Differential Privacy in New Settings

Cynthia Dwork*

Abstract

Differential privacy is a recent notion of privacy tailored to the problem of statistical disclosure control: how to release statistical information about a set of people without compromising the the privacy of any individual [7].

We describe new work [10, 9] that extends differentially private data analysis beyond the traditional setting of a trusted curator operating, in perfect isolation, on a static dataset. We ask

- How can we guarantee differential privacy, even against an adversary that has access to the algorithm's internal state, *eg,* by subpoena? An algorithm that achives this is said to be *pan-private.*
- How can we guarantee differential privacy when the algorithm must continually produce outputs? We call this *differential privacy under continual observation.*

We also consider these requirements in conjunction.

1 Introduction

Differential privacy is a recent privacy guarantee tailored to the problem of statistical disclosure control: how to publicly release statistical information about a set of people without compromising the privacy of any individual [7] (see [4, 5, 3] for motivation, history, basic techniques, and pointers to recent results).

In a nutshell, differential privacy requires that the probability distribution on the published results of an analysis is "essentially the same," independent of whether any individual opts in to, or opts out of, the data set. (The probabilities are over the coin flips of the privacy mechanism.) Statistical databases are frequently created to achieve a social goal, and increased participation in the databases permits more accurate analyses. The differential privacy guarantee supports the social goal by assuring each individual that she incurs little risk by joining the database: anything that can happen is essentially equally likely to do so whether she joins or abstains.

The strengths of differential privacy are

1. It is independent of any additional information, including other databases, available to the adversary;
2. It is achievable using fairly simple and general mechanisms; and
3. It frequently allows very accurate analyses.

*Microsoft Research. E-mail: dwork@microsoft.com.

Up to this point, research on differentially private data analysis has focussed on the setting of a trusted curator holding a large, static, data set, held in a permanently infrangible storage system. The curator either responds to queries (the *interactive* case) or prepares some sort of summary or synthetic database (the *non-interactive* case), intended to answer all queries of a particular type[1]. A line of work investigates the way in which the accuracy of of the responses to the queries may need to deteriorate with the number, sensitivity (a metric describing how much the addition or deletion of a member of the database can change affect the outputs), and geometry of the query sequence [2, 8, 12, 13], and general techniques have been developed that in some cases match these bounds [11, 1, 7, 14, 13].

In this paper we describe investigations of differential privacy in two new realms. We describe these informally, working from a real-life example.

Continual Observation. Consider a website for H1N1 self-assessment[2]. 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 like to *continually* analyze aggregate information of consenting users in order to monitor regional health conditions, with the goal, for example, of organizing improved flu response. Can we do this in a differentially private fashion with reasonable accuracy (despite the fact that the system is continually producing outputs)?

We would expect a given individual to interact very few times with the H1N1 self-assessment site. For simplicity, let us say this is just once (the general case is an easy extension). 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). That is, the probability of any output sequence should

[1] The one exception considers a distributed setting, in which each data holder controls her own data and decides in which analyses to participate [6]. However, this is done by emulating the curator via secure multiparty computation.

[2] https://h1n1.cloudapp.net is such a website. user-supplied data are stored for analysis only if the user consents.

be essentially the same, independent of the presence or absence of any single interaction with the site.

Pan-Privacy The privacy statement at the H1N1 self-assessment site encourages users to allow their data to be shared:

> "This information can be very helpful in monitoring regional health conditions, plan flu response, and conduct health research. By allowing the responses to the survey questions to be used for public health, education and research purposes, you can help your community."

But the site also describes conditions under which the data may be disclosed

> "...if required to do so by law or in the good faith belief that such action is necessary to (a) conform to the edicts of the law or comply with legal process served on Microsoft or the Site; (b) protect and defend the rights or property of Microsoft and our family of Web sites; or (c) act in urgent circumstances to protect the personal safety of users of Microsoft products or members of the public."

This raises the following question, orthogonal to that of continual observation: Is it possible to maintain a differentially private *internal state*, so that a consenting user has privacy even against a subpoena or other intrusion into the local state? Algorithms with this property are called *pan-private*, as they are private inside (internal state) and out (output sequence). The goal of pan-privacy is to provide a mathematically rigorous way of (essentially) eliminating the risk – of anything – incurred by sharing one's information *even in the presence of an intrusion*, further encouraging participation and thereby increasing the social benefit of the site. Another motivation is the prevention of "mission creep" for data, protecting the data curator from the (very real!) pressure to allow data to be used for purposes other than that for which they were collected. This fits well with streaming algorithms with small state, which can at most store a few data items, and we do focus on the streaming model. However, nothing prevents a streaming algorithm from storing the data of a person of interest. Pan-privacy rules this out, since the internal state must have essentially the same distribution, independent of whether the person of interest is, or is not, in the data set.

We remark that, since an intrusion can occur at an unpredictable time, designing pan-private algorithms is interesting even when the system will generate only a single output.

User-Level (Pan-)Privacy. We have made the reasonable assumption that a single individual interacts with the H1N1 self-assessment site at most a small number of times (one). Such an assumption is not reasonable for other kinds of websites, such as a search engine. We ask: what kinds of statistics can we gather while preserving the privacy of an individual's entire history of accesses to the website? That is, if we require that, no matter how many times an individual accesses the website, and no matter how these accesses are interleaved with those of others, the distribution on outputs, or, in the case of *user-level pan-privacy*, the distribution on pairs (internal state, output sequence), should be essentially the same, independent of presence or absence of of any individual's data in the stream, what can we compute?

2 Differential Privacy Basics

In the literature, a differentially private mechanism operates on a *database*, or *data set*. This is a collection of *rows*, where the data of an individual are held in a single row. Differential privacy ensures that the ability of an adversary to inflict harm (or good, for that matter) – of any sort, to any set of people – is essentially the same, independent of whether any individual opts in to, or opts out of, the dataset. This is done indirectly, simultaneously addressing all possible forms of harm and good, by focusing on the probability of any given output of a privacy mechanism and how this probability can change with the addition or deletion of any row. We will concentrate on pairs of databases (D, D') differing only in one row, meaning one is a subset of the other and the larger database contains just one additional row.

DEFINITION 2.1. [7] *A randomized function $\mathcal{K}$ gives ε-*differential privacy *if for all data sets D and D' differing in at most one row, and all $S \subseteq Range(\mathcal{K})$,*

$$\Pr[\mathcal{K}(D) \in S] \leq \exp(\varepsilon) \times \Pr[\mathcal{K}(D') \in S], \tag{2.1}$$

where the probability space in each case is over the coin flips of $\mathcal{K}$.

The multiplicative nature of the guarantee implies that an output whose probability is zero on a given database must also have probability zero on any neighboring database, and hence, by repeated application of the definition, on any other database.

The parameter ε is public, and its selection is a social question. We tend to think of ε as, say, 0.01, 0.1, or in some cases, $\ln 2$ or $\ln 3$.

DEFINITION 2.2. [7] *For $f : \mathcal{D} \to \mathbf{R}^d$, the L_1* sensitivity *of f is*

$$\Delta f = \max_{D,D'} \|f(D) - f(D')\|_1 \tag{2.2}$$

for all D, D' differing in at most one row.

The Laplace distribution with parameter b, denoted Lap(b), has density function $P(z|b) = \frac{1}{2b}\exp(-|z|/b)$ and variance $2b^2$. Taking $b = 1/\varepsilon$ we have that the density at z is proportional to $e^{-\varepsilon|z|}$. This distribution has highest density at 0 (good for accuracy), and for any z, z' such that $|z - z'| \leq 1$ the density at z is at most e^{ε} times the density at z'. Finally, the distribution gets flatter as ε decreases: smaller ε means better privacy, so the noise density should be less "peaked" at 0 and change more gradually as the magnitude of the noise increases.

THEOREM 2.1. [7] *For $f : \mathcal{D} \to \mathbf{R}^d$, the mechanism $\mathcal{K}$ that adds independently generated noise with distribution Lap($\Delta f/\varepsilon$) to each of the d output terms enjoys ε-differential privacy.*

As an example, given a dataset of n rows, consider the query "How many rows satisfy property P?" The sensitivity of this query is 1. The theorem says that adding to the true answer noise distributed according to Lap($1/\varepsilon$) suffices to ensure differential privacy.

3 Summary of Results

in this section we briefly summarize the results in [10, 9], and highlight three of these for more detailed description later in the paper.

Pan-privacy is introduced in [10], where *user-level pan-private* streaming algorithms are obtained for a variety of counting tasks.

We assume a data stream of unbounded length composed of elements in a universe 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 X is the universe of potential IP addresses. Thus, intuitively, privacy protects the presence or absence of an IP address in the stream, indpendent of the number of times it arises.

X-Adjacent Data Streams. Data streams (or stream prefixes) S and S' are X-adjacent if they differ only in the presence or absence of *any number of* occurrences of a single element $x \in X$. That is, the stream (or stream prefixes) obtained by deleting all occurrences of x from S and S' are identical.

User-Level Pan-Privacy. Let **Alg** be an algorithm. Let I denote the set of internal states of the algorithm, and σ the set of possible output sequences. Then algorithm **Alg** mapping data stream prefixes to the range $\mathrm{I} \times \sigma$, is *pan-private (against a single intrusion)* if for all sets $\mathrm{I}' \subseteq \mathrm{I}$ and $\sigma' \subseteq \sigma$, and for all pairs of X-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**.

Note that popular techniques from the streaming literature, such as Count-Min Sketch and subsampling, cannot be pan-private.

The algorithms in [10] solve the following counting problems (in each case, the algorithm is given a data stream prefix drawn from a known universe X):

- Density estimation: estimate what fraction of the elements of X appear at least once in the stream;
- t-cropped mean: estimate $\sum_{x \in X} \max(n_x, t)$, where n_x is the number of times x appears in the stream;
- k-heavy hitters: estimate the fraction of elements in X that appear at least k times in the stream;
- incidence counts: estimate the fraction of elements in X that appear exactly k times in the stream;
- modular incidence counts: estimate, for each $i = 0, 1, \ldots, k-1$ the fraction of elements in X that appear in the stream i times modulo k.

The paper differentiates between *announced* and *unannounced* intrusions. The former are made with the knowledge of the algorithm; this is what happens in the case of a subpoena. The latter are made without the knowledge of the algorithm; this may happen via a security breach. In general, it is possible for a pan-private algorithm to tolerate multiple announced intrusions, possibly at a loss of accuracy, as the algorithm can refresh randomness that is used for hiding events. The algorithms above also tolerate a single *unannounced* intrusion. The paper contains strong negative results for even two unannounced intrusions (but only for algorithms whose state size does not grow with the length of the stream). We will describe the density estimator in Section 4.

Differential privacy under continual observation is introduced in [9], where several results are obtained based on a *counter* primitive. We think of a counter as operating in a sequence of time intervals. In each interval the algorithm reads a symbol from the stream, updates its state, and produces an output. At all times the output is an estimate of the total number of events "of interest" seen, so far, in the stream. Formally, this can be modeled by a streaming algorithm operating on a stream over $\{0, 1\}$, where 0 corresponds to no event

in this time period and 1 corresponds to an event of interest (see Section 5).

The counter ensures *event-level privacy under continual observation*. Moreover, a minor modification to the counter adds pan-privacy against a single unannounced intrusion. The counter is described in Section 6.

The counter can be used to strengthen pan-private estimators from [10], described above, which only permit a single output (or small number of outputs), to obtain estimators that enjoy *user-level pan-privacy under continual observation*. On the one hand, this seems unlikely: the counter only enjoys event-level privacy, and this is inherent; it counts events that are generated by individuals, and if events caused by a given individual appear many times in the data stream then the counter must show a significantly higher number than if no events caused by this individual appear. On the other hand, speaking intuitively, if we can ensure that there is only one *significant* event (or a small number of significant events) for each individual that need be recorded by the counter, then event-level privacy of the counter is sufficient to ensure user-level privacy. This intuition can be realized, and in Section 7 we describe a continual observation user-level pan-private estimator [9].

4 Density Estimation

In this section we describe a pan-private streaming algorithm for estimating the density of the input stream [10]. The algorithm uses a technique based on *randomized response* [15]. In this technique the data themselves are randomized, and statistics are computed from the noisy data, taking into account the distribution on the perturbation. The term "randomized response" comes from the practice of having survey respondents secretly flip a coin and, based on the outcome, answer an invasive yes/no question (heads) or answer a more emotionally neutral yes/no question (tails), without revealing to the curator which of the two questions was answered (the result of the coin flip). In the computer science literature, the choice governed by the coin flip is usually between honestly reporting one's value and responding randomly, typically by flipping a second coin and reporting the outcome.

We first describe an algorithm with *additive* error, and then discuss how to modify it to obtain multiplicative error.

We sample a random a set $M \subseteq X$ of m items, where m is large enough so that, with high probability, the density of M's items in the input stream approximates the density of X's items. The algorithm maintains a table, with one entry per item in M. The table satisfies the following invariant: For items in M that have not yet appeared in the input stream, the entry is a draw from a distribution $\mathcal{D}_0$ (over $\{0,1\}$). For items that have appeared at least once, the entry is a draw from $\mathcal{D}_1$ (no matter how many times they appear). These two distributions should be "close enough" to guarantee pan-privacy for individual users, but "far enough" to allow collection of aggregate statistics about the fraction of users that appear at least once. Specifically, the distribution $\mathcal{D}_0$ will give the outputs 0 and 1 both with probability 1/2. The distribution $\mathcal{D}_1 = \mathcal{D}_1(\varepsilon)$ will give 1 with probability $1/2 + \varepsilon/4$ and 0 with probability $1/2 - \varepsilon/4$.

CLAIM 4.1. *For $\varepsilon \leq 1/2$, the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are "ε-differentially private". For both values $b \in \{0,1\}$, it holds that: $e^{-\varepsilon} \leq \Pr_{\mathcal{D}_1}[b]/\Pr_{\mathcal{D}_0}[b] \leq e^{\varepsilon}$.*

Density Estimator $(\varepsilon, \alpha, \beta)$

Init. Sample at random a set M of $m = \text{poly}(1/\varepsilon, 1/\alpha, \log(1/\beta))$ elements (representatives) in X. Create a table of size m with a single one-bit entry for each item in M. For every entry $x \in M$, generate a random initial value $b_x \sim \mathcal{D}_0$.

Processing. When a value $x \in M$ appears in the data stream, update x's entry in the table by drawing it from $\mathcal{D}_1$: $b_x \sim \mathcal{D}_1(\varepsilon)$.

Output. Compute θ, the fraction of entries in the table with value 1. Output the density value $f' = 4(\theta - 1/2)/\varepsilon + \text{Lap}(1/(\varepsilon \cdot m))$.

Figure 1: Density Estimator

THEOREM 4.1. [10] *Assume $\varepsilon \leq 1/2$. The density estimator of Figure 1 guarantees 2ε-differential pan-privacy. For a fixed input, with probability $1-\beta$ over the estimator's coins, the output is α-close to the fraction of items in X that appear in the input stream. The space used is* $\text{poly}(1/\varepsilon, 1/\alpha, \log(1/\beta))$.

Proof. We argue utility and privacy separately.

Privacy. For items not in M no information is stored by the algorithm, so privacy is perfect. For an item in the set M, if the item never appears in the input stream, then its entry is drawn from $\mathcal{D}_0$. If the item appears once or more in the input stream, then its entry is drawn from $\mathcal{D}_1$ (multiple appearances result in multiple samples being taken, but the item's entry will still be a sample from $\mathcal{D}_1$). Thus when an intrusion occurs, by Claim 4.1, the users in M are guaranteed ε-differential privacy against the intrusion.

Later, when the algorithm generates its output, the sensitivity of this output to the presence or absence of any one user is at most $1/m$ (0 if the user is not in M). By adding noise sampled from $\mathrm{Lap}(1/(\varepsilon \cdot m))$ we guarantee that users are guaranteed 2ε privacy against the combined information an adversary can gain from its intrusion and viewing the algorithm's output.

Utility. First, we claim that the fraction of items in M that appear at least once is, with probability at least $1 - \beta/2$, an $\alpha/2$ approximation to the fraction of items in X that appear at least once in the input stream. This follows by a Chernoff bound, because M is a set of $m \geq \mathrm{poly}(1/\alpha, \log(1/\beta))$ items chosen uniformly and at random from the set X.

We now complete the proof by showing that, with probability at least $1-\beta/2$, the algorithm's output is an $\alpha/2$-approximation to the fraction of items *in* M that appear at least once. Let f be the *true* fraction of items in M that appear at least once. The *expected* fraction of 1-entries in the table (over the randomness of drawing from $\mathcal{D}_0$ and $\mathcal{D}_1$ is then:

$$E[\theta] = f \cdot (1/2 + \varepsilon/4) + (1 - f) \cdot 1/2) = 1/2 + f \cdot \varepsilon/4$$

We can separate this fraction into the contribution of 1's from items that appeared at least once (samples from $\mathcal{D}_1$) and those that did not (samples from $\mathcal{D}_0$). Taking a Chernoff bound we get that, since $m = \mathrm{poly}(1/\varepsilon, 1/\alpha, \log(1/\beta))$, with probability $1 - \beta/2$ the observed fraction of 1's θ satisfies:

$$|\theta - E[\theta]| \leq \alpha \cdot \varepsilon/8$$

Now the algorithm outputs $f' = 4(\theta - 1/2)/\varepsilon + \mathrm{Lap}(1/(\varepsilon \cdot m))$. We conclude that with probability $1 - \beta$:

$$|f - f'| \leq \alpha$$

Obtaining a multiplicative error. If the fraction of elements of X that appear in the stream is very small, an additive accuracy guarantee might not provide a very meaningful result. In such cases it is better to give a multiplicative guarantee. This can be achieved as follows.

The universe X is hashed into $\ell = \log |X|$ smaller sets $X_0, X_1, \ldots, X_\ell = X$, $|X_i| = 2^i$. The density estimation algorithm is run independently for all ℓ choices of m, using noise $\mathrm{Lap}((\ell+1)/\varepsilon \cdot m)$ in the Output step. Finally, the "right" output must be chosen from among these ℓ outputs, as we now explain.

For $i = 0, \ldots, \ell$, the ith density estimator yields an additively accurate estimation γ'_i of the fraction of hash values of the input items in the hash set X_i. Were it not for collisions, the density of the input stream in a set X_i could give a good idea of the ratio between the number of distinct items in the input stream and $|X_i|$, but the collisions lead to under-counting. Of course, the problem gets worse as the set size shrinks. We thus want to use the density of the "right" X_i, one that is neither too large (leading to the additive error being very large in multiplicative terms) nor too small (leading to under-counting).

Hashing is done using pairwise independent hash functions. In this case, we know that, if the density of the output in X_k is γ_k, then with high probability the number of collisions is at most $\gamma_k^2 \cdot |X_k|$. If we use the first (smallest k) hash function for which $\gamma'_k \approx \alpha$, them the number of collisions is, with high probability, bounded by $\alpha^2 \cdot |X_k|$.

On the other hand, we know that the number of distinct elements in the input is at least about $\alpha \cdot |X_k|$. If we obtain an α^2-additive approximation to the density of the input's hash in X_k we can multiply it by $|X_k|$ and we actually have an $O((\alpha^2) \cdot |X_k|)$-additively accurate approximation on the *number* of distinct items in the input, which is also an $O(\alpha)$-multiplicative accurate estimation on the number of distinct elements (since the number of distinct elements was at least $\alpha \cdot |X_k|$).

Handling Multiple Announced Intrusions. It is possible to handle multiple *announced* intrusions by re-randomizing the randomized-response bits after each intrusion. For example, after the first announced intrusions, for every $x \in M$ in the table, we re-draw the bit b_x according to its current value: if previously it was 1, we re-draw from $\mathcal{D}_1$, and if it was 0 we re-draw from $\mathcal{D}_0$. Thus entries for items that appeared in the input stream are now distributed according to $\mathcal{D}'_1$, which is 1 w.p. $(1/2 + \varepsilon/8 + \varepsilon^2/16)$ and 0 w.p. $(1/2 - \varepsilon/8 - \varepsilon^2/16)$. Entries for items that did not appear in the stream are distributed according to $\mathcal{D}'_0$, which is 1 w.p. $(1/2 + \varepsilon/8)$ and 0 w.p. $(1/2 - \varepsilon/8)$. The algorithm then changes the randomized response distributions to be $\mathcal{D}'_1$ (for items that appear) and $\mathcal{D}'_0$ (for items that don't appear). Note that the algorithm keeps track of how many intrusions happened (there is no need to protect the intruder's privacy).

5 Definitions for Continual Observation Algorithms

Speaking intuitively, 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 this formally with a streaming algorithm, just as in earlier sections of this paper. Thus, as before, computation proceeds in a sequence of atomic steps. At each step the algorithm receives an input from the stream, computes (changes state), and produces outputs. Thus the intuitive notion

of "t time periods" corresponds to processing a sequence of t elements in the stream.

Because we are modeling real systems, where time is a factor (computers have clocks, as do the adversaries), we will occasionally need to model the fact that "nothing has happened" in a given time unit. We may do this by means of a "nothing happened" element. 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" menas "nothing happened," "1" means the event of interest occurred, and for $t = 1, 2, \ldots$ the algorithm outputs an approximation to the number of 1's seen in the length t prefix of the stream.

To define privacy, we need a notion of adjacent stream prefixes. The definition of adjacency in Section 2 permited strings of radically different lengths to be adjacent. To be able to capture time, we modify the definition as follows. Let X be the universe of possible input symbols. Let S and S' be stream prefixes (ie, finite streams) of symbols drawn from X. Then $\mathrm{Adj}(S, S')$ ("S is adjacent to S'") if and only if there exist $x, x' \in X$ so that if we change some of the instances of x in S to instances of x', then we get S'. More formally, $\mathrm{Adj}(S, S')$ iff $\exists x, x' \in X$ and $\exists T \subseteq [|S|]$, such that $S|_{T:x\to x'} = S'$. Here, T is a set of indices in the stream prefix S, and $S|_{T:x\to x'}$ is the result of replacing all the occurrences of x at these indices with x' (note that without loss of generality we can assume $S|_T$ contains only occurrences of x).

With this definition, X-adjacent prefixes are always of the same length.

6 The Counter Primitive

A Simple Solution. A natural approach to creating a counter is to noisily report each value (in $\{0,1\}$) observed at each step. This could be done using randomized response, drawing from distribution $\mathcal{D}_0$, respectively $\mathcal{D}_1$, as appropriate, and outputting the cumulative sum at each step. Alternatively, at step t, on input $x_t \in \{0,1\}$, the algorithm can add $x_t + \mathrm{Lap}(1/\varepsilon)$ to the output. We call this the *simple solution.* Event-level pan-privacy is immediate for both these algorithms, which keep no state. Accuracy, however, is more problematic. Over t steps of the simple solution, for large enough t, we expect the total amount of noise added to be $O(\sqrt{t}/\varepsilon)$. For "dense" data streams, where many of the x_t's are 1, this algorithm performs well. The problem is that if the number of 1's is smaller than $\sqrt{t}$, a reasonable scenario (especially if the granularity of time periods is small), the output count will be overwhelmed by the noise. We seek an algorithm that will produce accurate answers even for streams that are "sparse", i.e. with few 1's.

Cascading Buffers Counter [9]. We now describe a counter in which the error depends only polylogarithmically on the number of length of the stream prefix processed, addressing the deficiency, in the case of sparse data streams, of the simple approach. The basic strategy is to reduce the frequency of updates as the stream density decreases, and increase it when the density increases. Since the frequency of updates is publicly observable, we must do this in a privacy-preserving fashion.

The idea is as follows. We implement a (pan-private) internal *buffer* for the algorithm. We only update the output count when this buffer gets "flushed" (reducing the frequency of updates). The buffer keeps track of (a privacy-preserving and somewhat accurate approximation to) the number of 1's in the data stream since the last flush. Whenever this number hits a certain threshold ℓ, the buffer is flushed and reset to 0, and an update occurs. Thus, the buffer determines *when* updates occur (but not *what* the update is). To update the output count, we also keep a separate *accumulator* that tracks the *exact* number of 1's that have occurred in the input stream since the last flush (this accumulator is not pan-private). When an update occurs we add the value in the accumulator plus noise $\mathrm{Lap}(1/\varepsilon)$ to the output counter, and reset the accumulator. Finally, to assist with accurace on dense streams, a buffer flush is also triggered if there have been a large number of updates since the previous flush.

The poor performance of the simple solution on sparse streams is ameliorated because the frequency of updates is greately reduced. Since (Laplacian) noise is only added during an update, this reduces the noise.

The intuition for privacy is that the *step times* of updates are not disclosive because the buffer is differentially private, and the *content* of the updates preserves privacy because of the additional noise being added to the accumulator.

This is the basic idea behind the cascading buffers algorithm. One question remains, however: How do we implement the initial privacy-preserving "somewhat accurate" buffer? This in itself is a counting task! The first idea is to implement the buffer using the simple solution. This results in significantly improved accuracy. The error, over t steps, shrinks to about $O((\sqrt{n} + t^{1/4})/\varepsilon)$ (where n is the number of 1's in the data stream, and for the right choice of the threshold ℓ). For sparse data streams this is much better than $\sqrt{t}/\varepsilon$ of the simple solution. To get even better accuracy we can (essentially) now recursively use this new and more

accurate counter to implement a more accurate buffer, and get an even more accurate counter. This leads to a "cascading buffers" algorithm, with d levels of recursive buffering. The output level is $d+1$.

We note that the algorithm just described is not pan-private, because the accumulators hold sensitive information. It can easily be made pan-private at only a small cost to accuracy, as follows. On initialization and every reset, add to the accumulator a fresh draw from $\text{Lap}(1/\varepsilon)$.

We also note that repeated announced intrusions can be handled by flushing the buffer and accumulator after every intrusion (the error grows with the number of intrusions).

The full, pan-private, algorithm is presented in Figure 2. Theorem 6.1 and Corollary 6.1 give precise statements of its performance.

Cascading Buffers Counter$(\varepsilon, d, \ell, \kappa)$

Init. Initialize the output counter *ocount* $\leftarrow 0$. For $i \leftarrow 1 \dots d$ initialize the i-th level's state, including:

- The accumulator (accumulated count since the last flush), $a_i \leftarrow \text{Lap}(1/\varepsilon)$
- The buffer, $b_i \leftarrow 0$
- The number of updates since last flush, $u_i \leftarrow 0$

Processing. In each step t, on input $x_t \in \{0,1\}$, for each level $1 \le i \le d$, update the accumulator for level i: $a_i \leftarrow a_i + x_t$. Now, treat the input as a buffer update for level 1 with input x_t and proceed as follows.

Buffer update for level i, input x. On an update for level i with input $x \in \{0,1,\dots\}$, update the buffer $b_i \leftarrow b_i + x + \text{Lap}(1/\varepsilon)$, and the number of updates since the last flush $u_i \leftarrow u_i + 1$.
If $b_i \ge \ell$ (buffer overflow) or if $u_i = (\ell \cdot \varepsilon/4\kappa)^2$ (there have many updates since the last flush), then flush the buffer:

- Update the buffer of the next level $i+1$:
  For the output level ($i+1 = d+1$), update the output counter *ocount* $\leftarrow$ *ocount* $+ a_i + \text{Lap}(1/\varepsilon)$
  For $i < d$, run the buffer update procedure for level $i+1$ with input a_i
- Reset the accumulator $a_i \leftarrow \text{Lap}(1/\varepsilon)$, the buffer $b_i \leftarrow 0$, and the number of updates since last flush $u_i \leftarrow 0$

Figure 2: **Cascading Buffers Counter**

THEOREM 6.1. [9]

The counter of Figure 2 is an event-level pan-private counter with $((2d+1)\cdot\varepsilon)$-differential privacy. At any step t, for an input stream with correct output n_t, with all but $exp(-\kappa)$ probability over the counter's coins, the additive error is $O((d\ell)+(\sqrt{t}/(\ell\varepsilon/4\kappa)^d)+(\sqrt{n}/(\ell\varepsilon/\kappa)))$.

Proof. (Sketch.) We first sketch the proof of privacy, then address accuracy.

Privacy. To argue event-level pan-privacy, recall the processing of each input x_t: (i) It is added to level 1's buffer together with noise $\text{Lap}(1/\varepsilon)$ that was drawn independently specifically for x_t. This guarantees ε-differential event-level pan- privacy of level 1's buffer. (ii) It is added to the d accumulators. Since each of these is initialized with (independent) noise drawn from $\text{Lap}(1/\varepsilon)$, these accumulators maintain $d\varepsilon$-differential event-level pan-privacy. We conclude that the internal state of the algorithm at any single point in time is in fact $((d+1)\varepsilon)$-differential event-level pan-private.

This alone does not suffice, as an adversary can also observe the counter's continual observations before and after intruding into its internal state. The counter does not add new independent noise separately for each x_t added to the accumulators (noise is only added on initializations and after a flush occurs), and so (since we cannot add more noise after an undetected intrusion) information learned during an intrusion, taken together with the continual outputs, might lead to a privacy breach.

To see that this does not occur, observe that the counter only uses the accumulators when buffer flushes occur. Whenever an accumulator is used, the counter adds to that accumulator fresh independently drawn noise from $\text{Lap}(1/\varepsilon)$ (and then resets it). This guarantees event-level pan-privacy even given the continual observations. More formally, we have already argued $(d+1)\cdot\varepsilon$ event-level pan-privacy of the algorithm's internal state at any point in time. Now considering also the continual outputs, we observe that each x_t affects level 1's buffer (we have already accounted for this privacy loss) and the d accumulators. In turn, the d accumulators affect the output (when flushes occur) by being added to the buffers of layers $2 \dots d$ and to the output count. However, whenever an accumulator is added to a buffer or the output count, it is added with independently drawn noise from $\text{Lap}(1/\varepsilon)$. This guarantees that the additional privacy loss incurred from the continual observations is at most $d \cdot \varepsilon$. In total, we conclude that the counter is $(2d+1)\cdot\varepsilon$-event-level pan-private.

Accuracy. Accuracy will follow from the following two claims:

CLAIM 6.1. *For a length T stream prefix with n 1's, with all but $\exp(-\kappa)$ probability, the number of updates to the output counter ocount is $O((T/(\ell\cdot\varepsilon/4\kappa)^{2d}) + (n/(\ell\cdot\varepsilon/\kappa)^2))$.*

CLAIM 6.2. *For any input stream, step t and level $i \in \{1, \ldots, d\}$, let n_i be the number of 1's that have appeared in the input stream since level i's buffer was last flushed. With all but $\exp(-\kappa)$ probability, the difference between the buffer b_i and n_i is at most $2i \cdot \ell$.*

Both claims are proved by induction on the level number, i. The key to the proof of Claim 6.1 is to show that

$$B_i = T/(\ell \cdot \varepsilon/4\kappa)^{2i} + 2n/(\ell \cdot \varepsilon/4\kappa)^2$$

is, with high probability, an upper bound on the number of flushes of level i (or updates to $i+1$), when handling a stream prefix of length n. (The output counter *ocount* is level $d+1$.) The basis of the induction is straightforward. For a higher level, $i+1$, an update occurs every time level i's buffer flushes. The proof for level $i+1$ bounds separately the number of flushes for level i that occur because the buffer grows to ℓ, and the number of flushes that occur because there have been too many updates.

The key to the proof of Claim 6.2 is the observation that the algorithm satisfies the following invariant: At every step t, with overwhelming probability, simultaneously at every level i, the total amount of noise added to the buffer at level i (both via noise added to the accumulator in level $i-1$ and via noise added to the buffer itself when level $i-1$ flushes) since the buffer was last flushed is at most $\ell/2 < \ell$. The proof also uses the fact that immediately after buffer i is flushed, there are no 1's "trapped" in lower levels of the cascade.

COROLLARY 6.1. *Let T be a bound on the number of steps (length of the stream prefix to be processed). Initializing the counter of Figure 2 with $\varepsilon' = \varepsilon/\log T$, $d = O(\log T)$, $\ell = O(\log^3 T/\varepsilon)$ and $\kappa = \log^2 T$, we get an ε- event-level pan-private counter. With all but $\nu(T)$ probability, the error in every step t is at most $O((\sqrt{n_t} + \log^4 T)/\varepsilon)$, where n_t is the number of 1's in the stream prefix of length t.*

7 Continual Observation Density Estimation

In this section we use the (event-level) counter to convert the (user-level) single output density estimator of Section 4 to obtain a user-level pan-private continual observation density estimator.

The intuition is as follows. The algorithm will have two data structures. The first is a table of bits, one per element in M, a randomly chosen subset of the universe X. This is as in the single output density estimator, and the bits are initialized to independent draws from $\mathcal{D}_0$ (unbiased bits). The second data structure is a counter, as in Section 6.

We denote by x_t the tth element in the stream. A special "blank" symbol, $\perp$, is used to mean "nothing happened." Formally, the stream is a sequence of elements of $X' = X \cup \{\perp\}$.

Inputs to the counter are generated by the algorithm; the tth input to the counter, denoted $y_t \in \{0,1\}$, will be drawn, using fresh randomness, from one of three distributions: $\mathcal{D}_0$, $\mathcal{D}_+$, and $\mathcal{D}_-$, where:

- $\mathcal{D}_0$ is the unbiased distribution. This will be invoked on elements $x \notin M$. The expected contribution of these over t steps is $t/2$.
- $\mathcal{D}_+$ is biased toward 1: it assigns probability $1/2 + \varepsilon + \varepsilon^2/2$ to 1. It is used when processing $x \in M$ where $b_x = 0$.
- $\mathcal{D}_-$ is biased toward 0: it assigns probability $1/2 - \varepsilon + \varepsilon^2/2$ to 1. It is used when processing $x \in M$ where $b_x = 1$.

Recall from Section 4 that $\mathcal{D}_1 = \mathcal{D}_1(\varepsilon)$ is a distribution on $\{0,1\}$ assigning probability $1/2 + \varepsilon/4$ to 1.

Continual Observation Density Estimator $(\varepsilon, \alpha, \beta, T)$

Init. Initialize a counter that is polylog(T)-accurate and ε-event level pan-private (see Corollary 6.1). Sample at random a set M of $m = \text{poly}(1/\varepsilon, 1/\alpha, \log(1/\beta))$ elements (representatives) in X. Create a table of size m with a single one-bit entry for each item in M. For every entry $x \in M$, generate a random initial value $b_x \sim \mathcal{D}_0$.

Processing. In step t, let x_t be the current input. Generate an update value $y_t \in \{0,1\}$ for the counter:

- If x_t is $\perp$ ("nothing happened") or $x_t \notin M$, then choose y_t to be a uniformly random bit.
- Otherwise, if the current input value is $x_t \in M$, let b_{x_t} be x_t's entry in the table. If $b_{x_t} = 0$, then choose $y_t \sim D_+$. If $b_{x_t} = 1$, then choose $y_t \sim D_-$.

Update the counter with update value y. Update x's entry in the table by drawing it from $\mathcal{D}_1$: $b_{x_t} \sim \mathcal{D}_1(\varepsilon)$. Finally, let *ocount* be the counter's current output. The density estimator's output is $(\textit{ocount} - t/2)/(\varepsilon^2/2)$.

Figure 3: Continual Observation Density Estimator

THEOREM 7.1. [9] *The continual observation density estimator of Figure 3 is $O(\varepsilon)$-user level pan-private. With all but β probability, its additive error at step t is at most $O((\sqrt{t} + \alpha)/\varepsilon^2)$.*

Proof. (Sketch.) The key to privacy under continual observation is the following claim.

CLAIM 7.1. *In every step t with input x_t, the distribution of y_t (used to update the counter) is as follows: (i) If x_t is blank ($\perp$) or $x_t \notin M$, then y_t is a uniformly random bit. (ii) If $x_t \in M$ and this is not the first appearance of x_t in the data stream, y_t is a uniformly random bit (independently of any y_t values generated for previous appearances of x_t). (iii) If $x_t \in M$ and this* is the first appearance of x_t, *then y_t is 1 with probability $1/2 + \varepsilon^2/2$ and 0 with probability $1/2 - \varepsilon^2/2$.*

Proof. When the input is either a blank symbol, or an input $x_t \notin M$, the distribution of the counter update value y_t is drawn uniformly from $\{0, 1\}$, so it is unbiased.

When the input is an item $x_t \in M$, then if x_t has appeared previously in the data stream its bit in the table b_{x_t} is drawn from $\mathcal{D}_1$. The probability that y_t is 1 is thus:

$$(\frac{1}{2} - \frac{\varepsilon}{4})(\frac{1}{2} + \varepsilon + \frac{\varepsilon^2}{2}) + (\frac{1}{2} + \frac{\varepsilon}{4})(\frac{1}{2} - \varepsilon + \frac{\varepsilon^2}{2}) = \frac{1}{2}.$$

Thus, y_t is a uniformly random bit. The distribution of y_t is independent of the previous y values obtained on x_t inputs, because b_{x_t} was chosen independently of the previous value in x_t's entry of the table.

When the input is an item $x_t \in M$ that *has not appeared before*, then $b_{x,t}$ is drawn from $\mathcal{D}_0$ (it is uniformly random). This means that y_t is 1 with probability:

$$\frac{1}{2}(\frac{1}{2} + \varepsilon + \frac{\varepsilon^2}{2}) + \frac{1}{2}(\frac{1}{2} - \varepsilon + \frac{\varepsilon^2}{2}) = \frac{1}{2} + \frac{\varepsilon^2}{2}.$$

It follows from the claim that the inputs to the counter are simply independent and uniformly random bits, except when an item appears for the first time, at which point the input to the counter is biased towards 1. This means that the appearance of an input item in the data stream can only generate one significant event: its first appearance. The event-level pan-privacy of the counter will protect this single event. Combined with the user-level privacy of the density estimator this will guarantee user-level pan-privacy. (A formal argument studies the difference in probability distributions on internal states and output sequences on X'-adjacent stream prefixes.)

Accuracy of the continual density estimator will be "inherited" from the accuracy of the counter. In particular, with overwhelming probability, the counter's output *ocount* should be within a polylog$T$ error of the correct number of 1's that it received as input. By the accuracy of the density estimator, we know that the difference between the fraction of 1's in the table and the actual density is at most α. If the density of the stream at step t is m_t, by Claim 7.1 we know that the number of 1's given as input to the counter is with overwhelming probability within a $O(\sqrt{t}+\alpha)$-error of $t/2+(\varepsilon^2/2)\cdot m_t$.

We conclude that the density estimator's output $(ocount - t/2)/\varepsilon^2$ has (with overwhelming probability) error at most $O((\sqrt{t} + \alpha)/(\varepsilon^2/2))$.

8 Conclusions

We have described two new settings for differential privacy, motivated by real life concerns. Although the results described here are a (strict) subset of those obtained in [10, 9], there is still much more to be done: (1) [9] initiates a study of continual observation differentially private data structures, built on the counter. Necessarily, this first paper just scratches the surface of what can be achieved. Similarly, [10] initiates a study of pan-privacy to provide an added level of protection, this time against intrusions. We do not know the range of, and limitations on, what can be achieved pan-privately with decent accuracy.
(2) Continuing the original motivation of [10], formalizing the prevention of, or reasonable limitation on, the re-purposing of data, or "mission creep," and exploring the space of what can be achieved under these controls, could be of enormous social value.

Acknowledgement. The author thanks Guy Rothblum for many helpful discussions during the preparation of this paper.

References

[1] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: The SuLQ framework. In *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, June 2005.

[2] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 202–210, 2003.

[3] C. Dwork. A firm foundation for private data analysis. *Communications of the ACM (to appear)*.

[4] C. Dwork. An ad omnia approach to defining and achieving private data analysis. In F. Bonchi, E. Ferrari, B. Malin, and Y. Saygin, editors, *Privacy, Security, and Trust in KDD, First ACM SIGKDD International (PinKDD), Revised Selected Papers*, volume 4890 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2007.

[5] C. Dwork. The differential privacy frontier. In *Proceedings of the 6th Theory of Cryptography Conference (TCC)*, 2009.

[6] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: privacy via distributed noise generation. In *Advances in Cryptology: Proceedings of EUROCRYPT*, pages 486–503, 2006.

[7] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the 3rd Theory of Cryptography Conference*, pages 265–284, 2006.

[8] C. Dwork, F. McSherry, and K. Talwar. The price of privacy and the limits of lp decoding. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, pages pp. 85–94, 2007.

[9] C. Dwork, M. Naor, T. Pitassi, and G. Rothblum. Differential privacy under continual observation. Manuscript in preparation, 2009.

[10] C. Dwork, M. Naor, T. Pitassi, G. Rothblum, and S. Yekhanin. Pan-private streaming algorithms. Manuscript submitted for publication, 2009.

[11] C. Dwork and K. Nissim. Privacy-preserving datamining on vertically partitioned databases. In *Proceedings of CRYPTO 2004*, volume 3152, pages 528–544, 2004.

[12] C. Dwork and S. Yekhanin. New efficient attacks on statistical disclosure control mechanisms. In *Proceedings of CRYPTO 2008*, pages 468–480, 2008.

[13] M. Hardt and K. Talwar. On the geometry of differential privacy. arXiv:0907.3754v2, 2009.

[14] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual Symposium on Foundations of Computer Science*, 2007.

[15] S. Warner. Randomized response: a survey technique for eliminating evasive answer bias. *JASA*, pages 63–69, 1965.