the brute force search for a small database. The technique requires time that is polynomial in $(|\mathcal{Q}|, |\mathcal{X}|, n, \log(1/\beta))$. We will focus on the case of counting queries and sketch the construction.

As in the case of the base generator for arbitrary queries, the base generator begins by selecting a set S of $k = am$ queries according to $\mathcal{D}$ and computing noisy answers using Laplace noise. The generator for linear queries then runs a *syntheticizer* on S which, roughly speaking, transforms any synopsis giving good approximations to *any* set R of queries into a synthetic database yielding approximations of similar quality on the set R. The input to the syntheticizer will be the noisy values for the queries in S, that is, $R = S$. (Recall that when we modify the size of the database we always think in terms of the fractional version of the counting queries: "What fraction of the database rows satisfies property P?")

The resulting database may be quite large, meaning it may have many rows. The base generator then subsamples only $n' = (\log k \log(1/\beta))/\alpha^2$ of the rows of the synthetic database, creating a smaller synthetic database that with probability at least $1 - \beta$ has α-accuracy with respect to the answers given by the large synthetic database. This yields an $m = ((\log k \log(1/\beta))/\alpha^2) \log |\mathcal{X}|$-bit synopsis that, by the generalization lemma, with probability $(1 - \log(1/\beta))$ over the choice of the k queries, answers well on a $(1/2 + \eta)$ fraction of $\mathcal{Q}$ (as weighted by $\mathcal{D}$).

As in the case of the base generator for arbitrary queries, we require $k = am > 6 \log(1/\beta) + 6m$. Taking $\alpha^2 = (\log \mathcal{Q})/n$ we get that

$$\begin{aligned} k &> 6 \log(1/\beta) + 6 \frac{\log k \log(1/\beta) \log |\mathcal{X}|}{\alpha^2} \\ &= 6 \log(1/\beta) + 6n \log k \log(1/\beta) \frac{\log |\mathcal{X}|}{\log |\mathcal{Q}|}. \end{aligned}$$

The syntheticizer is nontrivial. Its properties are summarized by the following theorem.

Theorem 6.7. Let $\mathcal{X}$ be a data universe, $\mathcal{Q}$ a set of fractional counting queries, and A an (ε, δ)-differentially private synopsis generator with utility $(\alpha, \beta, 0)$ and arbitrary output. Then there exists a syntheticizer A' that is (ε, δ)-differentially private and has utility $(3\alpha, \beta, 0)$. A' outputs a (potentially large) synthetic database. Its running time is polynomial in the running time of A and $(|\mathcal{X}|, |\mathcal{Q}|, 1/\alpha, \log(1/\beta))$.

In our case, A is the Laplace mechanism, and the synopsis is simply the set of noisy answers. The composition theorem says that for A to be $(\varepsilon_{\text{base}}, \delta_{\text{base}})$-differentially private the parameter to the Laplace mechanism should be $\rho/(\varepsilon_{\text{base}}/\sqrt{2k \log(1/\delta_{\text{base}})})$. For fractional counting queries the sensitivity is $\rho = 1/n$.

Thus, when we apply the Theorem we will have an α of order $(\sqrt{k \log(1/\beta)}/\varepsilon_{\text{base}})\rho$. Here, ρ is the sensitivity. For counting queries it is 1, but we will shift to fractional counting queries, so $\rho = 1/n$.

Proof Sketch for Theorem 6.7. Run A to get (differentially private) (fractional) counts on all the queries in R. We will then use linear programming to find a low-weight fractional database that approximates

these fractional counts, as explained below. Finally, we transform this fractional database into a standard synthetic database by rounding the fractional counts.

The output of A yields a fractional count for each query $q \in \mathcal{Q}$. 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_q is the fractional count that A's output gives on query q. With probability $1 - \beta$, all of the entries in v are α-accurate.

A "fractional" database z that approximates these counts is obtained as follows. Recall the histogram representation of a database, where for each element in the universe $\mathcal{X}$ the histogram contains the number of instances of this element in the database. Now, for every $i \in \mathcal{X}$, we introduce a variable $a_i \geq 0$ that will "count" the (fractional) number of occurrences of i in the fractional database z. We will impose the constraint

$$\sum_{i \in \mathcal{X}} a_i = 1.$$

We represent the count of query q in z as the sum of the count of items i that satisfy q:

$$\sum_{i \in \mathcal{X} \text{ s.t. } q(i)=1} a_i$$

We want all of these counts to be within a an additive α accuracy of the respective counts in v_q. Writing this as a linear inequality we get:

$$(v_q - \alpha) \sum_{i \in \mathcal{X}} a_i \leq \sum_{i \in \mathcal{X} \text{ s.t. } q(i)=1} a_i \leq (v_q + \alpha) \sum_{i \in \mathcal{X}} a_i.$$

When the counts are all α-accurate with respect to the counts in v_c, it is also the case that (with probability $1 - \beta$) they are all 2α-accurate with respect to the true counts on the original database x.

We write a linear program with two such constraints for each query (a total of $2|\mathcal{Q}|$ 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 round down each a_i, for $i \in \mathcal{X}$, to the closest multiple of $\alpha/|\mathcal{X}|$, this changes each fractional count by at most a $\alpha/|\mathcal{X}|$ 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 $|\mathcal{X}|/\alpha$. □

Recall that in our application of Theorem 6.7 we defined A to be the mechanism that adds Laplace noise with parameter $\rho/(\varepsilon_{\text{base}}/\sqrt{2k\log(1/\delta_{\text{base}})})$. We have k draws, so by taking

$$\alpha' = \rho\sqrt{2k\log(1/\delta_{\text{base}})}(\log k + \log(1/\beta))$$

we have that A is $(\alpha', \beta, 0)$-accurate. For the base generator we chose error $\alpha^2 = (\log|\mathcal{Q}|)/n$. If the output of the syntheticizer is too large, we subsample

$$n' = \frac{\log|\mathcal{Q}|\log(1/\beta)}{\alpha^2} = \frac{\log k \log(1/\beta)}{\alpha^2}$$

rows. With probability $1-\beta$ the resulting database maintains $O(\rho\sqrt{(\log|\mathcal{Q}|)/n} + (\sqrt{2k\log(1/\delta_{\text{base}})}/\varepsilon_{\text{base}})(\log k + \log(1/\beta))$-accuracy on all of the concepts simultaneously.

Finally, the base generator can fail if the choice of queries $S \in \mathcal{D}^k$ does not lead to good generalization. With the parameters we have chosen this occurs with probability at most β, leading to a total failure probability of the entire generator of 3β.

Theorem 6.8 (Base Generator for Fractional Linear Queries)**.** For any data universe $\mathcal{X}$, database size n, and class $\mathcal{Q} : \{\mathcal{X}^n \to \mathbb{R}\}$ of fractional linear queries (with sensitivity at most $1/n$), for any $\varepsilon_{\text{base}}, \delta_{\text{base}} > 0$, there exists an $(\varepsilon_{\text{base}}, \delta_{\text{base}})$-differentially private $(k, \lambda, 1/3, 3\beta)$-base synopsis generator for $\mathcal{Q}$, where

$$k = O\left(\frac{n\log(|\mathcal{X}|)\log(1/\beta)}{\log|\mathcal{Q}|}\right)$$

$$\lambda = O\left(\frac{\log(1/\beta)}{\sqrt{n}}\left(\sqrt{\log|\mathcal{Q}|} + \sqrt{\frac{log|\mathcal{X}|}{log|\mathcal{Q}|}}\cdot\frac{1}{\varepsilon_{\text{base}}}\right)\right).$$

The running time of the base generator is $\text{poly}(|\mathcal{X}|, n, \log(1/\beta), \log(1/\varepsilon_{\text{base}}))$.

The sampling bound used here is the same as that used in the construction of the SmallDB mechanism, but with different parameters. Here we are using these bounds for a base generator in a complicated boosting algorithm with a very small query set; there we are using them for a single-shot generation of a synthetic database with an enormous query set.

6.2.3 Assembling the ingredients

The total error comes from the choice of μ (see Equation 6.2) and λ, the accuracy parameter for the based generator.

Let us recall Theorem 6.1:

Theorem 6.9 (Theorem 6.1)**.** Let $\mathcal{Q}$ be a query family with sensitivity at most ρ. For an appropriate setting of parameters, and with $T = \log|\mathcal{Q}|/\eta^2$ rounds, the algorithm of Figure 6.1 is an accurate and differentially private query-boosting algorithm:

1. When instantiated with a $(k, \lambda, \eta, \beta)$-base synopsis generator, the output of the boosting algorithm gives $(\lambda + \mu)$-accurate answers to *all* the queries in $\mathcal{Q}$ with probability at least $1 - T\beta$, where
$$\mu \in O(((\log^{3/2}|Q|)\sqrt{k}\sqrt{\log(1/\beta)}\rho)/(\varepsilon_{\text{sample}} \cdot \eta^3)). \quad (6.10)$$
2. If the base synopsis generator is $(\varepsilon_{\text{base}}, \delta_{\text{base}})$-differentially private, then the boosting algorithm is $((\varepsilon_{\text{sample}} + T \cdot \varepsilon_{\text{base}}), T(\beta + \delta_{\text{base}}))$-differentially private.

By Equation 6.7,
$$\varepsilon_{\text{sample}} \stackrel{\text{def}}{=} \sqrt{2kT\log(1/\beta)}(\alpha 4T\rho/\mu) + kT\left(\frac{\alpha 4T\rho}{\mu}\right)^2,$$
where $\alpha = (1/2)(\ln(1+2\eta)(1-2\eta)) \in O(1)$. We always have $T = (\log|\mathcal{Q}|)/\eta^2$, so substituting in this value into the above equation we see that the bound
$$\mu \in O(((\log^{3/2}|Q|)\sqrt{k}\sqrt{\log(1/\beta)}\rho)/(\varepsilon_{\text{sample}} \cdot \eta^3))$$
in the statement of the theorem is acceptable.

For the case of arbitrary queries, with η a constant, we have

$$\lambda \in O\left(\frac{\rho}{\varepsilon_{\text{base}}}(\sqrt{n \log |\mathcal{X}| \log(1/\delta_{\text{base}})}(\log(n \log |\mathcal{X}|) + \log(2/\beta)))\right).$$

Now, $\varepsilon_{\text{boost}} = T\varepsilon_{\text{base}} + \varepsilon_{\text{sample}}$. Set these two terms equal, so $T\varepsilon_{\text{base}} = \varepsilon_{\text{boost}}/2 = \varepsilon_{\text{sample}}$, whence we can replace the $1/\varepsilon_{\text{base}}$ term with $2T/\varepsilon_{\text{boost}} = (\log |\mathcal{Q}|/\eta^2)/2\varepsilon_{\text{boost}}$. Now our terms for λ and μ have similar denominators, since η is constant. We may therefore conclude that the total error is bounded by:

$$\lambda + \mu \in \tilde{O}\left(\frac{\sqrt{n \log |\mathcal{X}|}\rho \log^{3/2} |\mathcal{Q}|(\log(1/\beta))^{3/2}}{\varepsilon_{\text{boost}}}\right).$$

With similar reasoning, for the case of fractional counting queries we get

$$\lambda + \mu \in \tilde{O}\left(\frac{\sqrt{\log |\mathcal{X}|} \log |\mathcal{Q}| \log(1/\beta)^{3/2}}{\varepsilon_{\text{boost}}\sqrt{n}}\right).$$

To convert to a bound for ordinary, non-fractional, counting queries we multiply by n to obtain

$$\lambda + \mu \in \tilde{O}\left(\frac{\sqrt{n \log |\mathcal{X}|} \log |\mathcal{Q}| \log(1/\beta)^{3/2}}{\varepsilon_{\text{boost}}}\right).$$

6.3 Bibliographical notes

The boosting algorithm (Figure 6.1) is a variant of AdaBoost algorithm of Schapire and Singer [78]. See Schapire [77] for an excellent survey of boosting, and the textbook "Boosting" by Freund and Schapire [79] for a thorough treatment. The private boosting algorithm covered in this section is due to Dwork et al. [32], which also contains the base generator for linear queries. This base generator, in turn, relies on the syntheticizer of Dwork et al. [28]. In particular, Theorem 6.7 comes from [28]. Dwork, Rothblum, and Vadhan also addressed differentially private boosting in the usual sense.

7

When Worst-Case Sensitivity is Atypical

In this section, we briefly describe two general techniques, both enjoying unconditional privacy guarantees, that can often make life easier for the data analyst, especially when dealing with a function that has arbitrary, or difficult to analyze, worst-case sensitivity. These algorithms are most useful in computing functions that, for some exogenous reason, the analyst has reason to believe are "usually" insensitive in practice.

7.1 Subsample and aggregate

The Subsample and Aggregate technique yields a method for "forcing" the computation of a function $f(x)$ to be insensitive, even for an *arbitrary* function f. Proving privacy will be trivial. Accuracy depends on properties of the function f and the specific data set x; in particular, if $f(x)$ can be accurately estimated, with high probability, on $f(S)$, where S is a random subset of the elements in x, then accuracy should be good. Many maximum likelihood statistical estimators enjoy this property on "typical" data sets — this is why these estimators are employed in practice.

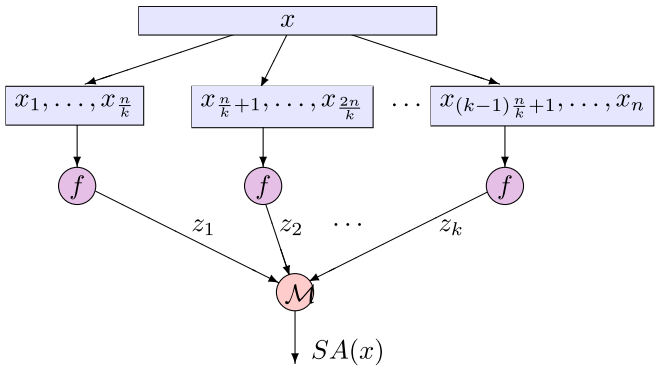

Figure 7.1: Subsample and Aggregate with a generic differentially private aggregation algorithm $\mathcal{M}$.

In Subsample and Aggregate, the n rows of the database x are randomly partitioned into m blocks $B_1, \ldots, B_m$, each of size n/m. The function f is computed *exactly, without noise*, independently on each block. The intermediate outcomes $f(B_1), \ldots, f(B_m)$ are then combined via a differentially private aggregation mechanism — typical examples include standard aggregations, such as the α-trimmed mean,[1] the Winsorized mean,[2] and the median, but there are no restrictions — and then adding Laplace noise scaled to the sensitivity of the aggregation function in question; see Figure 7.1.

The key observation in Subsample and Aggregate is that any single element can affect at most one block, and therefore the value of just a single $f(B_i)$. Thus, changing the data of any individual can change at most a single input to the aggregation function. Even if f is arbitrary, the analyst chooses the aggregation function, and so is free to choose one that is insensitive, *provided that choice is independent of the database!* Privacy is therefore immediate: For any $\delta \geq 0$ and any function f, if the aggregation mechanism $\mathcal{M}$ is (ε, δ)-differentially private

[1]The α-trimmed mean is the mean after the top and bottom α fraction of the inputs have been discarded.

[2]The Winsorized mean is similar to the α-trimmed mean except that, rather than being discarded, the top and bottom α fraction are replaced with the most extreme remaining values.

then so is the Subsample and Aggregate technique when instantiated with f and $\mathcal{M}$.[3]

Utility is a different story, and it is frustratingly difficult to argue even for the case in which data are plentiful and large random subsets are very likely to give similar results. For example, the data may be labeled training points in high dimensional space and the function is logistic regression, which produces a vector v and labels a point p with $+1$ if and only if $p \cdot v \geq T$ for some (say, fixed) threshold T. Intuitively, if the samples are sufficiently plentiful and typical then all blocks should yield similar vectors v. The difficulty comes in getting a good bound on the worst-case sensitivity of the aggregation function — we may need to use the size of the range as a fallback. Nonetheless, some nice applications are known, especially in the realm of statistical estimators, where, for example, it can be shown that, under the assumption of "generic normality," privacy can be achieved at *no* additional cost in statistical efficiency (roughly, accuracy as the number of samples grows). We do not define generic normality here, but note that estimators fitting these assumptions include the maximum likelihood estimator for "nice" parametric families of distributions such as gaussians, and maximum-likelihood estimators for linear regression and logistic regression.

Suppose the function f has a *discrete* range of cardinality m, say, $[m]$. In this case Subsample and Aggregate will need to aggregate a set of b elements drawn from $[m]$, and we can use Report Noisy Arg-Max to find the most popular outcome. This approach to aggregation requires $b \geq \log m$ to obtain meaningful results even when the intermediate outcomes are unanimous. We will see an alternative below with no such requirement.

Example 7.1 (Choosing a Model)**.** Much work in statistics and machine learning addresses the problem of *model selection*: Given a data set and a discrete collection of "models," each of which is a family of probability distributions, the goal is to determine the model that best "fits"

[3]The choice of aggregation function can even depend on the database, but the selection must be made in a differentially private fashion. The privacy cost is then the cost of composing the choice operation with the aggregation function.

the data. For example, given a set of labeled d-dimensional data, the collection of models might be all subsets of at most $s \ll d$ features, and the goal is to find the set of features that best permits prediction of the labels. The function f might be choosing the best model from the given set of of m models, a process known as *model fitting*, via an arbitrary learning algorithm. Aggregation to find the most popular value could be done via Report Noisy Max, which also yields an estimate of its popularity.

Example 7.2 (Significant Features)**.** This is a special case of model fitting. The data are a collection of points in $\mathbb{R}^d$ and the function is the very popular LASSO, which yields as output a list $L \in [d]^s$ of at most $s \ll d$ significant features. We can aggregate the output in two ways: feature by feature — equivalent to running d executions of Subsample and Aggregate, one for each feature, each with a range of size 2 — or on the set as a whole, in which case the cardinality of the range is $\binom{d}{s}$.

7.2 Propose-test-Release

At this point one might ask: what is the meaning of the aggregation if there is not substantial agreement among the blocks? More generally, for any reasonably large-scale statistical analysis in real life, we expect the results to be fairly stable, independent of the presence or absence of any single individual. Indeed, this is the entire intuition behind the significance of a statistic and underlying the utility of differential privacy. We can even go further, and argue that if a statistic is not stable, we should have no interest in computing it. Often, our database will in fact be a sample from a larger population, and our true goal is not to compute the value of the statistic on the database itself, but rather estimate it for the underlying population. Implicitly, therefore, when computing a statistic we are already assuming that the statistic is stable under subsampling!

Everything we have seen so far has provided privacy even on very "idiosyncratic" datasets, for which "typically" stable algorithms my be highly unstable. In this section we introduce a methodology, Propose-Test-Release, which is motivated by the philosophy that if there is

insufficient stability then the analysis can be abandoned because the results are not in fact meaningful. That is, the methodology allows the analyst to check that, *on the given dataset*, the function satisfies some "robustness" or "stability" criterion and, if it does not, to halt the analysis.

The goal of our first application of Propose-Test-Release is to come up with a variant of the Laplace mechanism that adds noise scaled to something strictly smaller than the sensitivity of a function. This leads to the notion of *local sensitivity*, which is defined for a (function, database) pair, say, (f, x). Quite simply, the local sensitivity of f with respect to x is the amount by which the $f(y)$ can differ from $f(x)$ for any y adjacent to x.

Definition 7.1 (Local Sensitivity)**.** The local sensitivity of a function $f : \mathcal{X}^n \to \mathbb{R}^k$ with respect to a database x is:

$$\max_{y \text{ adjacent to } x} \|f(x) - f(y)\|_1.$$

The Propose-Test-Release approach is to first *propose* a bound, say b, on local sensitivity — typically the data analyst has some idea of what this should be — and then run a differentially private *test* to ensure that the database is "far" from any database for which this bound fails to hold. If the test is passed, then the sensitivity is assumed to be bounded by b, and a differentially private mechanism such as, for example, the Laplace mechanism with parameter b/ϵ, is used to *release* the (slightly) noisy response to the query.

Note that we can view this approach as a two-party algorithm where one party plays an honest data analyst and the other is the Laplace mechanism. There is an interplay between the honest analyst and the mechanism in which the algorithm asks for an estimate of the sensitivity and then "instructs" the mechanism to use this estimated sensitivity in responding to subsequent queries. Why does it need to be so complicated? Why can't the mechanism simply add noise scaled to the local sensitivity without playing this private estimation game? The reason is that the local sensitivity may *itself* be sensitive. This fact, combined with some auxiliary information about the database, can lead to privacy problems: the adversary may know that the database is one of x,

which has very low local sensitivity for the computation in question, and a neighboring y, for which the function has very high local sensitivity. In this case the adversary may be able to guess rather accurately which of x and y is the true database. For example, if $f(x) = f(y) = s$ and the reponse is far from s, then the adversary would guess y.

This is captured by the math of differential privacy. There are neighboring instances of the median function which have the same median, say, m, but arbitrarily large gaps in the local sensitivity. Suppose the response R to the median query is computed via the Laplace mechanism with noise scaled to the local sensitivity. When the database is x the probability mass is close to m, because the sensitivity is small, but when the database is y the mass is far flung, because the sensitivity is large. As an extreme case, suppose the local sensitivity on x is exactly zero, for example, $\mathcal{X} = \{0, 10^6\}$, n is even, and x, which has size $n + 1$, contains $1 + n/2$ zeros. Then the median of x is zero and the local sensitivity of the median, when the database is x, is 0. In contrast, the neighboring database y has size n, contains $n/2$ zeros, has median zero (we have defined median to break ties in favor of the smaller value), and the local sensitivity of the median, when the database is y, is 10^6. On x all the mass of the Laplace mechanism (with parameter $0/\varepsilon = 0$) is concentrated on the single point 0; but on y the probability distribution has standard deviation $\sqrt{2} \cdot 10^6$. This destroys all hope of differential privacy.

To test that the database is "far" from one with local sensitivity greater than the proposed bound b, we may pose the query: "What is the distance of the true database to the closest one with local sensitivity exceeding b?" Distance to a fixed set of databases is a (global) sensitivity 1 query, so this test can be run in a differentially private fashion by adding noise $\mathrm{Lap}(1/\varepsilon)$ to the true answer. To err on the side of privacy, the algorithm can compare this noisy distance to a conservative threshold — one that is only negligibly likely to be exceeded due to a freak event of very large magnitude Laplace noise. For example, if the threshold used is, say, $\ln^2 n$, the probability of a false positive (passing the test when the local sensitivity in fact exceeds b) is at most $O(n^{-\varepsilon \ln n})$, by the properties of the Laplace distribution. Because of the negligible probability of a false positive, the technique cannot yield $(\varepsilon, 0)$-differential privacy for any ε.

To apply this methodology to consensus on blocks, as in our discussion of Subsample and Aggregate, view the intermediate results $f(B_1), \ldots, f(B_m)$ as a data set and consider some measure of the concentration of these values. Intuitively, if the values are tightly concentrated then we have consensus among the blocks. Of course, we still need to find the correct notion of concentration, one that is meaningful and that has a differentially private instantiation. In a later section we will define and weave together two notions of stability that seem relevant to Subsample and Aggregate: insensitivity (to the removal or addition of a few data points) and stability under subsampling, capturing the notion that a subsample should yield similar results to the full data set.

7.2.1 Example: the scale of a dataset

Given a dataset, a natural question to ask is, "What is the scale, or dispersion, of the dataset?" This is a different question from *data location*, which might be captured by the median or the mean. The data scale is more often captured by the variance or an interquantile range. We will focus on the *interquartile range (IQR)*, a well-known robust estimator for the scale of the data. We begin with some rough intuition. Suppose the data are *i.i.d.* samples drawn from a distribution with cumulative distribution function F. Then $\mathrm{IQR}(F)$, defined as $F^{-1}(3/4) - F^{-1}(1/4)$, is a constant, depending only on F. It might be very large, or very tiny, but either way, if the density of F is sufficiently high at the two quartiles, then, given enough samples from F, the empirical (that is, sample) interquartile distance should be close to $\mathrm{IQR}(F)$.

Our Propose-Test-Release algorithm for the interquartile distance first tests how many database points need to be changed to obtain a data set with a "sufficiently different" interquartile distance. Only if the (noisy) reply is "sufficiently large" will the algorithm release an approximation to the interquartile range of the dataset.

The definition of "sufficiently different" is multiplicative, as an additive notion for difference of scale makes no sense — what would be the right scale for the additive amount? The algorithm therefore works with the logarithm of the scale, which leads to a multiplicative noise

on the IQR. To see this, suppose that, as in what might be the typical case, the sample interquartile distance cannot change by a factor of 2 by modifying a single point. Then the logarithm (base 2) of the sample interquartile has local sensitivity bounded by 1. This lets us privately release an approximation to *the logarithm of* the sample interquartile range by adding to this value a random draw from $\mathrm{Lap}(1/\varepsilon)$.

Let $\mathrm{IQR}(x)$ denote the sample interquartile range when the the data set is x. The algorithm is (implicitly) *proposing* to add noise drawn from $\mathrm{Lap}(1/\varepsilon)$ to the value $\log_2(\mathrm{IQR}(x))$. To *test* whether this magnitude of noise is sufficient for differential privacy, we discretize $\mathbb{R}$ into disjoint bins $\{[k \ln 2, (k+1) \ln 2)\}_{k \in \mathbf{Z}}$ and ask how many data points must be modified in order to obtain a new database, the logarithm (base 2) of whose interquartile range is in a different bin than that of $\log_2(\mathrm{IQR}(x))$. If the answer is at least two then the local sensitivity (of the logarithm of the interquartile range) is bounded by the bin width. We now give more details.

To understand the choice of bin size, we write

$$\log_2(\mathrm{IQR}(x)) = \frac{\ln \mathrm{IQR}(x)}{\ln 2} = \frac{c \ln 2}{\ln 2},$$

whence we find that looking at $\ln(\mathrm{IQR}(x))$ on the scale of $\ln 2$ is equivalent to looking at $\log_2(\mathrm{IQR}(x))$ on the scale of 1. Thus we have scaled bins which are intervals whose endpoints are a pair of adjacent integers: $B_k = [k, k+1)$, $k \in \mathbf{Z}$, and we let $k_1 = \lfloor \log_2(\mathrm{IQR}(x)) \rfloor$, so $\log_2(\mathrm{IQR}(x)) \in [k_1, k_1 + 1)$ and we say informally that the logarithm of the IQR is in bin k_1. Consider the following testing query:

> **$\mathbf{Q_0}$** : How many data points need to change in order to get a new database z such that $\log_2(\mathrm{IQR}(z)) \notin B_{k_1}$?

Let $A_0(x)$ be the true answer to $\mathbf{Q_0}$ when the database is x. If $A_0(x) \geq 2$, then neighbors y of x satisfy $|\log_2(\mathrm{IQR}(y)) - \log_2(\mathrm{IQR}(x))| \leq 1$. That is, they are close to each other. This is not equivalent to being in the same interval in the discretization: $\log_2(\mathrm{IQR}(x))$ may lie close to one of the endpoints of the interval $[k_1, k_1 + 1)$ and $\log_2(\mathrm{IQR}(y))$ may lie just on the other side of the endpoint. Letting $R_0 = A_0(x) + \mathrm{Lap}(1/\varepsilon)$, a small R_0, even when the

draw from the Laplace distribution has small magnitude, might not actually indicate high sensitivity of the interquartile range. To cope with the case that the local sensitivity is very small, but $\log_2(\mathrm{IQR}(x))$ is very close to the boundary, we consider a second discretization $\{B_k^{(2)} = [k-0.5, k+0.5)\}_{k\in\mathbf{Z}}$. We denote the two discretizations by $B^{(1)}$ and $B^{(2)}$ respectively. The value $\log_2(\mathrm{IQR}(x))$ — indeed, any value — cannot be close to a boundary in both discretizations. The test is passed if R_0 is large in at least one discretization.

The **Scale** algorithm (Algorithm 12) below for computing database scale assumes that n, the size of the database, is known, and the distance query ("How far to a database whose interquartile range has sensitivity exceeding b?") is asking how many points must be *moved* to reach a database with high sensitivity of the IQR. We can avoid this assumption by having the algorithm first ask the (sensitivity 1) query: "How many data points are in x?" We remark that, for technical reasons, to cope with the case $IQR(x) = 0$, we define $\log 0 = -\infty$, $\lfloor -\infty \rfloor = -\infty$, and let $[-\infty, -\infty) = \{-\infty\}$.

Algorithm 12 The **Scale** Algorithm (releasing the interquartile range)

Require: dataset: $x \in \mathcal{X}^*$, privacy parameters: $\epsilon, \delta > 0$

for the jth discretization ($j = 1, 2$) **do**
  Compute $R_0(x) = A_0(x) + z_0$, where $z_0 \in_R \mathrm{Lap}(1/\varepsilon)$.
  if $R_0 \leq 1 + \ln(1/\delta)$ **then**
    Let $s^{(j)} = \perp$.
  else
    Let $s^{(j)} = IQR(x) \times 2^{z_s^{(j)}}$, where $z_s^{(j)} \sim \mathrm{Lap}(1/\varepsilon)$.
  end if
end for
if $s^{(1)} \neq \perp$ **then**
  Return $s^{(1)}$.
else
  Return $s^{(2)}$.
end if

Note that the algorithm is efficient: let $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$ denote the n database points *after sorting*, and let $x(m)$ denote the median, so $m = \lfloor (n+1)/2 \rfloor$. Then the local sensitivity of the median is $\max\{x(m) - x(m-1), x(m+1) - x(m)\}$ and, more importantly, one can compute $A_0(x)$ by considering $O(n)$ sliding intervals with width 2^{k_1} and 2^{k_1+1}, each having one endpoint in x. The computational cost for each interval is constant.

We will not prove convergence bounds for this algorithm because, for the sake of simplicity, we have used a base for the logarithm that is far from optimal (a better base is $1 + 1/\ln n$). We briefly outline the steps in the proof of privacy.

Theorem 7.1. Algorithm **Scale** (Algorithm 12) is $(4\varepsilon, \delta)$-differentially private.

Proof. (Sketch.) Letting s be shorthand for the result obtained with a single discretization, and defining $\mathcal{D}_0 = \{x : A_0(x) \geq 2\}$, the proof shows:

1. The worst-case sensitivity of query $\mathbf{Q_0}$ is at most 1.
2. Neighboring databases are almost equally likely to result in $\perp$: For all neighboring database x, y:
$$\Pr[s = \perp | x] \leq e^{\varepsilon} \Pr[s = \perp | y].$$
3. Databases not in $\mathcal{D}_0$ are unlikely to pass the test:
$$\forall x \notin \mathcal{D}_0 : \Pr[s \neq \perp | x] \leq \frac{\delta}{2}.$$
4. $\forall C \in \mathbb{R}^+, x \in \mathcal{D}_0$ and all neighbors y of x:
$$\Pr[s \in C | x] \leq e^{2\varepsilon} \Pr[s \in C | y].$$

Thus, we get $(2\varepsilon, \delta/2)$-differential privacy for each discretization. Applying Theorem 3.16 (Appendix B), which says that "the epsilons and the deltas add up," yields $(4\varepsilon, \delta)$-differential privacy. □

7.3 Stability and privacy

7.3.1 Two notions of stability

We begin by making a distinction between the two notions of stability intertwined in this section: stability under subsampling, which yields similar results under random subsamples of the data, and perturbation stability, or low local sensitivity, for a given dataset. In this section we will define and make use of extreme versions of both of these.

- *Subsampling stability*: We say f is q-subsampling stable on x if $f(\hat{x}) = f(x)$ with probability at least $3/4$ when $\hat{x}$ is a random subsample from x which includes each entry independently with probability q. We will use this notion in Algorithm $\mathcal{A}_{\text{samp}}$, a variant of Sample and Aggregate.

- *Perturbation Stability*: We say that f is *stable* on x if f takes the value $f(x)$ on all of the neighbors of x (and *unstable* otherwise). In other words, f is stable on x if the local sensitivity of f on x is zero. We will use this notion (implemented in Algorithm $\mathcal{A}_{\text{dist}}$ below) for the aggregation step of $\mathcal{A}_{\text{samp}}$.

At the heart of Algorithm $\mathcal{A}_{\text{samp}}$ is a relaxed version of perturbation stability, where instead of requiring that the value be unchanged on neighboring databases — a notion that makes sense for arbitrary ranges, including arbitrary discrete ranges — we required only that the value be "close" on neighboring databases — a notion that requires a metric on the range.

Functions f with arbitrary ranges, and in particular the problem of aggregating outputs in Subsample and Aggregate, motivate the next algorithm, $\mathcal{A}_{\text{dist}}$. On input f, x, $\mathcal{A}_{\text{dist}}$ outputs $f(x)$ with high probability if x is at distance at least $\frac{2\log(1/\delta)}{\varepsilon}$ from the nearest *un*stable data set. The algorithm is conceptually trivial: compute the distance to the nearest unstable data set, add Laplace noise $\text{Lap}(1/\varepsilon)$, and check that this noisy distance is at least $\frac{2\log(1/\delta)}{\varepsilon}$. If so, release $f(x)$, otherwise output $\perp$. We now make this a little more formal.

We begin by defining a quantitative measure of perturbation stability.

Definition 7.2. A function $f : \mathcal{X}^* \to \mathcal{R}$ is *k-stable on input x* if adding or removing any k elements from x does not change the value of f, that is, $f(x) = f(y)$ for all y such that $|x \triangle y| \leq k$. We say f is *stable* on x if it is (at least) 1-stable on x, and *unstable* otherwise.

Definition 7.3. The *distance to instability* of a data set $x \in \mathcal{X}^*$ with respect to a function f is the number of elements that must be added to or removed from y to reach a data set that is not stable under f.

Note that f is k-stable on x if and only if the distance of x to instability is at least k.

Algorithm $\mathcal{A}_{\text{dist}}$, an instantiation of Propose-Test-Release for discrete-valued functions g, appears in Figure 13.

Algorithm 13 $\mathcal{A}_{\text{dist}}$ (releasing $g(x)$ based on distance to instability)

Require: dataset: $x \in \mathcal{X}^*$, privacy parameters: $\epsilon, \delta > 0$, function $g : \mathcal{X}^* \to \mathbb{R}$

$d \leftarrow$ distance from x to nearest unstable instance
$\hat{d} \leftarrow d + \text{Lap}(1/\varepsilon)$
if $\hat{d} > \frac{\log(1/\delta)}{\varepsilon}$ **then**
  Output $g(x)$
else
  Output $\perp$
end if

The proof of the following propostion is immediate from the properties of the Laplace distribution.

Proposition 7.2. For every function g:

1. $\mathcal{A}_{\text{dist}}$ is (ε, δ)-differentially private.
2. For all $\beta > 0$: if g is $\frac{\ln(1/\delta)+\ln(1/\beta)}{\varepsilon}$-stable on x, then $\mathcal{A}_{\text{dist}}(x) = g(x)$ with probability at least $1 - \beta$, where the probability space is the coin flips of $\mathcal{A}_{\text{dist}}$.

This distance-based result is the best possible, in the following sense: if there are two data sets x and y for which $\mathcal{A}_{\text{dist}}$ outputs different

values $g(x)$ and $g(y)$, respectively, with at least constant probability, then the distance from x to y must be $\Omega(\log(1/\delta)/\varepsilon)$.

Distance to instability can be difficult to compute, or even to lower bound, so this is not in general a practical solution. Two examples where distance to instability turns out to be easy to bound are the median and the mode (most frequently occurring value).

$\mathcal{A}_{\text{dist}}$ may also be unsatisfactory if the function, say f, is not stable on the specific datasets of interest. For example, suppose f is not stable because of the presence of a few outliers in x. Instances of the average behave this way, although for this function there are well know robust alternatives such as the winsorized mean, the trimmed mean, and the median. By what about for general functions f? Is there a method of "forcing" an arbitrary f to be stable on a database x?

This will be the goal of $\mathcal{A}_{\text{samp}}$, a variant of Subsample and Aggregate that outputs $f(x)$ with high probability (over its own random choices) whenever f is subsampling stable on x.

7.3.2 Algorithm $\mathcal{A}_{\text{samp}}$

In $\mathcal{A}_{\text{samp}}$, the blocks $B_1, \ldots, B_m$ are chosen *with replacement*, so that each block has the same distribution as the inputs (although now an element of x may appear in multiple blocks). We will call these subsampled datasets $\hat{x}_1, \ldots, \hat{x}_m$. The intermediate outputs $z = \{f(\hat{x}_1), \ldots, f(\hat{x}_m)\}$ are then aggregated via $\mathcal{A}_{\text{dist}}$ with function $g = \text{mode}$. The distance measure used to estimate the stability of the mode on z is a scaled version of the difference between the popularity of the mode and that of the second most frequent value. Algorithm $\mathcal{A}_{\text{samp}}$, appears in Figure 14. Its running time is dominated by running f about $1/q^2$ times; hence it is efficient whenever f is.

The key property of Algorithm $\mathcal{A}_{\text{samp}}$ is that, on input f, x, it outputs $f(x)$ with high probability, over its own random choices, whenever f is q-subsampling stable on x for $q = \frac{\varepsilon}{64\log(1/\delta)}$. This result has an important statistical interpretation. Recall the discussion of model selection from Example 7.1. Given a collection of models, the *sample complexity* of model selection is the number of samples from a distribution in one of the models necessary to select the correct model

with probability at least 2/3. The result says that *differentially private* model selection increases the sample complexity of (non-private) model selection by a problem-independent (and range-independent) factor of $O(\log(1/\delta)/\varepsilon)$.

Algorithm 14 $\mathcal{A}_{\text{samp}}$: Bootstrapping for Subsampling-Stable f

Require: dataset: x, function $f : \mathcal{X}^* \to \mathbb{R}$, privacy parameters $\epsilon, \delta > 0$.

$q \leftarrow \frac{\epsilon}{64\ln(1/\delta)}$, $m \leftarrow \frac{\log(n/\delta)}{q^2}$.
Subsample m data sets $\hat{x}_1, ..., \hat{x}_m$ from x, where $\hat{x}_i$ includes each position of x independently with probability q.
if some element of x appears in more than $2mq$ sets $\hat{x}_i$ **then**
  Halt and output $\perp$.
else
  $z \leftarrow \{f(\hat{x}_1), \cdots, f(\hat{x}_m)\}$.
  For each $r \in \mathbb{R}$, let $count(r) = \#\{i : f(\hat{x}_i) = r\}$.
  Let $count_{(i)}$ denote the the ith largest count, $i = 1, 2$.
  $d \leftarrow (count_{(1)} - count_{(2)})/(4mq) - 1$
  Comment Now run $\mathcal{A}_{\text{dist}}(g, z)$ using d to estimate distance to instability:
  $\hat{d} \leftarrow d + \text{Lap}(\frac{1}{\epsilon})$.
  if $\hat{d} > \ln(1/\delta)/\varepsilon$ **then**
    Output $g(z) = mode(z)$.
  else
    Output $\perp$.
  end if
end if

Theorem 7.3.

1. Algorithm $\mathcal{A}_{\text{samp}}$ is (ε, δ)-differentially private.
2. If f is q-subsampling stable on input x where $q = \frac{\varepsilon}{64\ln(1/\delta)}$, then algorithm $\mathcal{A}_{\text{samp}}(x)$ outputs $f(x)$ with probability at least $1 - 3\delta$.
3. If f can be computed in time $T(n)$ on inputs of length n, then $\mathcal{A}_{\text{samp}}$ runs in expected time $O(\frac{\log n}{q^2})(T(qn) + n)$.

Note that the utility statement here is an input-by-input guarantee; f need not be q-subsampling stable on all inputs. Importantly, there is no dependence on the size of the range $\mathcal{R}$. In the context of model selection, this means that one can efficiently satisfy differential privacy with a modest blowup in sample complexity (about $\log(1/\delta)/\varepsilon$) whenever there is a particular model that gets selected with reasonable probability.

The proof of privacy comes from the insensitivity of the computation of d, the privacy of the Propose-Test-Release technique, and the privacy of Subsample and Aggregate, modified slightly to allow for the fact that this algorithm performs sampling with replacement and thus the aggregator has higher sensitivity, since any individual might affect up to $2mq$ blocks. The main observation for analyzing the utility of this approach is that the stability of the *mode* is a function of the difference between the frequency of the mode and that of the next most popular element. The next lemma says that if f is subsampling stable on x, then x is far from unstable with respect to the mode $g(z) = g(f(\hat{x}_1), \ldots, f(\hat{x}_m))$ (but not necessarily with respect to f), and moreover one can estimate the distance to instability of x *efficiently* and privately.

Lemma 7.4. Fix $q \in (0,1)$. Given $f : \mathcal{X}^* \to \mathcal{R}$, let $\hat{f} : \mathcal{X}^* \to \mathcal{R}$ be the function $\hat{f} = \text{mode}(f(\hat{x}_1), ..., f(\hat{x}_m))$ where each $\hat{x}_i$ includes each element of x independently with probability q and $m = \ln(n/\delta)/q^2$. Let $d(z) = (\text{count}_{(1)} - \text{count}_{(2)})/(4mq) - 1$; that is, given a "database" z of values, $d(z)+1$ isa scaled difference between the number of occurrences of the two most popular values. Fix a data set x. Let E be the event that no position of x is included in more than $2mq$ of the subsets $\hat{x}_i$. Then, when $q \leq \varepsilon/64\ln(1/\delta)$ we have:

1. E occurs with probability at least $1 - \delta$.
2. Conditioned on E, d lower bounds the stability of $\hat{f}$ on x, and d has global sensitivity 1.
3. If f is q-subsampling stable on x, then with probability at least $1 - \delta$ over the choice of subsamples, we have $\hat{f}(x) = f(x)$, and, conditioned on this event, the final test will be passed with

probability at least $1 - \delta$, where the probability is over the draw from $\text{Lap}(1/\varepsilon)$.

The events in Parts 2 and 3 occur simultaenously with probability at least $1 - 2\delta$.

Proof. Part 1 follows from the Chernoff bound. To prove Part 2, notice that, conditioned on the event E, adding or removing one entry in the original data set changes any of the counts $count_{(r)}$ by at most $2mq$. Therefore, $count_{(1)} - count_{(2)}$ changes by at most $4mq$. This in turn means that $d(f(\hat{x}_1), \ldots, f(\hat{x}_m))$ changes by at most one for any x and hence has global sensitivity of one. This also implies that d lower bounds the stability of $\hat{f}$ on x.

We now turn to part 3. We want to argue two facts:

1. If f is q-subsampling stable on x, then there is likely to be a large gap between the counts of the two most popular bins. Specifically, we want to show that with high probability $count_{(1)} - count_{(2)} \geq m/4$. Note that if the most popular bin has count at least $5m/8$ then the second most popular bin can have count at most $3m/8$, with a difference of $m/4$. By definition of subsampling stability the most popular bin has an expected count of at least $3m/4$ and hence, by the Chernoff bound, taking $\alpha = 1/8$, has probability at most $e^{-2m\alpha^2} = e^{-m/32}$ of having a count less than $5m/8$. (All the probabilities are over the subsampling.)

2. When the gap between the counts of the two most popular bins is large, then the algorithm is unlikely to fail; that is, the test is likely to succeed. The worry is that the draw from $\text{Lap}(\frac{1}{\varepsilon})$ will be negative and have large absolute value, so that $\hat{d}$ falls below the threshold $(\ln(1/\delta)/\varepsilon)$ even when d is large. To ensure this happens with probability at most δ it suffices that $d > 2\ln(1/\delta)/\varepsilon$.
   By definition, $d = (count_{(1)} - count_{(2)})/(4mq) - 1$, and, assuming we are in the high probability case just described, this implies

$$d \geq \frac{m/4}{4mq} - 1 = \frac{1}{16q} - 1$$

so it is enough to have

$$\frac{1}{16q} > 2\ln(1/\delta)/\varepsilon.$$

Taking $q \leq \varepsilon/64\ln(1/\delta)$ suffices.

Finally, note that with these values of q and m we have $e^{-m/32} < \delta$. □

Example 7.3. [The Raw Data Problem] Suppose we have an analyst whom we can trust to follow instructions and only publish information obtained according to these instructions. Better yet, suppose we have b such analysts, and we can trust them not to communicate among themselves. The analysts do not need to be identical, but they do need to be considering a common set of *options*. For example, these options might be different statistics in a fixed set S of possible statistics, and in this first step the analyst's goal is to choose, for eventual publication, the most significant statistic in S. Later, the chosen statistic will be recomputed in a differentially private fashion, and the result can be published.

As described the procedure is not private at all: the *choice* of statistic made in the first step may depend on the data of a single individual! Nonetheless, we can use the Subsample-and-Aggregate framework to carry out the first step, with the ith analyst receiving a subsample of the data points and applying to this smaller database the function f_i to obtain an option. The options are then aggregated as in algorithm $\mathcal{A}_{\text{samp}}$; if there is a clear winner this is overwhelmingly likely to be the selected statistic. This was *chosen* in a differentially private manner, and in the second step it will be *computed* with differential privacy.

Bibliographic Notes

Subsample and Aggregate was invented by Nissim, Raskhodnikova, and Smith [68], who were the first to define and exploit low local sensitivity. Propose-Test-Release is due to Dwork and Lei [22], as is the algorithm for releasing the interquartile range. The discussion of stability and privacy, and Algorithm $\mathcal{A}_{samp}$ which blends these two techniques, is due to Smith and Thakurta [80]. This paper demonstrates the power of

$\mathcal{A}_{samp}$ by analyzing the subsampling stability conditions of the famous LASSO algorithm and showing that differential privacy can be obtained "for free," via (a generalization of $\mathcal{A}_{samp}$), precisely under the (fixed data as well as distributional) conditions for which LASSO is known to have good explanatory power.

8

Lower Bounds and Separation Results

In this section, we investigate various lower bounds and tradeoffs:

1. How *inaccurate* must responses be in order not to completely destroy any reasonable notion of privacy?
2. How does the answer to the previous question depend on the number of queries?
3. Can we separate $(\varepsilon, 0)$-differential privacy from (ε, δ)-differential privacy in terms of the accuracy each permits?
4. Is there an intrinsic difference between what can be achieved for linear queries and for arbitrary low-sensitivity queries while maintaining $(\varepsilon, 0)$-differential privacy?

A different flavor of separation result distinguishes the computational complexity of generating a *data structure* handling all the queries in a given class from that of generating a *synthetic database* that achieves the same goal. We postpone a discussion of this result to Section 9.

8.1 Reconstruction attacks

We argued in Section 1 that any non-trivial mechanism must be randomized. It follows that, at least for some database, query, and choice of random bits, the response produced by the mechanism is not perfectly accurate. The question of how *in*accurate answers must be in order to protect privacy makes sense in all computational models: interactive, non-interactive, and the models discussed in Section 12.

For the lower bounds on distortion, we assume for simplicity that the database consists of a single — but very sensitive — bit per person, so we can think of the database as an n-bit Boolean vector $d = (d_1, \ldots, d_n)$. This is an abstraction of a setting in which the database rows are quite complex, for example, they may be medical records, but the attacker is interested in one specific field, such as the presence or absence of the sickle cell trait. The abstracted attack consists of issuing a string of queries, each described by a subset S of the database rows. The query is asking how many 1's are in the selected rows. Representing the query as the n-bit characteristic vector $\mathbf{S}$ of the set S, with 1s in all the positions corresponding to rows in S and 0s everywhere else, the true answer to the query is the inner product $A(S) = \sum_{i=1}^{n} d_i \mathbf{S}_i$.

Fix an arbitrary privacy mechanism. We will let $r(S)$ denote the response to the query S. This may be obtained explicitly, say, if the mechanism is interactive and the query S is issued, or if the mechanism is given all the queries in advance and produces a list of answers, or implicitly, which occurs if the mechanism produces a synopsis from which the analysts extracts $r(S)$. Note that $r(S)$ may depend on random choices made by the mechanism and the history of queries. Let $E(S, r(S))$ denote the *error*, also called *noise* or *distortion*, of the response $r(S)$, so $E(S, r(S)) = |A(S) - r(S)|$.

The question we want to ask is, "How much noise is needed in order to preserve privacy?" Differential privacy is a specific privacy guarantee, but one might also consider weaker notions, so rather than guaranteeing privacy the modest goal in the lower bound arguments will simply be to prevent privacy catastrophes.

Definition 8.1. A mechanism is *blatantly non-private* if an adversary can construct a candidate database c that agrees with the real database d in all but $o(n)$ entries, i.e., $\|c - d\|_0 \in o(n)$.

In other words, a mechanism is blatantly non-private if it permits a reconstruction attack that allows the adversary to correctly guess the secret bit of all but $o(n)$ members of the database. (There is no requirement that the adversary know on which answers it is correct.)

Theorem 8.1. Let $\mathcal{M}$ be a mechanism with distortion of magnitude bounded by E. Then there exists an adversary that can reconstruct the database to within $4E$ positions.

An easy consequence of the theorem is that a privacy mechanism adding noise with magnitude always bounded by, say, $n/401$, permits an adversary to correctly reconstruct 99% of the entries.

Proof. Let d be the true database. The adversary attacks in two phases:

1. **Estimate the number of 1s in all possible sets:** Query $\mathcal{M}$ on all subsets $S \subseteq [n]$.
2. **Rule out "distant" databases:** For every candidate database $c \in \{0,1\}^n$, if $\exists S \subseteq [n]$ such that $|\sum_{i \in S} c_i - \mathcal{M}(S)| > E$, then rule out c. If c is not ruled out, then output c and halt.

Since $\mathcal{M}(S)$ never errs by more than E, the real database will not be ruled out, so this simple (but inefficient!) algorithm will output *some* candidate database c. We will argue that the number of positions in which c and d differ is at most $4 \cdot E$.

Let I_0 be the indices in which $d_i = 0$, that is, $I_0 = \{i \mid d_i = 0\}$. Similarly, define $I_1 = \{i \mid d_i = 1\}$. Since c was not ruled out, $|\mathcal{M}(I_0) - \sum_{i \in I_0} c_i| \leq E$. However, by assumption $|\mathcal{M}(I_0) - \sum_{i \in I_0} d_i| \leq E$. It follows from the triangle inequality that c and d differ in at most $2E$ positions in I_0; the same argument shows that they differ in at most $2E$ positions in I_1. Thus, c and d agree on all but at most $4E$ positions. □

What if we consider more realistic bounds on the number of queries? We think of $\sqrt{n}$ as an interesting threshold on noise, for the following reason: if the database contains n people drawn uniformly at random

from a population of size $N \gg n$, and the fraction of the population satisfying a given condition is p, then we expect the number of rows in the database satisfying the property to be roughly $np \pm \Theta(\sqrt{n})$, by the properties of the binomial distribution. That is, the sampling error is on the order of $\sqrt{n}$. We would like that the noise introduced for privacy is smaller than the sampling error, ideally $o(\sqrt{n})$. The next result investigates the feasibility of such small error when the number of queries is linear in n. The result is negative.

Ignoring computational complexity, to see why there might exist a query-efficient attack we modify the problem slightly, looking at databases $d \in \{-1, 1\}^n$ and query vectors $v \in \{-1, 1\}^n$. The true answer is again defined to be $d \cdot v$, and the response is a noisy version of the true answer. Now, consider a candidate database c that is far from d, say, $\|c-d\|_0 \in \Omega(n)$. For a random $v \in_R \{-1, 1\}^n$, with constant probability we have $(c - d) \cdot v \in \Omega(\sqrt{n})$. To see this, fix $x \in \{-1, 1\}^n$ and choose $v \in_R \{-1, 1\}^n$. Then $x \cdot v$ is a sum of independent random variables $x_i v_i \in_R \{-1, 1\}$, which has expectation 0 and variance n, and is distributed according to a scaled and shifted binomial distribuiton. For the same reason, if c and d differ in at least αn rows, and v is chosen at random, then $(c - d) \cdot v$ is binomially distributed with mean 0 and variance at least αn. Thus, we expect $c \cdot v$ and $d \cdot v$ to differ by at least $\alpha\sqrt{n}$ with constant probability, by the properties of the binomial distribution. Note that we are using the *anti*-concentration property of the distribution, rather than the usual appeal to concentration.

This opens an attack for ruling out c when the noise is constrained to be $o(\sqrt{n})$: compute the difference between $c \cdot v$ and the noisy response $r(v)$. If the magnitude of this difference exceeds $\sqrt{n}$ — which will occur with constant probability over the choice of v — then rule out c. The next theorem formalizes this argument and further shows that the attack is resilient even to a large fraction of completely arbitrary responses: Using a linear number of ± 1 questions, an attacker can reconstruct almost the whole database if the curator is constrained to answer at least $\frac{1}{2} + \eta$ of the questions within an absolute error of $o(\sqrt{n})$.

Theorem 8.2. For any $\eta > 0$ and any function $\alpha = \alpha(n)$, there is constant b and an attack using bn ± 1 questions that reconstructs a

database that agrees with the real database in all but at most $(\frac{2\alpha}{\eta})^2$ entries, if the curator answers at least $\frac{1}{2}+\eta$ of the questions within an absolute error of α.

Proof. We begin with a simple lemma.

Lemma 8.3. Let $Y = \sum_{i=1}^{k} X_i$ where each X_i is a ± 2 independent Bernoulli random variable with mean zero. Then for any y and any $\ell \in \mathbb{N}$, $Pr[Y \in [2y, 2(y+\ell)]] \leq \frac{\ell+1}{\sqrt{k}}$.

Proof. Note that Y is always even and that $Pr[Y = 2y] = \binom{k}{(k+y)/2}(\frac{1}{2})^k$. This expression is at most $\binom{k}{\lceil k/2 \rceil}(\frac{1}{2})^k$. Using Stirling's approximation, which says that $n!$ can be approximated by $\sqrt{2n\pi}(n/e)^n$, this is bounded by $\sqrt{\frac{2}{\pi k}}$. The claim follows by a union bound over the $\ell+1$ possible values for Y in $[2y, 2(y+\ell)]$. □

The adversary's attack is to choose bn random vectors $v \in \{-1,1\}^n$, obtain responses $(y_1, \ldots, y_{bn})$, and then output any database c such that $|y_i - (Ac)_i| \leq \alpha$ for at least $\frac{1}{2}+\eta$ of the indices i, where A is the $bn \times n$ matrix whose rows are the random query vectors v.

Let the true database be d and let c be the reconstructed database. By assumption on the behavior of the mechanism, $|(Ad)_i - y_i| \leq \alpha$ for a $1/2+\eta$ fraction of $i \in [bn]$. Since c was not ruled out, we also have that $|(Ac)_i - y_i| \leq \alpha$ for a $1/2+\eta$ fraction of $i \in [bn]$. Since any two such sets of indices agree on at least a 2η fraction of $i \in [bn]$, we have from the triangle inequality that for at least $2\eta bn$ values of i, $|[(c-d)A]_i| \leq 2\alpha$.

We wish to argue that c agrees with d in all but $(\frac{2\alpha}{\eta})^2$ entries. We will show that if the reconstructed c is far from d, disagreeing on at least $(2\alpha/\eta)^2$ entries, the probability that a randomly chosen A will satisfy $|[A(c-d)]_i| \leq 2\alpha$ for at least $2\eta bn$ values of i will be extremely small — so small that, for a random A, it is extremely unlikely that there even exists a c far from d that is not eliminated by the queries in A.

Assume the vector $z = (c-d) \in \{-2, 0, 2\}^n$ has Hamming weight at least $(\frac{2\alpha}{\eta})^2$, so c is far from d. We have argued that, since c is produced by the attacker, $|(Az)_i| \leq 2\alpha$ for at least $2\eta bn$ values of i. We shall call such a z *bad with respect to* A. We will show that, with high probability over the choice of A, no z is bad with respect to A.

For any i, $v_i z$ is the sum of at least $(\frac{2\alpha}{\eta})^2$ ± 2 random values. Letting $k = (2\alpha/\eta)^2$ and $\ell = 2\alpha$, we have by Lemma 8.3 that the probability that $v_i z$ lies in an interval of size 4α is at most η, so the expected number of queries for which $|v_i z| \leq 2\alpha$ is at most ηbn. Chernoff bounds now imply that the probability that this number exceeds $2\eta bn$ is at most $\exp(-\frac{\eta bn}{4})$. Thus the probability of a particular $z = c - d$ being bad with respect to A is at most $\exp(-\frac{\eta bn}{4})$.

Taking a union bound over the atmost 3^n possible zs, we get that with probability at least $1 - \exp(-n(\frac{\eta b}{4} - \ln 3))$, no bad z exists. Taking $b > 4 \ln 3/\eta$, the probability that such a bad z exists is exponentially small in n. □

Preventing blatant non-privacy is a very low bar for a privacy mechanism, so if differential privacy is meaningful then lower bounds for preventing blatant non-privacy will also apply to any mechanism ensuring differential privacy. Although for the most part we ignore computational issues in this monograph, there is also the question of the efficiency of the attack. Suppose we were able to prove that (perhaps under some computational assumption) there exist low-distortion mechanisms that are "hard" to break; for example, mechanisms for which producing a candidate database c close to the original database is hard? Then, although a low-distortion mechanism might fail to be differentially private in theory, it could conceivably provide privacy against bounded adversaries. Unfortunately, this is not the case. In particular, when the noise is always in $o(\sqrt{n})$, there is an efficient attack using exactly n fixed queries; moreover, there is even a computationally efficient attack requiring a linear number of queries in which a 0.239 fraction may be answered with wild noise.

In the case of "internet scale" data sets, obtaining responses to n queries is infeasible, as n is extremely large, say, $n \geq 10^8$. What happens if the curator permits only a sublinear number of questions? This inquiry led to the first algorithmic results in (what has evolved to be) (ε, δ)-differential privacy, in which it was shown how to maintain privacy against a sublinear number of counting queries by adding binomial noise of order $o(\sqrt{n})$ — less than the sampling error! — to each true answer. Using the tools of differential privacy we can do this either

using either (1) the Gaussian mechanism or (2) the Laplace mechanism and advanced composition.

8.2 Lower bounds for differential privacy

The results of the previous section yielded lower bounds on distortion needed to ensure any reasonable notion of privacy. In contrast, the result in this section is specific to differential privacy. Although some of the details in the proof are quite technical, the main idea is elegant: suppose (somehow) the adversary has narrowed down the set of possible databases to a relatively small set S of 2^s vectors, where the L_1 distance between each pair of vectors is some large number Δ. Suppose further that we can find a k-dimensional query F, 1-Lipschitz in each of its output coordinates, with the property that the true answers to the query look very different (in L_∞ norm) on the different vectors in our set; for example, the distance on any two elements in the set may be $\Omega(k)$. It is helpful to think geometrically about the "answer space" $\mathbb{R}^k$. Each element x in the set S gives rise to a vector $F(x)$ in answer space. The actual response will be a perturbation of this point in answer space. Then a volume-based pigeon hole argument (in answer space) shows that, if with even moderate probability the (noisy) responses are "reasonably" close to the true answers, then ϵ cannot be very small.

This stems from the fact that for $(\varepsilon, 0)$-differentially private mechanisms $\mathcal{M}$, for *arbitrarily different* databases x, y, any response in the support of $\mathcal{M}(x)$ is also in the support of $\mathcal{M}(y)$. Taken together with the construction of an appropriate collection of vectors and a (contrived, non-counting) query, the result yields a lower bound on distortion that is linear k/ε. The argument appeals to Theorem 2.2, which discusses group privacy. In our case the group in question corresponds to the indices contributing to the (L_1) distance between a pair of vectors in S.

8.2.1 Lower bound by packing arguments

We begin with an observation which says, intuitively, that if the "likely" response regions, when the query is F, are disjoint, then we can bound

ϵ from below, showing that privacy can't be too good. When $\|F(x_i) - F(x_j)\|_\infty$ is large, this says that to get very good privacy, even when restricted to databases that differ in many places, we must get very erroneous responses on some coordinate of F.

The argument uses the histogram representation of databases. In the sequel, $d = |\mathcal{X}|$ denotes the size of the universe from which database elements are drawn.

Lemma 8.4. Assume the existence of a set $S = \{x_1, \ldots, x_{2^s}\}$, where each $x_i \in \mathbb{N}^d$, such that for $i \neq j$, $\|x_i - x_j\|_1 \leq \Delta$. Further, let $F : \mathbb{N}^d \to \mathbb{R}^k$ be a k-dimensional query. For $1 \leq i \leq 2^s$, let B_i denote a region in $\mathbb{R}^k$, the answer space, and assume that the B_i are mutually disjoint. If $\mathcal{M}$ is an $(\varepsilon, 0)$-differentially private mechanism for F such that, $\forall 1 \leq i \leq 2^s$, $\Pr[\mathcal{M}(x_i) \in B_i] \geq 1/2$, then $\varepsilon \geq \frac{\ln(2)(s-1)}{\Delta}$.

Proof. By assumption $\Pr[\mathcal{M}(x_j) \in B_j] \geq 2^{-1}$. Since the regions $B_1, \ldots, B_{2^s}$ are disjoint, $\exists j \neq i \in [2^s]$ such that $\Pr[\mathcal{M}(x_i) \in B_j] \leq 2^{-s}$. That is, for at least one of the $2^s - 1$ regions B_j, the probability that $\mathcal{M}(x_i)$ is mapped to this B_j is at most 2^{-s}. Combining this with differential privacy, we have

$$\frac{2^{-1}}{2^{-s}} \leq \frac{\Pr_{\mathcal{M}}[B_j|x_j]}{\Pr_{\mathcal{M}}[B_j|x_i]} \leq \exp(\varepsilon\Delta). \qquad \square$$

Corollary 8.5. Let $S = \{x_1, \ldots, x_{2^s}\}$ be as in Lemma 8.4, and assume that for any $i \neq j$, $\|F(x_i) - F(x_j)\|_\infty \geq \eta$. Let B_i denote the L_∞ ball in $\mathbb{R}^k$ of radius $\eta/2$ centered at x_i. Let $\mathcal{M}$ be any ε-differentially private mechansim for F satisfying

$$\forall 1 \leq i \leq 2^s : \Pr[\mathcal{M}(x_i) \in B_i] \geq 1/2.$$

Then $\varepsilon \geq \frac{(\ln 2)(s-1)}{\Delta}$.

Proof. The regions $B_1, \ldots, B_{2^s}$ are disjoint, so the conditions of Lemma 8.4 are satisfied. The corollary follows by applying the lemma and taking logarithms. $\square$

In Theorem 8.8 below we will look at queries F that are simply k independently and randomly generated (nonlinear!) queries. For

suitable S and F (we will work to find these) the corollary says that if with probability at least $1/2$ *all* responses simultaneously have small error, then privacy can't be too good. In other words,

Claim 8.6 (Informal Restatement of Corollary 8.5)**.** To obtain $(\varepsilon, 0)$-differential privacy for $\varepsilon \leq \frac{\ln(2)(s-1)}{\Delta}$, the mechanism must add noise with L_∞ norm greater than $\eta/2$ with probability exceeding $1/2$.

As a warm-up exercise, we prove an easier theorem that requires a large data universe.

Theorem 8.7. Let $\mathcal{X} = \{0,1\}^k$. Let $\mathcal{M} : \mathcal{X}^n \to \mathbb{R}^k$ be an $(\varepsilon, 0)$-differentially private mechanism such that for every database $x \in \mathcal{X}^n$ with probability at least $1/2$ $\mathcal{M}(x)$ outputs all of the 1-way marginals of x with error smaller than $n/2$. That is, for each $j \in [k]$, the jth component of $\mathcal{M}(x)$ should approximately equal the number of rows of x whose jth bit is 1, up to an error smaller than $n/2$. Then $n \in \Omega(k/\varepsilon)$.

Note that this bound is tight to within a constant factor, by the simple composition theorem, and that it separates $(\varepsilon, 0)$-differential privacy from (ε, δ)-differential privacy, for $\delta \in 2^{-o(n)}$, since, by the advanced composition theorem (Theorem 3.20), Laplace noise with parameter $b = \sqrt{k \ln(1/\delta)}/\varepsilon$ suffices for the former, in contrast to $\Omega(k/\varepsilon)$ needed for the latter. Taking $k \in \Theta(n)$ and, say, $\delta = 2^{-\log^2 n}$, yields the separation.

Proof. For every string $w \in \{0,1\}^k$, consider the database x_w consisting of n identical rows, all of which equal w. Let $B_w \in \mathbb{R}^k$ consist of all tuples of numbers that provide answers to the 1-way marginals on x with error less than $n/2$. That is,

$$B_w = \{(a_1, \ldots, a_k)\} \in \mathbb{R}^k : \forall i \in [k] \, |a_i - nw_i| < n/2\}.$$

Put differently, B_w is the open ℓ_∞ of radius $n/2$ around $nw \in \{0,n\}^k$. Notice that the sets B_w are mutually disjoint.

If M is an accurate mechanism for answering 1-way marginals, then for every w the probability of landing in B_w when the database is x_w should be at least $1/2$: $\Pr[\mathcal{M}(x_w) \in B_w] \geq 1/2$. Thus, setting $\Delta = n$ and $s = k$ in Corollary 8.5 we have $\varepsilon \geq \frac{(\ln 2)(s-1)}{\Delta}$. □

Theorem 8.8. For any $k, d, n \in \mathbb{N}$ and $\varepsilon \in (0, 1/40]$, where $n \geq \min\{k/\varepsilon, d/\varepsilon\}$, there is a query $F : \mathbb{N}^d \to \mathbb{R}^k$ with per-coordinate sensitivity at most 1 such that any $(\varepsilon, 0)$-differentially private mechanism adds noise of L_∞ norm $\Omega(\min\{k/\varepsilon, d/\varepsilon\})$ with probability at least $1/2$ on some databases of weight at most n.

Note that $d = |\mathcal{X}|$ need not be large here, in contrast to the requirement in Theorem 8.7.

Proof. Let $\ell = \min\{k, d\}$. Using error-correcting codes we can construct a set $S = \{x_1, \ldots, x_{2^s}\}$, where $s = \ell/400$, such that each $x_i \in \mathbb{N}^d$ and in addition

1. $\forall i : \|x_i\|_1 \leq w = \ell/(1280\varepsilon)$
2. $\forall i \neq j$, $\|x_i - x_j\|_1 \geq w/10$

We do not give details here, but we note that the databases in S are of size at most $w < n$, and so $\|x_i - x_j\|_1 \leq 2w$. Taking $\Delta = 2w$ the set S satisfies the conditions of Corollary 8.5. The remainder of our effort is to obtain the queries F to which we will apply Corollary 8.5. Given $S = \{x_1, \ldots, x_{2^s}\}$, where each $x_i \in \mathbb{N}^d$, the first step is to define a mapping from the space of histograms to vectors in $\mathbb{R}^{2^s}$, $\mathcal{L}_S : \mathbb{N}^d \to \mathbb{R}^{2^s}$. Intuitively (and imprecisely!), given a histogram x, the mapping lists, for each $x_i \in S$, the L_1 distance from x to x_i. More precisely, letting w be an upper bound on the weight of any x_i in our collection we define the mapping as follows.

- For every $x_i \in S$, there is a coordinate i in the mapping.
- The ith coordinate of $\mathcal{L}_S(x)$ is $\max\{w/30 - \|x_i - z\|_1, 0\}$.

Claim 8.9. If $x_1, \ldots, x_{2^s}$ satisfy the conditions

1. $\forall i \|x_i\|_1 \leq w$; and
2. $\forall i \neq j \|x_i - x_j\|_1 \geq w/10$

then the map $\mathcal{L}_S$ is 1-Lipschitz; in particular, if $\|z_1 - z_2\|_1 = 1$, then $\|\mathcal{L}_S(z_1) - \mathcal{L}_S(z_2)\|_1 \leq 1$, assuming $w \geq 31$.

Proof. Since we assume $w \geq 31$ we have that if $z \in \mathbb{N}^d$ is close to some $x_i \in S$, meaning $w/30 > \|x_i - z\|_1$, then z cannot be close to any other $x_j \in S$, and the same is true for all $\|z' - z\|_1 \leq 1$. Thus, for any z_1, z_2 such that $\|z_1 - z_2\| \leq 1$, if A denotes the set of coordinates where at least one of $\mathcal{L}_S(z_1)$ or $\mathcal{L}_S(z_2)$ is non-zero, then A is either empty or is a singleton set. Given this, the statement in the claim is immediate from the fact that the mapping corresponding to any particular coordinate is clearly 1-Lipschitz. □

We can finally describe the queries F. Corresponding to any $r \in \{-1, 1\}^{2^s}$, we define $f_r : \mathbb{N}^d \to \mathbb{R}$, as

$$f_r(x) = \sum_{i=1}^{d} \mathcal{L}_S(x)_i \cdot r_i \,,$$

which is simply the inner product $\mathcal{L}_S \cdot r$. F will be a random map $F : \mathbb{N}^d \to \mathbb{R}^k$: Pick $r_1, \ldots, r_k \in \{-1, 1\}^{2^s}$ independently and uniformly at random and define

$$F(x) = (f_{r_1}(x), \ldots, f_{r_k}(x)) \,.$$

That is, $F(x)$ is simply the result of the inner product of $\mathcal{L}_S(x)$ with k randomly chosen ± 1 vectors.

Note that for any $x \in S$ $\mathcal{L}_S(x)$ has one coordinate with value $w/30$ (and the others are all zero), so $\forall r_i \in \{-1, 1\}^{2^s}$ and $x \in S$ we have $|f_{r_i}(x)| = w/30$. Now consider any $x_h, x_j \in S$, where $h \neq j$. It follows that for any $r_i \in \{-1, 1\}^{2^s}$,

$$\Pr_{r_i}[|f_{r_i}(x_h) - f_{r_i}(x_j)| \geq w/15] \geq 1/2$$

(this event occurs when $(r_i)_h = -(r_i)_j$). A basic application of the Chernoff bound implies that

$$\Pr_{r_1,\ldots,r_k}[\text{For at least } 1/10 \text{ of the } r_i\text{s},$$
$$|f_{r_i}(x_h) - f_{r_i}(x_j)| \geq w/15] \geq 1 - 2^{-k/30} \,.$$

Now, the total number of pairs (x_i, x_j) of databases such that $x_i, x_j \in S$ is at most $2^{2s} \leq 2^{k/200}$. Taking a union bound this implies

$$\Pr_{r_1,\ldots,r_k}[\forall h \neq j, \quad \text{For at least } 1/10 \text{ of the } r_i\text{s},$$
$$|f_{r_i}(x_h) - f_{r_i}(x_j)| \geq w/15] \geq 1 - 2^{-k/40}$$

This implies that we can fix $r_1, \ldots, r_k$ such that the following is true.

$$\forall h \neq j, \quad \text{For at least } 1/10 \text{ of the } r_i\text{s}, \quad |f_{r_i}(x_h) - f_{r_i}(x_j)| \geq w/15$$

Thus, for any $x_h \neq x_j \in S$, $\|F(x_h) - F(x_j)\|_\infty \geq w/15$.

Setting $\Delta = 2w$ and $s = \ell/400 > 3\varepsilon w$ (as we did above), and $\eta = w/15$, we satisfy the conditions of Corollary 8.5 and conclude $\Delta \leq (s-1)/\varepsilon$, proving the theorem (via Claim 8.6). □

The theorem is almost tight: if $k \leq d$ then we can apply the Laplace mechanism to each of the k sensitivity 1 component queries in F with parameter k/ε, and we expect the maximum distortion to be $\Theta(k \ln k/\varepsilon)$. On the other hand, if $d \leq k$ then we can apply the Laplace mechanism to the d-dimensional histogram representing the database, and we expect the maximum distortion to be $\Theta(d \ln d/\varepsilon)$.

The theorem actually shows that, given knowledge of the set S and knowledge that the actual database is an element $x \in S$, the adversary can completely determine x if the L_∞ norm of the distortion is too small. How in real life might the adversary obtain a set S of the type used in the attack? This can occur when a *non-private* database system has been running on a dataset, say, x. For example, x could be a vector in $\{0,1\}^n$ and the adversary may have learned, through a sequence of linear queries, that $x \in \mathcal{C}$, a linear code of distance, say $n^{2/3}$. Of course, if the database system is not promising privacy there is no problem. The problem arises if the administrator decides to replace the existing system with a differentially private mechanism — after several queries have received noise-free responses. In particular, if the administrator chooses to use (ε, δ)-differential privacy for subsequent k queries then the distortion might fall below the $\Omega(k/\varepsilon)$ lower bound, permitting the attack described in the proof of Theorem 8.8.

The theorem also emphasizes that there is a fundamental difference between auxiliary information about (sets of) members of the database and information about the database *as a whole*. Of course, we already knew this: being told that the number of secret bits sums to exactly $5,000$ completely destroys differential privacy, and an adversary that already knew the secret bit of every member of the database except one individual could then conclude the secret bit of the remaining individual.

Additional Consequences. Suppose $k \leq d$, so $\ell = k$ in Theorem 8.8. The linear in k/ε lower bound on noise for k queries sketched in the previous section immediately yields a separation between counting queries and arbitrary 1-sensitivity queries, as the SmallDB construction answers (more than) n queries with noise roughly $n^{2/3}$ while maintaining differential privacy. Indeed, this result also permits us to conclude that there is no small α-net for large sets of arbitrary low sensitivity queries, for $\alpha \in o(n)$ (as otherwise the net mechanism would yield an $(\varepsilon, 0)$ algorithm of desired accuracy).

8.3 Bibliographic notes

The first reconstruction attacks, including Theorem 8.1, are due to Dinur and Nissim [18], who also gave an attack requiring only polynomial time computation and $O(n \log^2 n)$ queries, provided the noise is always $o(\sqrt{n})$. Realizing that attacks requiring n random linear queries, when n is "internet scale," are infeasible, Dinur, Dwork, and Nissim gave the first positive results, showing that for a sublinear number of subset sum queries, a form of privacy (now known to imply (ε, δ)-differential privacy) can be achieved by adding noise scaled to $o(\sqrt{n})$ [18]. This was exciting because it suggested that, if we think of the database as drawn from an underlying population, then, even for a relatively large number of counting queries, privacy could be achieved with distortion smaller than the sampling error. This eventulaly lead, via more general queries [31, 6], to differential privacy. The view of these queries as a privacy-preserving programming primitive [6] inspired McSherry's Privacy Integrated Queries programming platform [59].

The reconstruction attack of Theorem 8.2 appears in [24], where Dwork, McSherry, and Talwar showed that polynomial time reconstruction is possible even if a 0.239 fraction of the responses have wild, arbitrary, noise, provided the others have noise $o(\sqrt{n})$.

The geometric approach, and in particular Lemma 8.4, is due to Hardt and Talwar [45], who also gave a geometry-based algorithm proving these bounds tight for small numbers $k \leq n$ of queries, under a

commonly believed conjecture. Dependence on the conjecture was later removed by Bhaskara et al. [5]. The geometric approach was extended to arbitrary numbers of queries by Nikolov et al. [66], who gave an algorithm with instance-optimal mean squared error. For the few queries case this leads, via a boosting argument, to low expected worst-case error. Theorem 8.8 is due to De [17].

9

Differential Privacy and Computational Complexity

Our discussion of differential privacy has so far ignored issues of computational complexity, permitting both the curator and the adversary to be computationally unbounded. In reality, both curator and adversary may be computationally bounded.

Confining ourselves to a computationally bounded curator restricts what the curator can do, making it harder to achieve differential privacy. And indeed, we will show an example of a class of counting queries that, under standard complexity theoretic assumptions, does not permit *efficient* generation of a synthetic database, even though inefficient algorithms, such as SmallDB and Private Multiplicative Weights, are known. Very roughly, the database rows are digital signatures, signed with keys to which the curator does not have access. The intuition will be that any row in a synthetic database must either be copied from the original — violating privacy — or must be a signature on a *new* message, i.e., a forgery — violating the unforgeability property of a digital signature scheme. Unfortuately, this state of affairs is not limited to (potentially contrived) examples based on digital signatures: it is even difficult to create a synthetic database that maintains relatively

accurate two-way marginals.[1] On the positive side, given a set $\mathcal{Q}$ of queries and an n-row database with rows drawn from a universe $\mathcal{X}$, a synthetic database can be generated in time polynomial in n, $|\mathcal{X}|$, and $|\mathcal{Q}|$.

If we abandon the goal of a synthetic database and content ourselves with a data structure from which we can obtain a relatively accurate approximation to the answer to each query, the situation is much more interesting. It turns out that the problem is intimately related to the *tracing traitors* problem, in which the goal is to discourage piracy while distributing digital content to paying customers.

If the adversary is restricted to polynomial time, then it becomes easier to achieve differential privacy. In fact, the immensely powerful concept of *secure function evaluation* yields a natural way avoid the trusted curator (while giving better accuracy than randomized response), as well as a natural way to allow multiple trusted curators, who for legal reasons cannot share their data sets, to respond to queries on what is effectively a merged data set. Briefly put, secure function evaluation is a cryptographic primitive that permits a collection of n parties $p_1, p_2, \ldots, p_n$, of which fewer than some fixed fraction are faulty (the fraction varies according to the type of faults; for "honest-but-curious" faults the fraction is 1), to cooperatively compute any function $f(x_1, \ldots, x_n)$, where x_i is the input, or *value*, of party p_i, in such a way that no coalition of faulty parties can either disrupt the computation or learn more about the values of the non-faulty parties than can be deduced from the function output and the values of the members of the coalition. These two properties are traditionally called *correctness* and *privacy*. This privacy notion, let us call it *SFE privacy*, is very different from differential privacy. Let V be the set of values held by the faulty parties, and let p_i be a non-faulty party.[2] SFE privacy permits the faulty parties to learn x_i if x_i can be deduced from $V \cup \{f(x_1, \ldots, x_n)\}$; differential privacy would therefore not permit exact release of $f(x_1, \ldots, x_n)$. However, secure function evaluation

[1] Recall that the two-way marginals are the counts, for every pair of attribute values, of the number of rows in the database having this pair of values.

[2] In the honest but curious case we can let $V = \{x_j\}$ for any party P_j.

protocols for computing a function f can easily be modified to obtain differentially private protocols for f, simply by defining a new function, g, to be the result of adding Laplace noise $\mathrm{Lap}(\Delta f/\varepsilon)$ to the value of f. In principle, secure function evaluation permits evaluation of g. Since g is differentially private and the SFE privacy property, applied to g, says that nothing can be learned about the inputs that is not learnable from the value of $g(x_1, \ldots, x_n)$ together with V, differential privacy is ensured, provided the faulty players are restricted to polynomial time. Thus, secure function evaluation allows a computational notion of differential privacy to be achieved, even without a trusted curator, at no loss in accuracy when compared to what can be achieved with a trusted curator. In particular, counting queries can be answered with constant expected error while ensuring computational differential privacy, with no trusted curator. We will see that, without cryptography, the error must be $\Omega(n^{1/2})$, proving that computational assumptions provably buy accuracy, in the multiparty case.

9.1 Polynomial time curators

In this section we show that, under standard cryptographic assumptions, it is computationally difficult to create a synthetic database that will yield accurate answers to an appropriately chosen class of counting queries, while ensuring even a minimal notion of privacy.

This result has several extensions; for example, to the case in which the set of queries is small (but the data universe remains large), and the case in which the data universe is small (but the set of queries is large). In addition, similar negative results have been obtained for certain natural families of queries, such as those corresponding to conjunctions.

We will use the term *syntheticize* to denote the process of generating a synthetic database in a privacy-preserving fashion[3]. Thus, the results in this section concern the computational hardness of syntheticizing. Our notion of privacy will be far weaker than differential privacy, so hardness of syntheticizing will imply hardness of generating a synthetic

[3]In Section 6 a syntheticizer took as input a synopsis; here we are starting with a database, which is a trivial synopsis.

database in a differentially private fashion. Specifically, we will say that syntheticizing is hard if it is hard even to avoid leaking input items in their entirety. That is, some item is always completely exposed.

Note that if, in contrast, leaking a few input items is not considered a privacy breach, then syntheticizing is easily achieved by releasing a randomly chosen subset of the input items. Utility for this "synthetic database" comes from sampling bounds: with high probability this subset will preserve utility even with respect to a large set of counting queries.

When introducing complexity assumptions, we require a *security parameter* in order to express sizes; for example, sizes of sets, lengths of messages, number of bits in a decryption key, and so on, as well as to express computational difficulty. The security parameter, denoted κ, represents "reasonable" sizes and effort. For example, it is assumed that it is feasible to exhaustively search a set whose size is (any fixed) polynomial in the security parameter.

Computational complexity is an asymptotic notion — we are concerned with how the difficulty of a task increases as the sizes of the objects (data universe, database, query family) grow. Thus, for example, we therefore need to think not just of a distribution on databases of a single size (what we have been calling n in the rest of this monograph), but of an ensemble of distributions, indexed by the security parameter. In a related vein, when we introduce complexity we tend to "soften" claims: forging a signature is not impossible — one might be lucky! Rather, we assume that no efficient algorithm succeeds with non-negligible probability, where "efficient" and "non-negligible" are defined in terms of the security parameter. We will ignore these fine points in our intuitive discussion, but will keep them in the formal theorem statements.

Speaking informally, a distribution of databases is *hard to synthesticize* (with respect to some family $\mathcal{Q}$ of queries) if for any efficient (alleged) syntheticizer, with high probability over a database drawn from the distribution, at least one of the database items can be extracted from the alleged syntheticizer'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) syntheticizer indeed compromises the privacy of input items in a strong sense.

Definition 9.1 below will formalize our utility requirements for a syntheticizer. There are three parameters: α describes the accuracy requirement (being within α is considered accurate); γ describes the fraction of the queries on which a successful synthesis is allowed to be inaccurate, and β will be the probability of failure.

For an algorithm A producing synthetic databases, we say that an output $A(x)$ is (α, γ)-*accurate for a query set* $\mathcal{Q}$ if $|q(A(x)) - q(x)| \leq \alpha$ for a $1 - \gamma$ fraction of the queries $q \in \mathcal{Q}$.

Definition 9.1 ((α, β, γ)-Utility)**.** Let $\mathcal{Q}$ be a set of queries and $\mathcal{X}$ a data universe. A syntheticizer A has (α, β, γ)-utility for n-item databases with respect to $\mathcal{Q}$ and $\mathcal{X}$ if for any n-item database x:

$$\Pr\left[A(x) \text{ is } (\alpha, \gamma)\text{-accurate for } \mathcal{Q}\right] \geq 1 - \beta$$

where the probability is over the coins of A.

Let $\mathcal{Q} = \{\mathcal{Q}_n\}_{n=1,2,\ldots}$ be a query family ensemble, $\mathcal{X} = \{\mathcal{X}_n\}_{n=1,2,\ldots}$ be a data universe ensemble. An algorithm is said to be *efficient* if its running time is $\text{poly}(n, \log(|\mathcal{Q}_n|), \log(|\mathcal{X}_n|))$.

In the next definition we describe what it means for a family of distributions to be hard to syntheticize. A little more specifically we will say what it means to be hard to generate synthetic databases that provide (α, γ)-accuracy. As usual, we have to make this an asymptotic statement.

Definition 9.2 ($(\mu, \alpha, \beta, \gamma, \mathcal{Q})$-Hard-to-Syntheticize Database Distribution)**.** Let $\mathcal{Q} = \{\mathcal{Q}_n\}_{n=1,2,\ldots}$ be a query family ensemble, $\mathcal{X} = \{\mathcal{X}_n\}_{n=1,2,\ldots}$ be a data universe ensemble, and let $\mu, \alpha, \beta, \gamma \in [0, 1]$. Let n be a database size and $\mathcal{D}$ an ensemble of distributions, where $\mathcal{D}_n$ is over collections of $n + 1$ items from X_n.

We denote by $(x, i, x_i') \sim \mathcal{D}_n$ the experiment of choosing an n-element database, an index i chosen uniformly from $[n]$, and an additional element x_i' from $\mathcal{X}_n$. A sample from $\mathcal{D}_n$ gives us a pair of databases: x and the result of replacing the ith element of x (under

a canonical ordering) with x'_i. Thus, we think of $\mathcal{D}_n$ as specifying a distribution on n-item databases (and their neighbors).

We say that $\mathcal{D}$ is $(\mu, \alpha, \beta, \gamma, \mathcal{Q})$-*hard-to Syntheticize* if there exists an efficient algorithm T such that for any alleged efficient syntheticizer A the following two conditions hold:

1. With probability $1-\mu$ over the choice of database $x \sim \mathcal{D}$ and the coins of A and T, if $A(x)$ maintains α-utility for a $1-\gamma$ fraction of queries, then T can recover one of the rows of x from $A(x)$:

$$\Pr_{\substack{(x,i,x'_i)\sim D_n \\ \text{coin flips of } A,T}} [(A(x) \text{ maintains } (\alpha,\beta,\gamma)\text{-utility}) \text{ and } (x \cap T(A(x)) = \emptyset)] \leq \mu$$

2. For *every* efficient algorithm A, and for every $i \in [n]$, if we draw (x, i, 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:

$$\Pr_{\substack{(x,i,x'_i)\sim D_n \\ \text{coin flips of } A,\ T}} [x_i \in T(A(x'))] \leq \mu.$$

Later, we will be interested in offline mechanisms that produce arbitrary synopses, not necessarily synthetic databases. In this case we will be interested in the related notion of *hard to sanitize* (rather than hard to Syntheticize), for which we simply drop the requirement that A produce a synthetic database.

9.2 Some hard-to-Syntheticize distributions

We now construct three distributions that are hard to syntheticize.

A *signature scheme* is given by a triple of (possibly randomized) algorithms $(\text{Gen}, \text{Sign}, \text{Verify})$:

- $\text{Gen} : 1^{\mathbb{N}} \to \{(\text{SK}, \text{VK})_n\}_{n=1,2,\ldots}$ is used to generate a pair consisting of a (secret) *signing* key and a (public) *verification* key. It takes only the security parameter $\kappa \in \mathbb{N}$, written in unary, as input, and produces a pair drawn from $(\text{SK}, \text{VK})_\kappa$, the distribution on (signature,verification) key pairs indexed by κ; we let

$p_s(\kappa), p_v(\kappa), \ell s(\kappa)$ denote the lengths of the signing key, verification key, and signature, respectively.

- Sign : $\mathrm{SK}_\kappa \times \{0,1\}^{\ell(\kappa)} \to \{0,1\}^{\ell s(\kappa)}$ takes as input a signing key from a pair drawn from $(\mathrm{SK}, \mathrm{VK})_\kappa$ and a message m of length $\ell(\kappa)$, and produces a signature on m;
- Verify : $\mathrm{VK}_\kappa \times \{0,1\}^* \times \{0,1\}^{\ell(\kappa)} \to \{0,1\}$ takes as input a verification key, a string σ, and a message m of length $\ell(\kappa)$, and checks that σ is indeed a valid signature of m under the given verification key.

Keys, message lengths, and signature lengths are all polynomial in κ.

The notion of security required is that, given any polynomial (in κ) number of valid (message, signature) pairs, it is hard to forge *any* new signature, even a new signature of a previously signed message (recall that the signing algorithm may be randomized, so there may exist multiple valid signatures of the same message under the same signing key). Such a signature scheme can be constructed from any one-way function. Speaking informally, these are functions that are easy to compute — $f(x)$ can be computed in time polynomial in the length (number of bits) of x, but hard to invert: for every probabilistic polynomial time algorithm, running in time polyomial in the security parameter κ, the probability, over a randomly chosen x in the domain of f, of finding *any* valid pre-image of $f(x)$, grows more slowly than the inverse of any polynomial in κ.

Hard to Syntheticize Distribution I: Fix an arbitrary signature scheme. The set $\mathcal{Q}_\kappa$ of counting queries contains one counting query q_{vk} for each verification key $vk \in \mathrm{VK}_\kappa$. The data universe $\mathcal{X}_\kappa$ consists of the set of all possible (message, signature) pairs of the form for messages of length $\ell(\kappa)$ signed with keys in VK_κ.

The distribution $\mathcal{D}_\kappa$ on databases is defined by the following sampling procedure. Run the signature scheme generator $\mathrm{Gen}(1^\kappa)$ to obtain (sk, vk). Randomly choose $n = \kappa$ messages in $\{0,1\}^{\ell(\kappa)}$ and run the signing procedure for each one, obtaining a set of n (message, signature) pairs all signed with key sk. This is the database x. Note that all the messages in the database are signed with the *same* signing key.

A data universe item (m, σ) satisfies the predicate q_{vk} if and only if $\text{Verify}(vk, m, \sigma) = 1$, i.e., σ is a valid signature for m according to verification key vk.

Let $x \in_R \mathcal{D}_\kappa$ be a database, and let sk be the signing key used, with corresponding verification key vk. Assuming that the synthesizer 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 query vk is 1). By the unforgeability properties of the signature scheme, all of these must come from the input database x — the polynomial time bounded curator, running in time $\text{poly}(\kappa)$, cannot generate generate a *new* valid (message, signature) pair. (Only slightly) more formally, the probability that an efficient algorithm could produce a (message, signature) pair that is verifiable with key vk, but is not in x, is negligible, so with overwhelming probability any y that is produced by an efficient syntheticizer will only contain rows of x.[4] This contradicts (any reasonable notion of) privacy.

In this construction, both $\mathcal{Q}_\kappa$ (the set of verification keys) and $\mathcal{X}_\kappa$ (the set of (message, signature) pairs) are large (superpolynomial in κ). When both sets are small, efficient differentially private generation of synthetic datasets is possible. That is, there is a differentially private syntheticizer whose running time is polynomial in $n = \kappa$, $|\mathcal{Q}_\kappa|$ and $|\mathcal{X}_\kappa|$: compute noisy counts using the Laplace mechanism to obtain a synopsis and then run the syntheticizer from Section 6. Thus, when both of these have size polynomial in κ the running time of the syntheticizer is polynomial in κ.

We now briefly discuss generalizations of the first hardness result to the cases in which one of these sets is small (but the other remains large).

Hard to Syntheticize Distribution II: In the database distribution above, we chose a single (sk, vk) key pair and generated a database of

[4]The quantification order is important, as otherwise the syntheticizer could have the signing key hardwired in. We first fix the syntheticizer, then run the generator and build the database. The probability is over all the randomness in the experiment: choice of key pair, construction of the database, and randomness used by the syntheticizer.

messages, all signed using sk; hardness was obtained by requiring the syntheticizer to generate a new signature under sk, in order for the syntheticized database to provide an accurate answer to the query q_{vk}. To obtain hardness for syntheticizing when the size of the set of queries is only polynomial in the security parameter, we again use digital signatures, signed with a unique key, but we cannot afford to have a query for each possible verification key vk, as these are too numerous.

To address this, we make two changes:

1. Database rows now have the form (verification key, message, signature). more precisely, the data universe consists of (key,message,signature) triples $\mathcal{X} = \{(vk, m, s) : vk \in \mathrm{VK}_\kappa, m \in \{0,1\}^{\ell(\kappa)}, s \in \{0,1\}^{\ell s(\kappa)}\}$.
2. We add to the query class exactly $2p_v(\kappa)$ queries, where $p_v(\kappa)$ is the length of the verification keys produced by running the generation algorithm $\mathrm{Gen}(1^\kappa)$. The queries have the form (i, b) where $1 \leq i \leq p_v(\kappa)$ and $b \in \{0,1\}$. The meaning of the query "(i, b)" is, "What fraction of the database rows are of the form (vk, m, s) where $\mathrm{Verify}(vk, m, s) = 1$ and the ith bit of vk is b?" By populating a database with messages signed according to a single key vk, we ensure that the responses to these queries should be close to one for all $1 \leq i \leq p(\kappa)$ when $vk_i = b$, and close to zero when $vk_i = 1 - b$.

With this in mind, the hard to syntheticize distribution on databases is constructed by the following sampling procedure: Generate a signature-verification key pair $(sk, vk) \leftarrow \mathrm{Gen}(1^\kappa)$, and choose $n = \kappa$ messages $m_1, \ldots, m_n$ uniformly from $\{0,1\}^{\ell(\kappa)}$. The database x will have n, rows; for $j \in [n]$ the jth row is the verification key, the jth message and its valid signature, i.e., the tuple $(vk, m_j, \mathrm{Sign}(m_j, sk))$. Next, choose i uniformly from $[n]$. To generate the $(n+1)$st item x'_i, just generate a new message-signature pair (using the same key sk).

Hard to Syntheticize Distribution III: To prove hardness for the case of a polynomial (in κ) sized message space (but superpolynomial sized query set) we use a *pseudorandom function*. Roughly speaking, these are polynomial time computable functions with small descriptions that

cannot efficiently be distinguished, based only on their input-output behavior, from truly random functions (whose descriptions are long). This result only gives hardness of syntheticizing if we insist on maintaining utility for *all* queries. Indeed, if we are interested only in ensuring on-average utility, then the base generator for counting queries described in Section 6 yields an efficient algorithm for syntheticizing when the universe $\mathcal{X}$ is of polynomial size, even when $\mathcal{Q}$ is exponentially large.

Let $\{f_s\}_{s\in\{0,1\}^\kappa}$ be a family of pseudo-random functions from $[\ell]$ to $[\ell]$, where $\ell \in \text{poly}(\kappa)$. More specifically, we need that the set of all pairs of elements in $[\ell]$ is "small," but larger than κ; this way the κ-bit string describing a function in the family is shorter than the $\ell \log_2 \ell$ bits needed to describe a random function mapping $[\ell]$ to $[\ell]$. Such a family of pseudorandom functions can be constructed from any one-way function.

Our data universe will be the set of all pairs of elements in $[\ell]$: $\mathcal{X} = \{(a, b) : a, b \in [\ell]\}$. $\mathcal{Q}_\kappa$ will contain two types of queries:

1. There will be one query for each function $\{f_s\}_{s\in\{0,1\}^\kappa}$ in the family. A universe element $(a, b) \in \mathcal{X}$ satisfies the query s if and only if $f_s(a) = b$.
2. There will be a relatively small number, say κ, truly random queries. Such a query can be constructed by randomly choosing, for each $(a, b) \in \mathcal{X}$, whether or not (a, b) will satisfy the query.

The hard to syntheticize distribution is generated as follows. First, we select a random string $s \in \{0, 1\}^\kappa$, specifying a function in our family. Next, we generate, for $n = \kappa$ distinct values $a_1, \ldots, a_n$ chosen at random from $[\ell]$ without replacement, the universe element $(a, f_s(a))$.

The intuition is simple, relies only on the first type of query, and does not make use of the distinctness of the a_i. Given a database x generated according to our distribution, where the pseudo-random function is given by s, the syntheticizer must create a synthetic database (almost) all of whose rows must satisfy the query s. The intuition is that it can't *reliably* find input-output pairs that do not appear in x. A little more precisely, for an arbitrary element $a \in [\ell]$ such that no

row in x is of the form $(a, f_s(a))$, the pseudo-randomness of f_s says that an efficient syntheticizer should have probability at most negligibly more than $1/\ell$ of finding $f_s(a)$. In this sense the pseudo-randomness gives us properties similar to, although somewhat weaker than, what we obtained from digital signatures.

Of course, for any given $a \in [\ell]$, the syntheticizer can indeed guess with probability $1/\ell$ the value $f_s(a)$, so without the second type of query, nothing obvious would stop it from ignoring x, choosing an arbitrary a, and outputting a database of n copies of (a, b), where b is chosen uniformly at random from $[\ell]$. The intuition is now that such a synthetic database would give the wrong fraction — either zero or one, when the right answer should be about $1/2$ — on the truly random queries.

Formally, we have:

Theorem 9.1. 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 query family $\mathcal{Q}$ of size $\exp(\text{poly}(\kappa))$, a data universe $\mathcal{X}$ of size $O(n^{2+2a})$, and a distribution on databases of size n that is $(\mu, \alpha, \beta, 0, \mathcal{Q})$-*hard-to-syntheticize* (i.e., hard to syntheticize for worst-case queries) for $\alpha \leq 1/3$, $\beta \leq 1/10$ and $\mu = 1/40n^{1+a}$.

The above theorem shows hardness of sanitizing with synthetic data. Note, however, that when the query set is small one can always simply release noisy counts for every query. We conclude that sanitizing for small query classes (with large data universes) is a task that separates efficient syntheticizing from efficient synopsis generation (sanitization with arbitrary outputs).

9.2.1 Hardness results for general synopses

The hardness results of the previous section apply only to syntheticizers — offline mechanisms that create synthetic databases. There is a tight connection between hardness for more general forms of privacy-preserving offline mechanisms, which we have been calling *offline query release mechanisms* or synopsis generators, and the existence of *traitor tracing* schemes, a method of content distribution in which (short) key

strings are distributed to subscribers in such a way that a sender can broadcast encrypted messages that can be decrypted by any subscriber, and any useful "pirate" decoder constructed by a coalition of malicious subscribers can be traced to at least one colluder.

A (private-key, stateless) 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 ciphertext using any of the subscriber keys. The Trace algorithm gets the key bk and oracle access to a (pirate, stateless) decryption box, 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 $t = N$, tracing works even if all the subscribers join forces to try and create a pirate decoder. A more complete definition follows.

Definition 9.3. A scheme (Setup, Encrypt, Decrypt, Trace) as above is a *t-resilient traitor-tracing scheme* if (i) the ciphertexts it generates are semantically secure (roughly speaking, polynomial time algorithms cannot distinguish encryptions of 0 from encryptions of 1), and (ii) no polynomial time adversary A can "win" in the following game with non-negligible probability (over the coins of Setup, A, and Trace):

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 key bk and black-box access[5] 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 (even a weaker condition than creating a usable pirate decryption device), *and* the output of Trace is not in $\{i_1, \ldots, i_t\}$, meaning that the adversary avoided detection.

[5]Black-box access to an algorithm means that one has no access to the algorithm's internals; one can only feed inputs to the algorithm and observe its outputs.

The intuition for why traitor-tracing schemes imply hardness results for counting query release is as follows. Fix a traitor tracing scheme. We must describe databases and counting queries for which query release is computationally hard.

For any given $n = \kappa$, the database $x \in \{\{0,1\}^d\}^n$ will contain user keys from the traitor tracing scheme of a colluding set of n users; here d is the length of the decryption keys obtained when the Setup algorithm is run on input 1^κ. The query family $\mathcal{Q}_\kappa$ will have a query q_c for each possible ciphertext c asking "For what fraction of the rows $i \in [n]$ does c decrypt to 1 under the key in row i?" Note that, since every user can decrypt, if the sender distributes an encryption c of the bit 1, the answer will be 1: all the rows decrypt c to 1, so the fraction of such rows is 1. If instead the sender distributes an encryption c' of the bit 0, the answer will be 0: since no row decrypts c' to 1, the fraction of rows decrypting c' to 1 is 0. Thus, the exact answer to a query q_c, where c is an encryption of a 1-bit messages b, is b itself.

Now, suppose there were an efficient offline differentially private query release mechanism for queries in $\mathcal{Q}$. The colluders could use this algorithm to efficiently produce a synopsis of the database enabling a data analyst to efficiently compute approximate answers to the queries q_c. If these approximations are at all non-trivial, then the analyst can use these to correctly decrypt. That is, the colluders could use this to form a pirate decoder box. But traitor tracing ensures that, for any such box, the Trace algorithm can recover the key of at least one user, i.e., a row of the database. This violates differential privacy, contradicting the assumption that there is an efficient differentially private algorithm for releasing $\mathcal{Q}$.

This direction has been used to rule out the existence of efficient offline sanitizers for a particular class of $2^{\tilde{O}(\sqrt{n})}$ counting queries; this can be extended to rule out the existence of efficient *on-line* sanitizers answering $\tilde{\Theta}(n^2)$ counting queries drawn adaptively from a second (large) class.

The intuition for why hardness of offline query release for counting queries implies traitor tracing is that failure to protect privacy immediately yields some form of traceability; that is, the *difficulty* of providing an object that yields (approximate) functional equivalence for a set of

rows (decryption keys) while preserving privacy of each individual row (decryption key) — that is, the difficulty of producing an untraceable decoder — is precisely what we are looking for in a traitor tracing scheme.

In a little more detail, given a hard-to-sanitize database distribution and family of counting queries, a randomly drawn n-item database can act like a "master key," where the secret used to decrypt messages is the *counts* of random queries on this database. For a randomly chosen subset S of polylog(n) queries, a random set of polylog(n) rows drawn from the database (very likely) yields good approximation to all queries in S. Thus, individual user keys can be obtained by randomly partitioning the database into $n/\text{polylog}(n)$ sets of polylog(n) rows and assigning each set to a different user. These sets are large enough that with overwhelming probability their counts on a random collection of say polylog(n) queries are *all* close to the counts of the original database.

To complete the argument, one designs an encryption scheme in which decryption is equivalent to computing approximate counts on small sets of random queries. Since by definition a pirate decryption box can decrypt, the a pirate box can be used to compute approximate counts. If we view 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.

9.3 Polynomial time adversaries

Definition 9.4 (Computational Differential Privacy)**.** A randomized algorithm $C_\kappa : \mathcal{X}^n \to Y$ is *ε-computationally differentially private* if and only if for all databases x, y differing in a single row, and for all nonuniform polynomial (in κ) algorithms T,

$$\Pr[T(C_\kappa(x)) = 1] \leq e^\varepsilon \Pr[T(C_\kappa(y)) = 1] + \nu(\kappa),$$

where $\nu(\cdot)$ is any function that grows more slowly than the inverse of any polynomial and the agorithm C_κ runs in time polynomial in n, $\log|\mathcal{X}|$, and κ.

Intuitively, this says that if the adversary is restricted to polynomial time then computationally differentially private mechanisms provide the same degree of privacy as do $(\varepsilon, \nu(\kappa))$-differentially private algorithms. In general there is no hope of getting rid of the $\nu(\kappa)$ term; for example, when encryption is involved there is always some (neglibly small) chance of guessing the decryption key.

Once we assume the adversary is restricted to polynomial time, we can use the powerful techniques of *secure multiparty computation* to provide *distributed* online query release algorithms, replacing the trusted server with a distributed protocoal that simulates a trusted curator. Thus, for example, a set of hospitals, each holding the data of many patients, can collaboratively carry out statistical analyses of the union of their patients, while ensuring differential privacy for each patient. A more radical implication is that individuals can maintain their own data, opting in or out of each specific statistical query or study, all the while ensuring differential privacy of their own data.

We have already seen one distributed solution, at least for the problem of computing a sum of n bits: randomized response. This solution requires no computational assumptions, and has an expected error of $\Theta(\sqrt{n})$. In contrast, the use of cryptographic assumptions permits much more accurate and extensive analyses, since by simulating the curator it can run a distributed implementation of the Laplace mechanism, which has constant expected error.

This leads to the natural question of whether there is some other approach, not relying on cryptographic assumptions, that yields better accuracy in the distributed setting than does randomized response. Or more generally, is there a separation between what can be accomplished with computational differential privacy and what can be achieved with "traditional" differential privacy? That is, does cryptography provably buy us something?

In the multiparty setting the answer is yes. Still confining our attention to summing n bits, we have:

Theorem 9.2. For $\varepsilon < 1$, every n-party $(\varepsilon, 0)$-differentially private protocol for computing the sum of n bits (one per party) incurs error $\Omega(n^{1/2})$ with high probability.

A similar theorem holds for (ε, δ)-differential privacy provided $\delta \in o(1/n)$.

Proof. (sketch) Let $X_1, \ldots, X_n$ be uniform independent bits. The transcript T of the protocol is a random variable $T = T(P_1(X_1), \ldots, P_n(X_n)$, where for $i \in [n]$ the protocol of player i is denoted P_i. Conditioned on $T = t$, the bits $X_1, \ldots, X_n$ are still independent bits, each with bias $O(\varepsilon)$. Further, by differential privacy, the uniformity of the X_i, and Bayes' Law we have:

$$\frac{\Pr[X_i = 1|T = t]}{\Pr[X_i = 0|T = t]} = \frac{\Pr[T = t|X_i = 1]}{\Pr[T = t|X_i = 0]} \leq e^{\varepsilon} < 1 + 2\varepsilon.$$

To finish the proof we note that the sum of n independent bits, each with constant bias, falls outside any interval of size $o(\sqrt{n})$ with high probability. Thus, with high probability, the sum $\sum_i X_i$ is not in the interval $[\text{output(T)} - o(n^{1/2}), \text{output(T)} + o(n^{1/2})]$. □

A more involved proof shows a separation between computational differential privacy and ordinary differential privacy even for the two-party case. It is a fascinating open question whether computational assumptions buy us anything in the case of the trusted curator. Initial results are negative: for *small* numbers of *real-valued* queries, i.e., for a number of queries that does not grow with the security parameter, there is a natural class of utility measures, including L_p distances and mean-squared errors, for which any computationally private mechanism can be converted to a statistically private mechanism that is roughly as efficient and achieves almost the same utility.

9.4 Bibliographic notes

The negative results for polynomial time bounded curators and the connection to traitor tracing are due to Dwork et al. [28]. The connection to traitor tracing was further investigated by Ullman [82], who showed that, assuming the existence of 1-way functions, it is computationally hard to answer $n^{2+o(1)}$ arbitrary linear queries with differential privacy (even if without privacy the answers are easy to compute). In "Our Data, Ourselves," Dwork, Kenthapadi, McSherry, Mironov, and

Naor considered a distributed version of the precursor of differential privacy, using techniques from secure function evaluation in place of the trusted curator [21]. A formal study of *computational* differential privacy was initiated in [64], and the separation between the accuracy that can be achieved with $(\varepsilon, 0)$-differential privacy in the multiparty and single curator cases in Theorem 9.2 is due to McGregor et al. [58]. The initial results regarding whether computational assumptions on the adversary buys anything in the case of a trusted curator are due to Groce et al. [37].

Construction of pseudorandom functions from any one-way function is due to Håstad et al. [40].

10

Differential Privacy and Mechanism Design

One of the most fascinating areas of game theory is mechanism design, which is the science of designing incentives to get people to do what you want them to do. Differential privacy has proven to have interesting connections to mechanism design in a couple of unexpected ways. It provides a tool to quantify and control privacy loss, which is important if the people the mechanism designer is attempting to manipulate care about privacy. However, it also provides a way to limit the sensitivity of the outcome of a mechanism to the choices of any single person, which turns out to be a powerful tool even in the absence of privacy concerns. In this section, we give a brief survey of some of these ideas.

Mechanism Design is the problem of *algorithm design* when the inputs to the algorithm are controlled by individual, self-interested agents, rather than the algorithm designer himself. The algorithm maps its reported inputs to some outcome, over which the agents have preferences. The difficulty is that the agents may mis-report their data if doing so will cause the algorithm to output a different, preferred outcome, and so the mechanism designer must design the algorithm so that the agents are always incentivized to report their true data.

The concerns of mechanism design are very similar to the concerns of private algorithm design. In both cases, the inputs to the algorithm are thought of as belonging to some third party[1] which has preferences over the outcome. In mechanism design, we typically think of individuals as getting some explicit value from the outcomes of the mechanism. In private algorithm design, we typically think of the individual as experiencing some explicit harm from (consequences of) outcomes of the mechanism. Indeed, we can give a utility-theoretic definition of differential privacy which is equivalent to the standard definition, but makes the connection to individual utilities explicit:

Definition 10.1. An algorithm $A : \mathbb{N}^{|\mathcal{X}|} \to R$ is ϵ-differentially private if for every function $f : R \to \mathbb{R}_+$, and for every pair of neighboring databases $x, y \in \mathbb{N}^{|\mathcal{X}|}$:

$$\exp(-\epsilon)\mathbb{E}_{z\sim A(y)}[f(z)] \leq \mathbb{E}_{z\sim A(x)}[f(z)] \leq \exp(\epsilon)\mathbb{E}_{z\sim A(y)}[f(z)].$$

We can think of f as being some function mapping outcomes to an arbitrary agent's utility for those outcomes. With this interpretation, a mechanism is ϵ-differentially private, if for every agent it promises that their participation in the mechanism cannot affect their expected future utility by more than a factor of $\exp(\epsilon)$ *independent of what their utility function might be.*

Let us now give a brief definition of a problem in mechanism design. A mechanism design problem is defined by several objects. There are n agents $i \in [n]$, and a set of outcomes $\mathcal{O}$. Each agent has a type, $t_i \in \mathcal{T}$ which is known only to her, and there is a utility function over outcomes $u : \mathcal{T} \times \mathcal{O} \to [0, 1]$. The utility that agent i gets from an outcome $o \in \mathcal{O}$ is $u(t_i, o)$, which we will often abbreviate as $u_i(o)$. We will write $t \in \mathcal{T}^n$ to denote vectors of all n agent types, with t_i denoting the type of agent i, and $t_{-i} \equiv (t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n)$ denoting the vector of types of all agents *except* agent i. The type of an agent i completely specifies her utility over outcomes — that is, two agents $i \neq j$ such that $t_i = t_j$ will evaluate each outcome identically: $u_i(o) = u_j(o)$ for all $o \in \mathcal{O}$.

[1] In the privacy setting, the database administrator (such as a hospital) might already have access to the data itself, but is nevertheless acting so as to protect the interests of the agents who own the data when it endeavors to protect privacy.

A mechanism M takes as input a set of reported types, one from each player, and selects an outcome. That is, a mechanism is a mapping $M : \mathcal{T}^n \to \mathcal{O}$. Agents will choose to report their types strategically so as to optimize their utility, possibly taking into account what (they think) the other agents will be doing. In particular, they need not report their true types to the mechanism. If an agent is always incentivized to report some type, no matter what her opponents are reporting, then reporting that type is called a *dominant strategy*. If reporting one's true type is a dominant strategy for every agent, then the mechanism is called *truthful*, or equivalently, *dominant strategy truthful*.

Definition 10.2. Given a mechanism $M : \mathcal{T}^n \to \mathcal{O}$, truthful reporting is an ϵ-approximate *dominant strategy* for player i if for every pair of types $t_i, t_i' \in T$, and for every vector of types t_{-i}:

$$u(t_i, M(t_i, t_{-i})) \geq u(t_i, M(t_i', t_{-i})) - \epsilon.$$

If truthful reporting is an ϵ-approximate dominant strategy for every player, we say that M is ϵ-approximately dominant strategy truthful. If $\epsilon = 0$, then M is *exactly truthful*.

That is, a mechanism is truthful if no agent can improve her utility by misrepresenting her type, no matter what the other players report.

Here we can immediately observe a syntactic connection to the definition of differential privacy. We may identify the type space T with the data universe X. The input to the mechanism therefore consists of a database of size n, consisting of the reports of each agent. In fact, when an agent is considering whether she should truthfully report her type t_i or lie, and misreport her type as t_i', she is deciding which of two databases the mechanism should receive: $(t_1, \ldots, t_n)$, or $(t_1, \ldots, t_{i-1}, t_i', t_{i+1}, \ldots, t_n)$. Note that these two databases differ only in the report of agent i! That is, they are *neighboring databases*. Thus, differential privacy gives a guarantee of approximate truthfulness!

10.1 Differential privacy as a solution concept

One of the starting points for investigating the connection between differential privacy and game theory is observing that differential privacy

is a *stronger* condition than approximate truthfulness. Note that for $\epsilon \leq 1$, $\exp(\epsilon) \leq 1 + 2\epsilon$ and so the following proposition is immediate.

Proposition 10.1. If a mechanism M is ϵ-differentially private, then M is also 2ϵ-approximately dominant strategy truthful.

As a solution concept, this has several robustness properties that strategy proof mechanisms do not. By the composition property of differential privacy, the composition of 2 ϵ-differentially private mechanisms remains 4ϵ-approximately dominant strategy truthful. In contrast, the incentive properties of general strategy proof mechanisms may not be preserved under composition.

Another useful property of differential privacy as a solution concept is that it generalizes to group privacy: suppose that t and $t' \in \mathcal{T}^n$ are not neighbors, but instead differ in k indices. Recall that by group privacy we then have for any player i: $\mathbb{E}_{o\sim M(t)}[u_i(o)] \leq \exp(k\epsilon)\mathbb{E}_{o\sim M(t')}[u_i(o)]$. That is, changes in up to k types changes the expected output by at most $\approx (1+k\epsilon)$, when $k \ll 1/\epsilon$. Therefore, differentially private mechanisms make truthful reporting a $2k\epsilon$-approximate dominant strategy *even for coalitions of k agents* — i.e., differential privacy automatically provides robustness to collusion. Again, this is in contrast to general dominant-strategy truthful mechanisms, which in general offer no guarantees against collusion.

Notably, differential privacy allows for these properties in very general settings *without the use of money!* In contrast, the set of exactly dominant strategy truthful mechanisms when monetary transfers are not allowed is extremely limited.

We conclude with a drawback of using differential privacy as a solution concept as stated: not only is truthfully reporting one's type an approximate dominant strategy, *any report* is an approximate dominant strategy! That is, differential privacy makes the outcome approximately independent of any single agent's report. In some settings, this shortcoming can be alleviated. For example, suppose that M is a differentially private mechanism, but that agent utility functions are defined to be functions both of the outcome of the mechanism, *and* of the reported type t_i' of the agent: formally, we view the outcome space as $\mathcal{O}' = \mathcal{O} \times T$. When the agent reports type t_i' to the mechanism, and

the mechanism selects outcome $o \in \mathcal{O}$, then the utility experienced by the agent is controlled by the outcome $o' = (o, t_i')$. Now consider the underlying utility function $u : T \times \mathcal{O}' \to [0, 1]$. Suppose we have that *fixing* a selection o of the mechanism, truthful reporting is a dominant strategy — that is, for all types t_i, t_i', and for all outcomes $o \in \mathcal{O}$:

$$u(t_i, (o, t_i)) \geq u(t_i, (o, t_i')).$$

Then it remains the fact that truthful reporting to an ϵ-differentially private mechanism $M : T^n \to \mathcal{O}$ remains a 2ϵ approximate dominant strategy, because for any misreport t_i' that player i might consider, we have:

$$\begin{aligned} u(t_i, (M(t), t_i)) &= \mathbb{E}_{o \sim M(t)}[u(t_i, (o, t_i))] \\ &\geq (1 + 2\epsilon)\mathbb{E}_{o \sim M(t_i', t_{-i})}[u(t_i, (o, t_i))] \\ &\geq \mathbb{E}_{o \sim M(t_i', t_{-i})}[u(t_i, (o, t_i'))] \\ &= u(t_i, (M(t_i', t_{-i}), t_i')). \end{aligned}$$

However, we no longer have that every report is an approximate dominant strategy, because player i's utility can depend arbitrarily on $o' = (o, t_i')$, and only o (and not player i's report t_i' itself) is differentially private. This will be the case in all examples we consider here.

10.2 Differential privacy as a tool in mechanism design

In this section, we show how the machinery of differential privacy can be used as a tool in designing novel mechanisms.

10.2.1 Warmup: digital goods auctions

To warm up, let us consider a simple special case of the first application of differential privacy in mechanism design. Consider a *digital goods auction*, i.e., one where the seller has an unlimited supply of a good with zero marginal cost to produce, for example a piece of software or other digital media. There are n unit demand buyers for this good, each with unknown valuation $v_i \in [0, 1]$. Informally, the valuation v_i of a bidder i represents the maximum amount of money that buyer i

would be willing to pay for a good. There is no prior distribution on the bidder valuations, so a natural revenue benchmark is the revenue of the *best fixed price*. At a price $p \in [0,1]$, each bidder i with $v_i \geq p$ will buy. Therefore the total revenue of the auctioneer is

$$\mathrm{Rev}(p, v) = p \cdot |\{i : v_i \geq p\}|.$$

The optimal revenue is the revenue of the best fixed price: $\mathrm{OPT} = \max_p \mathrm{Rev}(p, v)$. This setting is well studied: the best known result for exactly dominant strategy truthful mechanisms is a mechanism which achieves revenue at least $\mathrm{OPT} - O(\sqrt{n})$.

We show how a simple application of the exponential mechanism achieves revenue at least $\mathrm{OPT} - O\left(\frac{\log n}{\epsilon}\right)$. That is, the mechanism trades exact for approximate truthfulness, but achieves an exponentially better revenue guarantee. Of course, it also inherits the benefits of differential privacy discussed previously, such as resilience to collusion, and composability.

The idea is to select a price from the exponential mechanism, using as our "quality score" the revenue that this price would obtain. Suppose we choose the range of the exponential mechanism to be $\mathcal{R} = \{\alpha, 2\alpha, \ldots, 1\}$. The size of the range is $|\mathcal{R}| = 1/\alpha$. What have we lost in potential revenue if we restrict ourselves to selecting a price from $\mathcal{R}$? It is not hard to see that

$$\mathrm{OPT}_{\mathcal{R}} \equiv \max_{p \in \mathcal{R}} \mathrm{Rev}(p, v) \geq \mathrm{OPT} - \alpha n.$$

This is because if p^* is the price that achieves the optimal revenue, and we use a price p such that $p^* - \alpha \leq p \leq p^*$, every buyer who bought at the optimal price continues to buy, and provides us with at most α less revenue per buyer. Since there are at most n buyers, the total lost revenue is at most αn.

So how do we parameterize the exponential mechanism? We have a family of discrete ranges $\mathcal{R}$, parameterized by α. For a vector of values v and a price $p \in \mathcal{R}$, we define our quality function to be $q(v, p) = \mathrm{Rev}(v, p)$. Observe that because each value $v_i \in [0,1]$, we can restrict attention to prices $p \leq 1$ and hence, the *sensitivity* of q is $\Delta = 1$: changing one bidder valuation can only change the revenue at a fixed

price by at most $v_i \leq 1$. Therefore, if we require ϵ-differential privacy, by Theorem 3.11, we get that with high probability, the exponential mechanism returns some price p such that

$$\text{Rev}(p, v) \geq (\text{OPT} - \alpha n) - O\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\alpha}\right)\right).$$

Choosing our discretization parameter α to minimize the two sources of error, we find that this mechanism with high probability finds us a price that achieves revenue

$$\text{Rev}(p, v) \geq \text{OPT} - O\left(\frac{\log n}{\epsilon}\right).$$

What is the right level to choose for the privacy parameter ϵ? Note that here, we do not necessarily view privacy itself as a goal of our computation. Rather, ϵ is a way of trading off the revenue guarantee with an upper bound on agent's incentives to deviate. In the literature on large markets in economics, a common goal when exact truthfulness is out of reach is "asymptotic truthfulness" – that is, the maximum incentive that any agent has to deviate from his truthful report tend to 0 as the size of the market n grows large. To achieve a result like that here, all we need to do is set ϵ to be some diminishing function in the number of agents n. For example, if we take $\epsilon = 1/\log(n)$, then we obtain a mechanism that is asymptotically exactly truthful (i.e., as the market grows large, the approximation to truthfulness becomes exact). We can also ask what our approximation to the optimal revenue is as n grows large. Note that our approximation to the optimal revenue is only additive, and so even with this setting of ϵ, we can still guarantee revenue at least $(1 - o(1))\text{OPT}$, so long as OPT grows more quickly than $\log(n)^2$ with the size of the population n.

Finally, notice that we could make the reported value v_i of each agent i binding. In other words, we could allocate an item to agent i and extract payment of the selected posted price p whenever $v_i \geq p$. If we do this, the mechanism is approximately truthful, because the price is picked using a differentially private mechanism. Additionally, it is not the case that *every* report is an approximate dominant strategy: if an agent over-reports, she may be forced to buy the good at a price higher than her true value.

10.2.2 Approximately truthful equilibrium selection mechanisms

We now consider the problem of approximately truthful equilibrium selection. We recall the definition of a *Nash Equilibrium*: Suppose each player has a set of actions $\mathcal{A}$, and can choose to play any action $a_i \in \mathcal{A}$. Suppose, moreover, that *outcomes* are merely choices of actions that the agents might choose to play, and so agent utility functions are defined as $u : \mathcal{T} \times \mathcal{A}^n \to [0, 1]$. Then:

Definition 10.3. A set of actions $a \in \mathcal{A}^n$ is an ϵ-approximate Nash equilibrium if for all players i and for all actions a'_i:

$$u_i(a) \geq u_i(a'_i, a_{-i}) - \epsilon$$

In other words, every agent is simultaneously playing an (approximate) best response to what the other agents are doing, assuming they are playing according to a.

Roughly speaking, the problem is as follows: suppose we are given a game in which each player knows their own payoffs, but not others' payoffs (i.e., the players do not know what the types are of the other agents). The players therefore do not know the equilibrium structure of this game. Even if they did, there might be multiple equilibria, with different agents preferring different equilibria. Can a mechanism offered by an intermediary incentivize agents to truthfully report their utilities and follow the equilibrium it selects?

For example, imagine a city in which (say) Google Navigation is the dominant service. Every morning, each person enters their starting point and destination, receives a set of directions, and chooses his/her route according to those directions. Is it possible to design a navigation service such that: Each agent is incentivized to both (1) report truthfully, and (2) then follow the driving directions provided? Both misreporting start and end points, and truthfully reporting start and end points, but then following a different (shorter) path are to be disincentivized.

Intuitively, our two desiderata are in conflict. In the commuting example above, if we are to guarantee that every player is incentivized to truthfully follow their suggested route, then we must compute an