12

Additional Models

So far, we have made some implicit assumptions about the model of private data analysis. For example, we have assumed that there is some trusted curator who has direct access to the private dataset, and we have assumed that the adversary only has access to the output of the algorithm, not to any of its internal state during its execution. But what if this is not the case? What if we trust no one to look at our data, even to perform the privacy preserving data analysis? What if some hacker might gain access to the internal state of the private algorithm while it is running? In this section, we relax some of our previously held assumptions and consider these questions.

In this section we describe some additional computational models that have received attention in the literature.

- The *local model* is a generalization of randomized response (see Section 2), and is motivated by situations in which individuals do not trust the curator with their data. While this lack of trust can be addressed using secure multiparty computation to simulate the role played by the trusted curator, there are also some techniques that do not require cryptography.

The next two models consider streams of *events*, each of which may be associated with an individual. For example, an event may be a search by a particular person on an arbitrary term. In a given event stream, the (potentially many) events associated with a given individual can be arbitrarily interleaved with events associated with other individuals.

- In *pan-privacy* the curator is trusted, but may be subject to compulsory non-private data release, for example, because of a subpoena, or because the entity holding the information is purchased by another, possibly less trustworthy, entity. Thus, in pan-privacy the *internal state* of the algorithm is also differentially private, as is the joint distribution of the internal state and the outputs.
- The *continual observation* model addresses the question of maintaining privacy when the goal is to continually monitor and report statistics about events, such as purchases of over-the-counter medications that might be indicative of an impending epidemic. Some work addresses pan-privacy under continual observation.

12.1 The local model

So far, we have considered a *centralized* model of data privacy, in which there exists a database administrator who has direct access to the private data. What if there is instead no trusted database administrator? Even if there is a suitable trusted party, there are many reasons not to want private data aggregated by some third party. The very existence of an aggregate database of private information raises the possibility that at some *future time*, it will come into the hands of an untrusted party, either maliciously (via data theft), or as a natural result of organizational succession. A superior model — from the perspective of the owners of private data — would be a local model, in which agents could (randomly) answer questions in a differentially private manner about their own data, without ever sharing it with anyone else. In the context of predicate queries, this seems to severely limit the expressivity of a private mechanism's interaction with the data: The mechanism can ask each user whether or not her data satisfies a given predicate, and the user may flip a coin, answering truthfully only with slightly higher probability than answering falsely. In this model what is possible?

The local privacy model was first introduced in the context of learning. The local privacy model formalizes randomized response: there is no central database of private data. Instead, each individual maintains possession of their own data element (a database of size 1), and answers questions about it only in a differentially private manner. Formally, the database $x \in \mathbb{N}^{|\mathcal{X}|}$ is a collection of n elements from some domain $\mathcal{X}$, and each $x_i \in x$ is held by an individual.

Definition 12.1 (Local Randomizer). An ε-local randomizer $R : \mathcal{X} \to W$ is an ε-differentially private algorithm that takes as input a database of size $n = 1$.

In the local privacy model, algorithms may interact with the database only through a local randomizer oracle:

Definition 12.2 (LR Oracle). An LR oracle $LR_D(\cdot, \cdot)$ takes as input an index $i \in [n]$ and an ε-local randomizer R and outputs a random value $w \in W$ chosen according to the distribution $R(x_i)$, where $x_i \in D$ is the element held by the ith individual in the database.

Definition 12.3 ((Local Algorithm)). An algorithm is ε-local if it accesses the database D via the oracle LR_D, with the following restriction: If $LR_D(i, R_1), \ldots, LR_D(i, R_k)$ are the algorithm's invocations of LR_D on index i, where each R_J is an ε_j-local randomizer, then $\varepsilon_1 + \cdots + \varepsilon_k \leq \varepsilon$.

Because differential privacy is composable, it is easy to see that ε-local algorithms are ε-differentially private.

Observation 12.1. ε-local algorithms are ε-differentially private.

That is to say, an ε-local algorithm interacts with the data using only a sequence of ε-differentially private algorithms, each of which computes only on a database of size 1. Because nobody other than its owner ever touches any piece of private data, the local setting is far more secure: it does not require a trusted party, and there is no central party who might be subject to hacking. Because even the algorithm never sees private data, the internal state of the algorithm is always differentially private as well (i.e., local privacy implies pan privacy, described in the next section). A natural question is how restrictive the local privacy model is. In this section, we merely informally discuss results. The interested reader can follow the bibliographic references at the end of this section for more information. We note that an alternative name for the local privacy model is the *fully distributed* model.

We recall the definition of the statistical query (SQ) model, introduced in Section 11. Roughly speaking, given a database x of size n, the statistical query model allows an algorithm to access this database by making a polynomial (in n) number of noisy linear queries to the database, where the error in the query answers is some inverse polynomial in n. Formally:

Definition 12.4. A *statistical query* is some function $\phi : \mathcal{X} \times \{0,1\} \to [0,1]$. A *statistical query oracle* for a distribution over labeled examples $\mathcal{D}$ with tolerance τ is an oracle $\mathcal{O}^{\tau}_{\mathcal{D}}$ such that for every statistical query ϕ:

$$\left|\mathcal{O}^{\tau}_{\mathcal{D}}(\phi) - \mathbb{E}_{(x,y)\sim\mathcal{D}}[\phi(x,y)]\right| \leq \tau$$

In other words, an SQ oracle takes as input a statistical query ϕ, and outputs some value that is guaranteed to be within $\pm\tau$ of the expected value of ϕ on examples drawn from $\mathcal{D}$.

Definition 12.5. An algorithm A is said to SQ-learn a class of functions C if for every $\alpha, \beta > 0$ there exists an $m = \mathrm{poly}(d, 1/\alpha, \log(1/\beta))$ such that A makes at most m queries of tolerance $\tau = 1/m$ to $\mathcal{O}^{\tau}_{\mathcal{D}}$, and with probability $1-\beta$, outputs a hypothesis $f \in C$ such that:

$$\mathrm{err}(f, \mathcal{D}) \leq \min_{f^* \in C} \mathrm{err}(f^*, \mathcal{D}) + \alpha$$

More generally, we can talk about an algorithm (for performing any computation) as operating in the SQ model if it accesses the data only through an SQ oracle:

Definition 12.6. An algorithm A is said to operate in the SQ model if there exists an m such that A makes at most m queries of tolerance $\tau = 1/m$ to $\mathcal{O}^{\tau}_{D}$, and does not have any other access to the database. A is efficient if m is polynomial in the size of the database, D.

It turns out that up to polynomial factors in the size of the database and in the number of queries, any algorithm that can be implemented in the SQ model can be implemented and analyzed for privacy in the local privacy model, and vice versa. We note that there is a distinction between an algorithm being implemented in the SQ model, and its privacy analysis being carried out in the local model: almost all of the algorithms that we have presented in the end access the data using noisy linear queries, and so can be thought of as acting in the SQ model. However, their privacy guarantees are analyzed in the centralized model of data privacy (i.e., because of some "global" part of the analysis, as in the sparse vector algorithm).

In the following summary, we will also recall the definition of PAC learning, also introduced in Section 11:

Definition 12.7. An algorithm A is said to PAC-learn a class of functions C if for every $\alpha, \beta > 0$, there exists an $m = \mathrm{poly}(d, 1/\alpha, \log(1/\beta))$ such that for every distribution $\mathcal{D}$ over labeled examples, A takes as input m labeled examples drawn from $\mathcal{D}$ and outputs a hypothesis $f \in C$ such that with probability $1 - \beta$:

$$\mathrm{err}(f, \mathcal{D}) \leq \min_{f^* \in C} \mathrm{err}(f^*, \mathcal{D}) + \alpha$$

If $\min_{f^* \in C} \mathrm{err}(f^*, \mathcal{D}) = 0$, the learner is said to operate in the *realizable* setting (i.e., there exists some function in the class which perfectly labels the data). Otherwise, the learner is said to operate in the *agnostic* setting. If A also has run time that is polynomial in $d, 1/\alpha$, and $\log(1/\beta)$, then the learner is said to be *efficient.* If there is an algorithm which PAC-learns C, then C is said to be PAC-learnable. Note that the main distinction between an SQ learning algorithm and a PAC learning algorithm, is that the PAC learning algorithm gets direct access to the database of examples, whereas the SQ learning algorithm only has access to the data through a noisy SQ oracle.

What follows is some of our understanding of the limitations of the SQ model and problems which separate it from the centralized model of data privacy.

1. A single sensitivity-1 query can be answered to error $O(1)$ in the centralized model of data privacy using the Laplace mechanism, but requires error $\Theta(\sqrt{n})$ in the local data privacy model.
2. The set of function classes that we can (properly) learn in the local privacy model is exactly the set of function classes that we can properly learn in the SQ model (up to polynomial factors in the database size and query complexity of the algorithm). In contrast, the set of things we can (properly or agnostically) learn in the centralized model corresponds to the set of things we can learn in the PAC model. SQ learning is strictly weaker, but this is not a huge handicap, since parity functions are essentially the only interesting class that is PAC learnable but not SQ learnable. We remark that we refer explicitly to proper learning here (meaning the setting in which there is some function in the class which perfectly labels the data). In the PAC model there is no information theoretic difference between proper and agnostic learning, but in the SQ model the difference is large: see the next point.
3. The set of queries that we can release in the local privacy model are exactly those queries that we can agnostically learn in the SQ model. In contrast, the set of things we can release in the centralized model corresponds to the set of things we can agnostically learn in the PAC model. This is a much bigger handicap — even conjunctions (i.e., marginals) are not agnostically learnable in the SQ model. This follows from the information theoretic reduction from agnostic learning (i.e., *distinguishing*) to query release that we saw in Section 5 using the iterative construction mechanism.

We note that if we are only concerned about computationally bounded adversaries, then in principle distributed agents can use *secure multiparty computation* to simulate private algorithms in the centralized setting. While this does not actually give a differential privacy guarantee, the result of such simulations will be indistinguishable from the result of differentially private computations, from the point of view of a computationally bounded adversary. However, general secure multiparty computation protocols typically require huge amounts of message passing (and hence sometimes have unreasonably large run times), whereas algorithms in the local privacy model tend to be extremely simple.

12.2 Pan-private streaming model

The goal of a pan-private algorithm is to remain differentially private even against an adversary that can, on rare occasions, observe the algorithm's internal state. Intrusions can occur for many reasons, including hacking, subpoena, or *mission creep*, when data collected for one purpose are used for a different purpose ("Think of the children!"). Pan-private streaming algorithms provide protection against all of these. Note that ordinary streaming algorithms do *not* necessarily provide privacy against intrusions, as even a low-memory streaming algorithm can hold a small number of data items in memory, which would be completely exposed in an intrusion. On the technical side, intrusions can be *known* to the curator (subpoena) or unknown (hacking). These can have very different effects, as a curator aware of an intrusion can take protective measures, such as re-randomizing certain variables.

12.2.1 Definitions

We assume a data stream of unbounded length composed of elements in a universe $\mathcal{X}$. It may be helpful to keep in mind as motivation data analysis on a query stream, in which queries are accompanied by the IP address of the issuer. For now, we ignore the query text itself; the universe $\mathcal{X}$ is the universe of potential IP addresses. Thus, intuitively, *user-level* privacy protects the presence or absence of an IP address in the stream, indpendent of the number of times it arises, should it actually be present at all. In contrast, *event-level* privacy merely protects the privacy of individual accesses. For now, we focus on user-level privacy.

As usual in differentially private algorithms, the adversary can have arbitrary control of the input stream, and may have arbitrary auxiliary knowledge obtained from other sources. It can also have arbitrary computational power.

We assume the algorithm runs until it receives a special signal, at which point it produces (observable) outputs. The algorithm may optionally continue to run and produce additional outputs later, again in response to a special signal. Since outputs are obvservable we do not provide privacy for the special signals.

A streaming algorithm experiences a sequence of internal states. and produces a (possibly unbounded) sequence of outputs. Let I denote the set of possible internal states of the algorithm, and σ the set of possible output sequences. We assume that the adversary can only observe internal states and the output sequence; it cannot see the data in the stream (although it may have auxiliary knowledge about some of these data) and it has no access to the *length* of the input sequence.

Definition 12.8 ($\mathcal{X}$-Adjacent Data Streams)**.** We think of data streams as being of unbounded length; *prefixes* have finite length. Data streams S and S' are $\mathcal{X}$-adjacent if they differ only in the presence or absence of *all* occurrences of a single element $u \in \mathcal{X}$. We define $\mathcal{X}$-adjacency for stream prefixes analogously.

User-Level Pan-Privacy. An algorithm **Alg** mapping data stream prefixes to the range $\mathrm{I} \times \sigma$, is *pan-private against a single intrusion* if for all sets $\mathrm{I}' \subseteq \mathrm{I}$ of internal states and $\sigma' \subseteq \sigma$ of output sequences, and for all pairs of adjacent data stream prefixes S, S'

$$\Pr[\mathbf{Alg}(S) \in (\mathrm{I}', \sigma')] \leq e^{\varepsilon} \Pr[\mathbf{Alg}(S') \in (\mathrm{I}', \sigma')],$$

where the probability spaces are over the coin flips of the algorithm **Alg**.

This definition speaks only of a single intrusion. For multiple intrusions we must consider interleavings of observations of internal states and outputs.

The relaxation to *event-level privacy* is obtained by modifying the notion of adjacency so that, roughly speaking, two streams are event-adjacent if they differ in a single instance of a single element in $\mathcal{X}$; that is, one instance of one element is deleted/added. Clearly, event-level privacy is a much weaker guarantee than user-level privacy.

Remark 12.1. If we assume the existence of a very small amount of secret storage, not visible to the adversary, then many problems for which we have been unable to obtain pan-private solutions have (non-pan-) private streaming solutions. However, the *amount* of secret storage is not so important as its *existence*, since secret storage is vulnerable to the social pressures against which pan-privacy seeks to protect the data (and the curator).

Pan-Private Density Estimation. Quite surprisingly, pan-privacy can be achieved even for *user-level* privacy of many common streaming computations. As an example, consider the problem of *density estimation*: given a universe $\mathcal{X}$ of data elements and a stream σ, the goal is to estimate the fraction of $\mathcal{X}$ that acutally appears in the stream. For example, the universe consists of all teenagers in a given community (represented by IP addresses), and the goal is to understand what fraction visit the Planned Parenthood website.

Standard low-memory streaming solutions for density estimation involve recording the results of deterministic computations of at least some input items, an approach that is inherently not pan-private. Here is a simple, albeit high-memory, solution inspired by randomized response. The algorithm maintains a bit b_a for each IP address a (which may appear any number of times in the stream), initialized uniformly at random. The stream is processed one element at a time. On input a the algorithm flips a bit biased to 1; that is, the biased bit will take value 0 with probability $1/2 - \varepsilon$, and value 1 with probability $1/2 + \varepsilon$. The algorithm follows this procedure independent of the number of times IP address a appears in the data stream. This algorithm is $(\varepsilon, 0)$-differentially private. As with randomized response, we can estimate the fraction of "real" 1's by $z = 2(y - |\mathcal{X}|/2)/|\mathcal{X}|$, where y is the actual number of 1's in the table after the stream is processed. To ensure pan-privacy, the algorithm publishes a noisy version of z. As with randomized response, the error will be on the order of $1/\sqrt{|\mathcal{X}|}$, yielding meaningful results when the density is high.

Other problems enjoying user-level pan-private algorithms include:

- Estimating, for any t, the fraction of elements appearing exactly t times;

- Estimating the *t-cropped mean*: roughly, the average, over all elements, of the minimum of t and the number of occurrences of the element in the data stream;
- Estimating the fraction of k-heavy hitters (elements of $\mathcal{X}$ that appear at least k times in the data stream).

Variants of these problems can also be defined for *fully dynamic* data, in which counts can be decremented as well as incremented. For example, density estimation (what fraction appeared in the stream?) becomes "How many (or what fraction) of elements have a (net) count equal to zero?" These, too, can be solved with user-level pan-privacy, using differentially private variations of *sketching* techniques from the streaming literature.

12.3 Continual observation

Many applications of data analysis involve repeated computations, either because the entire goal is one of monitoring of, for example, traffic conditions, search trends, or incidence of influenza. In such applications the system is required to continually produce outputs. We therefore need techniques for achieving *differential privacy under continual observation.*

As usual, differential privacy will require having essentially the same distribution on outputs for each pair of adjacent databases, but how should we define adjacency in this setting? Let us consider two example scenarios.

Suppose the goal is to monitor public health by analyzing statistics from an H1N1 self-assessment Web site.[1] Individuals can interact with the site to learn whether symptoms they are experiencing may be indicative of the H1N1 flu. The user fills in some demographic data (age, zipcode, sex), and responds to queries about his symptoms (fever over 100.4°F?, sore throat?, duration of symptoms?). We would expect a given individual to interact very few times with the H1N1 self-assessment site (say, if we restrict our attention to a six-month

[1] https://h1n1.cloudapp.net provided such a service during the winter of 2010; user-supplied data were stored for analysis with the user's consent.

period). For simplicity, let us say this is just once. In such a setting, it is sufficient to ensure *event-level* privacy, in which the privacy goal is to hide the presence or absence of a single event (interaction of one user with the self-assessment site).

Suppose again that the goal is to monitor public health, this time by analyzing search terms submitted to a medical search engine. Here it may no longer be safe to assume an individual has few interactions with the Web site, even if we restrict attention to a relatively short period of time. In this case we would want *user-level* privacy, ensuring that the entire set of a user's search terms is protected simultaneously.

We think of continual observation algorithms as taking steps at discrete time intervals; at each step the algorithm receives an input, computes, and produces output. We model the data as arriving in a stream, at most one data element in each time interval. To capture the fact that, in real life, there are periods of time in which nothing happens, null events are modeled by a special symbol in the data stream. Thus, the intuitive notion of "t time periods" corresponds to processing a sequence of t elements in the stream.

For example, the motivation behind the counter primitive below is to count the number of times that something has occurred since the algorithm was started (the counter is very general; we don't specify *a priori* what it is counting). This is modeled by an input stream over $\{0, 1\}$. Here, "0" means "nothing happened," "1" means the event of interest occurred, and for $t = 1, 2, \ldots, T$ the algorithm outputs an approximation to the number of 1s seen in the length t prefix of the stream.

There are three natural options:

1. Use randomized response for each time period and add this randomized value to the counter;
2. Add noise distributed according to $\mathrm{Lap}(1/\varepsilon)$ to the true value for each time step and add this perturbed value to the counter;
3. Compute the true count at each time step, add noise distributed according to $\mathrm{Lap}(T/\varepsilon)$ to the count, and release this noisy count.

All of these options result in noise on the order of at least $\Omega(\sqrt{T}/\varepsilon)$. The hope is to do much better by exploiting structure of the query set.

Let $\mathcal{X}$ be the universe of possible input symbols. Let S and S' be stream prefixes (i.e., finite streams) of symbols drawn from $\mathcal{X}$. Then $\mathrm{Adj}(S, S')$ ("S is adjacent to S'") if and only if there exist $a, b \in \mathcal{X}$ so that if we change some of the instances of a in S to instances of b, then we get S'. More formally, $\mathrm{Adj}(S, S')$ iff $\exists a, b \in \mathcal{X}$ and $\exists R \subseteq [|S|]$, such that $S|_{R:a\to b} = S'$. Here, R is a set of indices in the stream prefix S, and $S|_{R:a\to b}$ is the result of replacing all the occurrences of a at these indices with b. Note that adjacent prefixes are always of the same length.

To capture event-level privacy, we restrict the definition of adjacency to the case $|R| \leq 1$. To capture user-level privacy we do not constrain the size of R in the definition of adjacency.

As noted above, one option is to publish a noisy count at each time step; the count published at time t reflects the approximate number of 1s in the length t prefix of the stream. The privacy challenge is that early items in the stream are subject to nearly T statistics, so for $(\varepsilon, 0)$-differential privacy we would be adding noise scaled to T/ε, which is unacceptable. In addition, since the 1s are the "interesting" elements of the stream, we would like that the distortion be scaled to the number of 1s seen in the stream, rather than to the length of the stream. This rules out applying randomized response to each item in the stream independently.

The algorithm below follows a classical approach for converting static algorithms to dynamic algorithms.

Assume T is a power of 2. The intervals are the natural ones corresponding to the labels on a complete binary tree with T leaves, where the leaves are labeled, from left to right, with the intervals $[0, 0], [1, 1], \ldots, [T-1, T-1]$ and each parent is labeled with the interval that is the union of the intervals labeling its children. The idea is to compute and release a noisy count for each label $[s, t]$; that is, the released value corresponding to the label $[s, t]$ is a noisy count of the number of 1s in positions $s, s+1, \ldots, t$ of the input stream. To learn the approximate cumulative count at time $t \in [0, T-1]$ the analyst uses the binary representation of t to determine a set of at most $\log_2 T$

Counter (T, ε)

Initialization. Initialize $\xi = \log_2 T/\varepsilon$, and sample Counter $\sim \text{Lap}(\xi)$.

Intervals. For $i \in \{1, \ldots, \log T\}$, associate with each string $s \in \{0,1\}^i$ the time interval S of $2^{\log T - i}$ time periods $\{s \circ 0^{\log T - i}, \ldots s \circ 1^{\log T - i}\}$. The interval *begins in time* $s \circ 0^{\log T - i}$ and *ends in time* $s \circ 1^{\log T - i}$.

Processing. In time period $t \in \{0, 1, \ldots, T-1\}$, let $x_t \in \{0,1\}$ be the t-th input bit:

1. For every interval I beginning at time t, initialize c_I to an independent random draw: $c_I \leftarrow \text{Lap}((\log_2 T)/\varepsilon)$;
2. For every interval I containing t, add x_t to c_I: $c_I \leftarrow c_I + x_t$;
3. For every interval I that ends in time t, output c_I.

Figure 12.1: Event-level private counter algorithm (not pan-private).

disjoint intervals whose union is $[0, t]$, and computes the sum of the corresponding released noisy counts.[2] See Figure 12.1.

Each stream position $t \in [0, T-1]$ appears in at most $1 + \log_2 T$ intervals (because the height of the tree is $\log_2 T$), and so each element in the stream affects at most $1 + \log_2 T$ released noisy counts. Thus, adding noise to each interval count distributed according to $\text{Lap}((1 + \log_2 T)/\varepsilon)$ ensures $(\varepsilon, 0)$-differential privacy. As for accuracy, since the binary representation of any index $t \in [0, T-1]$ yields a disjoint set of at most $\log_2 T$ intervals whose union is $[0, t]$ we can apply Lemma 12.2 below to conclude that the expected error is tightly concentrated around $(\log_2 T)^{3/2}$. The maximum expected error, over all times t, is on the order of $(\log_2 T)^{5/3}$.

Lemma 12.2. Let Let $Y_1, \ldots, Y_k$ be independent variables with distribution $\text{Lap}(b_i)$. Let $Y = \sum_i Y_i$ and $b_{\max} = \max_i b_i$. Let $\nu \geq \sqrt{\sum_i (b_i)^2}$, and $0 < \lambda < \frac{2\sqrt{2}\nu^2}{b_{\max}}$. Then

$$\Pr[Y > \lambda] \leq \exp\left(-\frac{\lambda^2}{8\nu^2}\right).$$

[2]This algorithm can be optimized slightly (for example, we never use the count corresponding to the root, eliminating one level from the tree), and it can be modified to handle the case in which T is not a power of 2 and, more interestingly, when T is not known *a priori*.

Proof. The moment generating function of Y_i is $\mathbb{E}[\exp(hY_i)] = 1/(1 - h^2b_i^2)$, where $|h| < 1/b_i$. Using the inequality $(1-x)^{-1} \leq 1 + 2x \leq \exp(2x)$ for $0 \leq x < 1/2$, we have $\mathbb{E}[\exp(hY_i)] \leq \exp(2h^2b_i^2)$, if $|h| < 1/2b_i$. We now calculate, for $0 < h < 1/\sqrt{2}b_{\max}$:

$$\begin{aligned}
\Pr[Y > \lambda] &= \Pr[\exp(hY) > \exp(h\lambda)] \\
&\leq \exp(-h\lambda)\mathbb{E}[\exp(hY)] \\
&= \exp(-h\lambda)\prod_i \mathbb{E}[\exp(hY_i)] \\
&\leq \exp(-h\lambda + 2h^2\nu^2).
\end{aligned}$$

By assumption, $0 < \lambda < \frac{2\sqrt{2}\nu^2}{b_{\max}}$. We complete the proof by setting $h = \lambda/4\nu^2 < 1/\sqrt{2}b_{\max}$. □

Corollary 12.3. Let $Y, \nu, \{b_i\}_i, b_{\max}$ be as in Lemma 12.2. For $\delta \in (0,1)$ and $\nu > \max\{\sqrt{\sum_i b_i^2}, b_{\max}\sqrt{\ln(2/\delta)}\}$, we have that $\Pr[|Y| > \nu\sqrt{8\ln(2/\delta)}] \leq \delta$.

In our case, all the b_i's are the same (e.g., $b = (\log_2 T)/\varepsilon$). Taking $\nu = \sqrt{k}b$ we have the following corollary:

Corollary 12.4. For all $\lambda < \alpha(\sqrt{k}b) < 2\sqrt{2}kb = 2\sqrt{2k}\nu$,

$$\Pr[Y > \lambda] \leq e^{-\alpha^2/8}.$$

Note that we have taken the unusual step of adding noise to the count *before* counting, rather than after. In terms of the outputs it makes no difference (addition is commutative). However, it has an interesting effect on the algorithm's internal states: they are differentially private! That is, suppose the intrusion occurs at time t, and consider any $i \in [0,t]$. Since there are at most $\log_2 T$ intervals containing step i (in the algorithm we abolished the interval corresponding to the root), x_i affects at most $\log_2 T$ of the noisy counts, and so x_i is protected against the intrusion for exactly the same reason that it is protected in the algorithm's outputs. Nevertheless, the algorithm in Figure 12.1 is *not* pan-private even against a single intrusion. This is because, while its internal state and its outputs are each independently differentially private, the joint distribution does not ensure ε-differential privacy. To see why this is so, consider an intruder that sees the internal state at time t and knows the entire data stream except x_{t+1}, and let $I = [a, b]$ be an interval containing both t and $t + 1$. Since the adversary knows $x_{[0,t]}$, it can subtract from c_I the contribution from the stream occurring up through time t (that is, it subtracts off from the observed c_I at time t the values $x_a, x_{a+1}, \ldots, x_t$, all of which it knows). From this the intruder learns the value of the Laplace draw to which c_I was initialized. When c_I is published at the end of step b, the adversary subtracts from the published value this initial draw, together with the contributions of all elements in $x_{[a,b]}$ except x_{t+1}, which it does not know. What remains is the unknown x_{t+1}.

12.3.1 Pan-private counting

Although the algorithm in Figure 12.1 is easily modified to ensure *event-level pan-privacy against a single intrusion*, we give a different algorithm here in order to introduce a powerful *bijection* technique which has proved useful in other applications. This algorithm maintains in its internal state a single noisy counter, or accumulator, as well as noise values for each interval. The output at any given time period t is the sum of the accumulator and the noise values for the intervals containing t. When an interval I ends, its associated noise value, η_I, is erased from memory.

Theorem 12.5. The counter algorithm of Figure 12.2, when run with parameters T, ε, and suffering at most one intrusion, yields an $(\varepsilon, 0)$-pan-private counter that, with probability at least $1 - \beta$ has maximum error, over its T outputs, of $O(\log(1/\beta) \cdot \log^{2.5} T/\varepsilon)$. We note also that in every round *individually* (rather than in all rounds simultaneously), with all but β probability, the error has magnitude at most $O(\log(1/\beta) \cdot \log^{1.5} T/\varepsilon)$.

Proof. The proof of accuracy is the same as that for the algorithm in Figure 12.1, relying on Corollary 12.4. We focus here on the proof of pan-privacy.

During an intrusion between atomic steps t^* and $t^* + 1$, that is, immediately following the processing of element t^* in the input stream

```
Pan-Private Counter $(T, \varepsilon)$
Initialization. Initialize $\xi = (1 + \log T)/\varepsilon$, and sample Counter $\sim \mathrm{Lap}(\xi)$.
Intervals. For $i \in \{1, \ldots, \log T\}$, associate with each string $s \in \{0,1\}^i$ the time interval $S$ of $2^{\log T - i}$ time periods $\{s \circ 0^{\log T - i}, \ldots s \circ 1^{\log T - i}\}$. The interval begins in time $s \circ 0^{\log T - i}$ and ends in time $s \circ 1^{\log T - i}$.
Processing. In time period $t \in \{0, 1, \ldots, T-1\}$, let $x_t \in \{0,1\}$ be the $t$-th input bit:
    1. Counter $\leftarrow$ Counter $+ x_t$;
    2. For every interval $I$ which begins in time $t$, sample noise $\eta_I \sim \mathrm{Lap}(\xi)$;
    3. Let $I_1, \ldots, I_{\log T}$ be the $\log T$ intervals that contain $t$. Output Counter $+ \sum_{i=1}^{\log T} \eta_{I_i}$.
    4. For every interval $I$ that ends in time $t$, erase $\eta_I$.
```

Figure 12.2: Event-level pan-private counter algorithm.

(recall that we begin numbering the elements with 0), the view of the adversary consists of (1) the noisy cumulative count (in the variable "count"), (2) the interval noise values η_S in memory when the intrusion occurs, and (3) the complete sequence of all of the algorithm's outputs in rounds $0, 1, \ldots, t$. Consider adjacent databases x and x', which differ in time t, say, without loss of generality, $x_t = 1$ and $x'_t = 0$, and an intrusion immediately following time period $t^* \geq t$ (we will discuss the case $t^* < t$ below). We will describe a bijection between the vector of noise values used in executions on x and executions on x', such that corresponding noise values induce identical adversary views on x and x', and the probabilities of adjacent noise values differ only by an e^{ε} multiplicative factor. This implies ε-differential pan-privacy.

By assumption, the true count just after the time period $t^* \geq t$ is larger when the input is x than it is when the input is x'. Fix an arbitrary execution E_x when the input stream is x. This amounts to fixing the randomness of the algorithm, which in turn fixes the noise values generated. We will describe the corresponding execution $E_{x'}$ by describing how its noise values differ from those in E_x.

The program variable Counter was initialized with Laplace noise. By increasing this noise by 1 in $E_{x'}$ the value of Counter just after step t^* is identical in $E_{x'}$ and E_x. The noise variables in memory immediately following period t^* are independent of the input; these will be unchanged in $E_{x'}$. We will make the sequence of outputs in $E_{x'}$ *identical* to those in E_x by changing a collection of $\log T$ interval noise values η_S that are *not* in memory when the adversary intrudes, so that the sum of *all* noise values in all rounds up through $t-1$ is unchanged, but the sum from round t on is larger by 1 for database x' than for x. Since we *increased* the initialization noise for Counter, we now need to *decrease* the sum of interval noise values for periods $0, \ldots, t-1$ by 1, and leave unchanged the sum of interval noise values from period t.

To do this, we find a collection of disjoint intervals whose union is $\{0, \ldots, t-1\}$. There is always such a collection, and it is always of size at most $\log T$. We can construct it iteratively by, for i decreasing from $\lfloor \log(t-1) \rfloor$ to 0, choosing the interval of size 2^i that is contained in $\{0, \ldots, t-1\}$ and is not contained in a previously chosen interval (if such an interval exists). Given this set of disjoint intervals, we notice also that they all end by time $t-1 < t \le t^*$, and so their noises are not in memory when the adversary intrudes (just following period t^*). In total (taking into account also changing the initial noise value for Counter), the complete view seen by the adversary is identical and the probabilities of the (collection of) noise values used for x and x' differ by at most an e^{ε} multiplicative factor.

Note that we assumed $t^* \ge t$. If $t^* < t$ then the initial noise added to Counter in $E_{x'}$ will be the same as in E_x, and we need to add 1 to the sum of interval noises in every time period from t through T (the sum of interval noises before time t remains unchanged). This is done as above, by finding a disjoint collection of at most $\log T$ intervals that exactly covers $\{t, \ldots, T-1\}$. The noise values for these intervals are not yet in memory when the intrusion occurs in time $t^* < t$, and the proof follows similarly. □

12.3.2 A logarithmic (in T) lower bound

Given the upper bound of Theorem 12.5, where the error depends only poly-logarithmically on T, it is natural to ask whether *any* dependence is inherent. In this section we show that a logarithmic dependence on T is indeed inherent.

Theorem 12.6. Any differentially private event-level algorithm for counting over T rounds must have error $\Omega(\log T)$ (even with $\varepsilon = 1$).

Proof. Let $\varepsilon = 1$. Suppose for the sake of contradiction that there exists a differentially private event-level counter for streams of length T that guarantees that with probability at least $2/3$, its count at all time periods is accurate up to a maximum error of $(\log_2 T)/4$. Let $k = (\log_2 T)/4$. We construct a set S of T/k inputs as follows. Divide the T time periods into T/k consecutive phases, each of length k (except, possibly, the last one). For $i = 1, \ldots, T/k$, the i-th input $x^i \in S$ has 0 input bits everywhere except during the ith phase. That is, $x^i = 0^{k \cdot i} \circ 1^k \circ 0^{k \cdot ((T/k)-(i+1))}$.

Fo $1 \leq i \leq T/k$, we say an output *matches* i if just before the ith phase the output is less than $k/2$ and at the end of the ith phase the output is at least $k/2$. By accuracy, on input x^i the output should match i with probability at least $2/3$. By ε differential privacy, this means that for every $i, j \in [T/k]$ such that $i \neq j$, the output on input x^i should match j with probability at least

$$\begin{aligned} e^{-2\varepsilon \cdot k} &= e^{-\varepsilon \log(T^{1/2})} \\ &= e^{-\log(T^{1/2})} = 1/\sqrt{T}. \end{aligned}$$

This is a contradiction, because the events that the output matches j are disjoint for different j, and yet the sum of their probabilities on input x^i exceeds 1. □

12.4 Average case error for query release

In Sections 4 and 5, we considered various mechanisms for solving the private query release problem, where we were interested in *worst case error*. That is, given a class of queries $\mathcal{Q}$, of size $|\mathcal{Q}| = k$, we wished to recover a vector of answers $\hat{a} \in \mathbb{R}^k$ such that for *each* query $f_i \in \mathcal{Q}$, $|f_i(x) - \hat{a}_i| \leq \alpha$ for some worst-case error rate α. In other words, if we let $a \in \mathbb{R}^k$ denote the vector of *true* answers, with $a_i \equiv f_i(x)$, then we require a bound of the form: $\|a - \hat{a}\|_\infty \leq \alpha$. In this section, we consider a weakened utility guarantee, on the ℓ_2 (rather than ℓ_∞) error: a bound of the form $\|a - \hat{a}\|_2 \leq \alpha$. A bound of this form does not guarantee that we have low error for *every* query, but it does guarantee that on average, we have small error.

Although this sort of bound is weaker than worst-case error, the mechanism is particularly simple, and it makes use of an elegant geometric view of the query release problem that we have not seen until now.

Recall that we can view the database x as a vector $x \in \mathbb{N}^{|\mathcal{X}|}$ with $\|x\|_1 = n$. We can similarly also view the queries $f_i \in \mathcal{Q}$ as vectors $f_i \in \mathbb{N}^{|\mathcal{X}|}$, such that $f_i(x) = \langle f_i, x\rangle$. It will therefore be helpful to view our class of queries $\mathcal{Q}$ as a matrix $A \in \mathbb{R}^{k\times|\mathcal{X}|}$, with the ith row of A being the vector f_i. We can then see that our answer vector $a \in \mathbb{R}^k$ is, in matrix notation:

$$A \cdot x = a.$$

Let's consider the domain and range of A when viewed as a linear map. Write $B_1 = \{x \in \mathbb{R}^{|\mathcal{X}|} : \|x\|_1 = 1\}$ denote the unit ℓ_1 ball in $|\mathcal{X}|$ dimensional space. Observe that $x \in nB_1$, since $\|x\|_1 = n$. We will refer to nB_1 as "Database Space." Write $K = AB_1$. Note similarly that for all $x \in nB_1$, $a = A \cdot x \in nK$. We will refer to nK as "answer space." We make a couple of observations about K: Note that because B_1 is centrally symmetric, so is K — that is, $K = -K$. Note also that $K \subset \mathbb{R}^k$ is a convex polytope with vertices $\pm A^1, \ldots, \pm A^{|\mathcal{X}|}$ equal to the columns of A, together with their negations.

The following algorithm is extremely simple: it simply answers every query independently with the Laplace mechanism, and then *projects back into answer space.* In other words, it adds independent Laplace noise to every query, which as we have seen, by itself leads to distortion that is linear in k (or at least $\sqrt{k}$, if we relax to (ε, δ)-differential privacy). However, the resulting vector $\tilde{a}$ of answers is likely not consistent with *any* database $y \in nB_1$ in database space. Therefore, rather than returning $\tilde{a}$, it instead returns some consistent answer vector $\hat{a} \in nK$ that is as close to $\tilde{a}$ as possible. As we will see, this projection step improves the accuracy of the mechanism, while having no effect on privacy (since it is just post-processing!)

We first observe that Project is differentially private.

Theorem 12.7. For any $A \in [0,1]^{k\times|\mathcal{X}|}$, $\textbf{Project}(x, A, \varepsilon)$ preserves (ε, δ)-differential privacy.

Algorithm 18 The K-Projected Laplace Mechanism. It takes as input a matrix $A \in [0,1]^{k\times|\mathcal{X}|}$, a database $x \in nB_1$, and a privacy parameters ε and δ.

```
Project(x, A, ε, δ):
  Let $a = A \cdot x$
  For each $i \in [k]$, sample $\nu_i \sim \mathrm{Lap}(\sqrt{8k\ln(1/\delta)}/\varepsilon)$, and let $\tilde{a} = a + \nu$.

  Output $\hat{a} = \arg\min_{\hat{a}\in nK} \|\hat{a} - \tilde{a}\|_2^2$.
```

Proof. We simply note that $\tilde{a}$ is the output of the Laplace mechanism on k sensitivity 1 queries, which is (ε, δ)-differentially private by Theorems 3.6 and 3.20 . Finally, since $\hat{a}$ is derived from $\tilde{a}$ without any further access to the private data, the release of $\hat{a}$ is differentially private by the post-processing guarantee of differential privacy, Proposition 2.1. □

Theorem 12.8. For any class of linear queries A and database x, let $a = A \cdot x$ denote the true answer vector. Let $\hat{a}$ denote the output of the mechanism **Project**: $\hat{a} =$ **Project**(x, A, ε). With probability at least $1 - \beta$:

$$\|a - \hat{a}\|_2^2 \leq \frac{kn\sqrt{192\ln(1/\delta)\ln(2|\mathcal{X}|/\beta)}}{\varepsilon}.$$

To prove this theorem, we will introduce a couple of simple concepts from convex geometry. For a convex body $K \subset \mathbb{R}^k$, its *polar body* is K° defined to be $K^\circ = \{y \in \mathbb{R}^k : \langle y, x\rangle \leq 1 \text{ for all } x \in K\}$. The *Minkowski Norm* defined by a convex body K is

$$\|x\|_K \equiv \min\{r \in \mathbb{R} \text{ such that } x \in rK\}.$$

The *dual norm* of $\|x\|_K$ is the Minkowski norm induced by the polar body of K, i.e., $\|x\|_{K^\circ}$. This norm also has the following form:

$$\|x\|_{K^\circ} = \max_{y\in K}\langle x, y\rangle.$$

The key fact we will use is *Holder's Inequality*, which is satisfied by all centrally symmetric convex bodies K:

$$|\langle x, y\rangle| \leq \|x\|_K \|y\|_{K^\circ}.$$

Proof of Theorem 12.8. The proof will proceed in two steps. First we will show that: $\|a-\hat{a}\|_2^2 \leq 2\langle \hat{a}-a, \tilde{a}-a\rangle$, and then we will use Holder's inequality to bound this second quantity.

Lemma 12.9.

$$\|a - \hat{a}\|_2^2 \leq 2\langle \hat{a} - a, \tilde{a} - a\rangle$$

Proof. We calculate:

$$\begin{aligned}
\|\hat{a} - a\|_2^2 &= \langle \hat{a} - a, \hat{a} - a\rangle \\
&= \langle \hat{a} - a, \tilde{a} - a\rangle + \langle \hat{a} - a, \hat{a} - \tilde{a}\rangle \\
&\leq 2\langle \hat{a} - a, \tilde{a} - a\rangle.
\end{aligned}$$

The inequality follows from calculating:

$$\begin{aligned}
\langle \hat{a} - a, \tilde{a} - a\rangle &= \|\tilde{a} - a\|_2^2 + \langle \hat{a} - \tilde{a}, \tilde{a} - a\rangle \\
&\geq \|\hat{a} - \tilde{a}\|_2^2 + \langle \hat{a} - \tilde{a}, \tilde{a} - a\rangle \\
&= \langle \hat{a} - \tilde{a}, \hat{a} - a\rangle,
\end{aligned}$$

Where the final inequality follows because by choice of $\hat{a}$, for all $a' \in nK$: $\|\tilde{a} - \hat{a}\|_2^2 \leq \|\tilde{a} - a'\|_2^2$. □

We can now complete the proof. Recall that by definition, $\tilde{a}-a = \nu$, the vector of i.i.d. Laplace noise added by the Laplace mechanism. By Lemma 12.9 and Holder's inequality, we have:

$$\begin{aligned}
\|a - \hat{a}\|_2^2 &\leq 2\langle \hat{a} - a, \nu\rangle \\
&\leq 2\|\hat{a} - a\|_K \|\nu\|_{K^\circ}.
\end{aligned}$$

We bound these two terms separately. Since by definition $\hat{a}, a \in nK$, we have $\max(\|\hat{a}\|_K, \|a\|_K) \leq n$, and so by the triangle inequality, $\|\hat{a} - a\|_K \leq 2n$.

Next, observe that since $\|\nu\|_{K^\circ} = \max_{y\in K}\langle y, \nu\rangle$, and since the maximum of a linear function taken over a polytope is attained at a vertex, we have: $\|\nu\|_{K^\circ} = \max_{i\in[|\mathcal{X}|]} |\langle A^i, \nu\rangle|$.

Because each $A^i \in \mathbb{R}^k$ is such that $\|A^i\|_\infty \leq 1$, and recalling that for any scalar q, if $Z \sim \text{Lap}(b)$, then $qZ \sim \text{Lap}(qb)$, we can apply Lemma by

Lemma 12.2 to bound the weighted sums of Laplace random variables $\langle A^i, \nu \rangle$. Doing so, we have that with probability at least $1-\beta$:

$$\max_{i\in[|\mathcal{X}|]} |\langle A^i, \nu\rangle| \leq \frac{8k\sqrt{\ln(1/\delta)\ln(|\mathcal{X}|/\beta)}}{\epsilon}.$$

Combining all of the above bounds, we get that with probability $1-\beta$:

$$\|a-\hat{a}\|_2^2 \leq \frac{16nk\sqrt{\ln(1/\delta)\ln(|\mathcal{X}|/\beta)}}{\epsilon}. \qquad \square$$

Let's interpret this bound. Observe that $\|a-\hat{a}\|_2^2 = \sum_{i=1}^k (a_i - \hat{a}_i)^2$, and so this is a bound on the sum of squared errors over all queries. Hence, the *average* per-query squared error of this mechanism is only:

$$\frac{1}{k}\sum_{i=1}^{k}(a_i-\hat{a}_i)^2 \leq \frac{16n\sqrt{\ln(1/\delta)\ln(|\mathcal{X}|/\beta)}}{\epsilon}.$$

In contrast, the private multiplicative weights mechanism guarantees that $\max_{i\in[k]} |a_i - \hat{a}_i| \leq \tilde{O}(\sqrt{n}\log|\mathcal{X}|^{1/4}/\varepsilon^{1/2})$, and so matches the average squared error guarantee of the projected Laplace mechanism, with a bound of: $\tilde{O}(n\sqrt{\log|\mathcal{X}|}/\varepsilon)$. However, the multiplicative weights mechanism (and especially its privacy analysis) its much more complex than the Projected Laplace mechanism! In particular, the *private* part of the K-Projected Laplace mechanism is simply the Laplace mechanism itself, and requires no coordination between queries. Interestingly — and, it turns out, necessarily — coordination occurs in the projection phase. Since projection is in post-precessing, it incurs no further privacy loss; indeed, it can be carrie out (online, if necessary) by the data analyst himself.

12.5 Bibliographical notes

The local model of data privacy has its roots in randomized response, as first proposed by Warner in 1965 [84]. The local model was formalized by Kasiviswanathan et al. [52] in the context of learning, who proved that private learning in the local modal is equivalent to non-private learning in the statistical query (SQ) model. The set of queries which can be *released* in the local model was shown to be exactly equal to the set of queries that can be *agnostically learned* in the SQ model by Gupta et al. [38].

Pan-Privacy was introduced by Dwork et al. [27], and further explored by Mir et al. [62]. The pan-private density estimation, as well as a low-memory variant using hashing, appear in [27].

Privacy under continual observation was introduced by Dwork et al. [26]; our algorithm for counting under continual observation is from that paper, as is the lower bound on error. Similar algorithms were given by Chan et al. [11]. The proof of concentration of measure inequality for the sums of Laplace random variables given in Lemma 12.2 is from [11].

The Projected Laplace mechanism for achieving low average error was given by Nikolov et al.[66], who also give *instance optimal* algorithms for the (average error) query release problem for any class of queries. This work extends a line of work on the connections between differential privacy and geometry started by Hardt and Talwar [45], and extended by Bhaskara et al. [5] and Dwork et al. [30].

Dwork, Naor, and Vadhan proved an exponential gap between the number of queries that can be answered (with non-trivial error) by stateless and stateful differentially private mechanisms [29]. The lesson learned — that coordination is essential for accurately and privately answering very large numbers of queries — seems to rule out the independent noise addition in the Projected Laplace mechanism. The statefulness of that algorithm appears in the projection step, resolving the paradox.