Differential Privacy in Streaming & Cloud Computation

Kavish Shah, Ernest Ng

December 27, 2022

Contents

1 Abstract

Differential Privacy (DP) has gained widespread attention in recent years as a tool for ensuring data protection in a variety of settings. In this survey, we examine the literature on differentially private algorithms in stream processing and cloud computation settings. We begin by reviewing key definitions regarding differential privacy. We then discuss notions such as pan-privacy and continual observation which are necessary for the study of DP in stream processing. Next, we consider sketch algorithms, which allow for the efficient processing of large datasets while preserving privacy, as well how DP can be achieved in adversarially robust streaming algorithms. Finally, we review various programming frameworks for achieving DP in streaming and MapReduce computations, and discuss several open problems that remain in this field.

2 Preliminaries

2.1 Differential Privacy basics

The aim of differential privacy is to ensure that removing one record from a database results in a limited impact on the result of any statistical analysis performed over said database. *Adjacent* databases are defined as databases which are identical except for one particular data entry. Consider two databases D_1 and D_2 which differ by one element, such that one is a proper subset of the other and the larger database contains just one additional row. The following definition is due to Dwork et al. [9]:

Definition 1. *A randomized function $\mathcal{K}$ is ε-**differentially private** if for all databases D_1 and D_2 differing by at most one element, and all $S \subseteq Range(\mathcal{K})$, we have that:*

$$\Pr[\mathcal{K}(D_1) \in S] \leq \exp(\varepsilon) \times \Pr[\mathcal{K}(D_2) \in S]$$

The probability is taken is over the coin tosses (internal randomization) of $\mathcal{K}$.

The intuition for this definition is that for any particular individual, the inclusion or exclusion of their datum should not significantly impact the output of the computation performed by $\mathcal{K}$. This gives a sense of plausible deniability because no third party can infer whether a particular individual was a part of the dataset purely by examining the output for the computation. Note that ε is a parameter dictating the strength of the privacy guarantees, where smaller values of ε represent stronger privacy.

Event-level and User-level privacy
Note that a single user can have multiple distinct entries in the stream or no entries at all. This raises two types of privacy risks. First is an *event-level* risk where the outputs from the algorithm could reveal if a particular user has a specific entry in the stream. The second type of risk is that the output would reveal if the user has an entry in the data stream or not, which is called *user-level* risk.

The following example highlights the need for both event-level and user-level privacy.

1.) Suppose we wish to monitor the incidence of a disease in a population. If users were to report their symptoms to a disease self-assessment website, we wish to ensure **event-level** privacy, i.e. we wish to hide the presence / absence of a single event (the user's visit to the website).

2.) Suppose we are monitoring search queries in a medical search engine. Here we want **user-level** privacy (ensuring that all search queries from a single user are protected).

Global Sensitivity
The *global sensitivity* of a function f is the maximum change in the output due to a single change in the input dataset. Dwork et al. [11] formally define the global sensitivity as follows:

Definition 2. *For $f : \mathcal{D} \to \mathbb{R}^d$, the **global sensitivity** of f is*

$$GS_f = \max_{D,D'} \left\| f(D) - f(D') \right\|_1$$

for all neighboring data sets D and D'.

Laplace Mechanism
The Laplace mechanism works by adding noise to the results of a statistical analysis sampled form of a Laplace distribution. This is a unimodal distribution centered at zero and is characterized by its scale parameter b, which determines the amount of noise added to the results. The Laplace probability distribution Lap(b) has probability density function $f(x) = \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right)$, where $x \in \mathbb{R}$. The following result due to Roth & Dwork [10] demonstrates how the Laplace mechanism can guarantee ε-differential privacy:

Theorem 1 (Laplace mechanism [11]). *Let $f : X^* \to \mathbb{R}$ be a sensitivity ℓ function. The mechanism that on input $S \in X^*$ returns*

$$f(S) + \text{Lap}\left(\frac{\ell}{\varepsilon}\right)$$

achieves ε-differential privacy.

2.2 Elementary DP Properties

We first present several elementary results from the DP literature regarding the composition of DP algorithms.

Composition & Post-Processing:
Let x denote a database (collection of records), whose records originate from some universe $\mathcal{X}$. Following the canonical statements of composition theorems in the DP literature [10], we represent a database by its histogram, i.e. we have $x \in \mathbb{N}^{|\mathcal{X}|}$, where each component of the tuple x denotes the no. of occurrences of some $x_i \in \mathcal{X}$ in the database x.

We first note that we can compose differentially-private algorithms in a manner such that the privacy loss is controlled, using the following theorem:

Theorem 2 (**Composition**, Theorem 3.14 from [10]). *Let $M_1 : \mathbb{N}^{|\mathcal{X}|} \to R_1$ be an ε_1-differentially private algorithm, and let $M_2 : \mathbb{N}^{|\mathcal{X}|} \to R_2$ be an ε_2-differentially private algorithm. Then their composition $M_{1,2} : \mathbb{N}^{|\mathcal{X}|} \to R_1 \times R_2$, defined as $M_{1,2}(x) := (M_1(x), M_2(x))$ is $(\varepsilon_1 + \varepsilon_2)$-differentially private.*

Now, given a computational graph composed of k different differentially private computations, repeatedly application of the composition theorem above gives us the following result regarding the privacy of the overall computation:

Theorem 3 (Corollary 3.15 from [10]). *Suppose mechanisms $\mathcal{K}_i, i \in [r]$ each provide ε_i-differential privacy. Then, the overall mechanism $\mathcal{K}$ consisting of the composition of these r mechanisms is $\left(\sum_{i\in[r]} \varepsilon_i\right)$-differentially private.*

We also note that differential privacy is preserved under post-processing via a data-independent mapping f:

Theorem 4 (**Post-Processing**, Proposition 2.1 from [10]). *Let $M : \mathbb{N}^{|\mathcal{X}|} \to R$ be a randomized algorithm that is ε-differentially private. Let $f : R \to R'$ be any arbitrary randomized mapping.*
Then $f \circ M : \mathbb{N}^{|\mathcal{X}|} \to R'$ is ε-differentially private.

Generalization properties:
Stemmer [26] discusses the generalization properties of DP, which are used by Hassidim et al's design of adversarially robust streaming algorithms [18], discussed in section 3.3.

Let $\mathcal{D}$ be some distribution over the universe $\mathcal{X}$, and let $h : \mathcal{X} \to \{0,1\}$ be some predicate. Let $S \sim \mathcal{D}^n$ be some sample consisting of n elements drawn i.i.d from $\mathcal{D}$. Then, using Chernoff-Hoeffding bounds, we know that with high probability, the empirical average of h on S is close to the expectation of $h(x)$ where x is drawn from $\mathcal{D}$ (from [26]), that is:

$$\text{with high probability, } \frac{1}{|S|}\sum_{x\in S} h(x) \approx \mathbb{E}_{x\sim\mathcal{D}}[h(x)]$$

Now, suppose $\mathcal{A} : \mathcal{X}^n \to 2^{\mathcal{X}}$ is any arbitrary algorithm that takes a sample $S \sim \mathcal{D}^n$ and outputs a predicate $h \leftarrow \mathcal{A}(S)$. Stemmer [26] states that in general, we cannot claim that the empirical average of h on S is close to expectation of h, since the algorithm $\mathcal{A}$ could just naively return the predicate that returns 1 only when $x \in S$ and overfit on the sample S.

However, it turns out that if $\mathcal{A}$ is a differentially private algorithm, then we can conclude that with high probability, the empirical average of h on S will be close to the expectation of h over $\mathcal{D}$, thanks to the following result discussed by Hassidim et al. [18]:

Theorem 5 (**Generalization properties of DP**). *Let $\mathcal{A} : \mathcal{X}^n \to 2^{\mathcal{X}}$ be an ε-differentially private algorithm that returns a predicate $h : \mathcal{X} \to \{0,1\}$. Let $\mathcal{D}$ be a distribution over $\mathcal{X}$ and let S be a sample consisting of n i.i.d elements drawn from $\mathcal{D}$. Let $h \leftarrow \mathcal{A}(S)$ be the predicate obtained by running algorithm $\mathcal{A}$ on the sample S.*
Then, with high probability, we have that:

$$\frac{1}{|S|}\sum_{x\in S} h(x) \approx \mathbb{E}_{x\sim\mathcal{D}}[h(x)]$$

2.3 Data Streams

Below, we adopt the notation for data streams used by Mir et al. [23]. Suppose data records originate from a universe $\mathcal{U}$, where $|\mathcal{U}| = m$. An update is defined as an ordered pair $(i, d) \in \mathcal{U} \times \mathbb{Z}$. Consider a semi-infinite sequence of updates $(i_1, d_1), (i_2, d_2), \ldots$; the input for all our algorithms consists of the first t updates, denoted $S_t = (i_1, d_1), \ldots, (i_t, d_t)$. Here, $i_j \in [n]$ refers to the j^{th} element and $d_j \in \mathcal{Z}$ is the weight. The state after t updates is an m-dimensional vector $\mathbf{a}^{(t)}$, indexed by the elements in $\mathcal{U}$ (we will omit the superscript when it is clear from the context). The elements of the vector $\mathbf{a} = \mathbf{a}^{(t)}$, referred to as the state vector, are defined as follows:

$$a_i = \sum_{j:i_j=i} d_j$$

Note that, d_i could be either positive or negative. If $d_i \geq 0$, the model is called a *cash register* model (or *insertion-only model)*, whereas if there is no restriction on d_i, it is called a *turnstile model*.

2.4 Adjacency in Data Streams

If we want to establish differential privacy guarantees on data streams, we will need to establish a notion of adjacency. Two streams can be considered adjacent if the entries corresponding to a particular user are replaced and everything else remains the same. This is formally defined in [23] as follows:
Consider two online updates $S = \{(i_1, d_1), \ldots, (i_t, d_t)\}$ and $S' = \left\{\left(i'_1, d'_1\right), \ldots, \left(i'_{t'}, d'_{t'}\right)\right\}$ associated with state vectors $\mathbf{a}$ and $\mathbf{a}'$ respectively.

Definition 3. *S and S' are said to be neighbors if there exists a (multi)set of updates in S indexed by $K \subseteq [t]$ that update the same ID $i \in \mathcal{U}$, and there exists a (multi)set of updates in S' indexed by $K' \subseteq [t']$ that updates some $j(\neq i) \in \mathcal{U}$ such that $\sum_{k \in K} d_k = \sum_{k \in K'} d'_k$ and for all other updates in S and S' indexed by $Q = [t] - K$ and $Q' = [t'] - K'$ respectively,*

$$\forall i \in \mathcal{U} \sum_{k \in Q,\, s.t.\ i_k = i} d_k = \sum_{k \in Q',\, s.t.} \sum_{i'_k = i} d'_k$$

Note that in the definition above, t and t' do not necessarily have to be equal since we allow the d_i's to be integers. This definition implies that if we swap out all entries of one particular user in S with entries of some other user and call the resultant stream S', then S and S' are considered adjacent streams (neighbors). We allow t and t' to be different because the update weights can be broken up into multiple updates or combined into one.

2.5 k-wise Independent Hash Functions

The literature on differentially private stream algorithms often feature the use of k-wise independent hash functions as a way of hashing data records. Here we provide the definition of such functions:

Definition 4. *A k-**wise independent hash function** $h : [1..m] \to [1..T]$ is a randomized function where, for any k distinct elements $j_1, \ldots, j_k \in [1..m]$ and any k possible values $t_1, \ldots, t_k \in [1..T]$, we have that:*

$$\Pr[h(j_i) = t_i] = \frac{1}{T^k} \quad \textit{for all } i$$

where the probability is over the randomization of the function h.

The significance of k-wise independent hash functions is that they can be stored with relatively small space, providing space complexity gains [24]. For example, for prime T, consider the function $h : [1..m] \to [1..T]$ given by:

$$h(j) = (a \cdot j + b) \mod T$$

where a, b are chosen independently and uniformly at random from $[1..T]$.

One can show that this function is indeed a 2-wise (pairwise) independent hash function by finding distinct $j, j' \in [1...m]$ where $j \neq j'$ such that $h(j) = s$ and $h(j') = s'$, for some fixed but arbitrary $s, s' \in [1..T]$. This is analogous to solving the two following equations simultaneously:

$$\begin{cases} a \cdot j + b = s \mod T \\ a \cdot j' + b = s' \mod T \end{cases}$$

Using canonical results from abstract algebra, we know that if T is prime, then $\mathbb{Z}/T\mathbb{Z}$ is a field. Then, expressing the two equations above as a linear system over the finite field $\mathbb{Z}/T\mathbb{Z}$ and solving for a and b using

linear-algebraic techniques, we see that there exist unique $a^*, b^* \in [1..T]$ satisfying the above two equations. Then, by independence, since a, b are chosen uniformly at random from $[1..T]$, the joint probability of the event that $h(j) = s$ and $h(j') = s$ is given by:

$$\Pr[h(j) = s \wedge\ h(j') = s'] = \Pr[a = a^* \wedge\ b = b^*] = \frac{1}{T^2}$$

This shows that h is indeed a 2-wise independent hash function. Note that this 2-wise independent hash function only requires $O(\log T)$ space to store. This is because we only need to remember $a, b \in [1..T]$ to compute the hashed value above, and storing the random seed (a, b) as a binary string requires $O(\log T)$ bits [21]. [1] Analogously, one can show that any k-wise independent hash function requires $O(k \log T)$ space to store. Such a function can be a polynomial of degree k with coefficients chosen independently and uniformly at random and from $[1..T]$, from which we take its value modulo T [24].

3 Theoretical Results

3.1 Pan-Privacy

Note that the internal state of a streaming algorithm may contain information about the updates from a specific time interval. Ideally, we would like to provide privacy guarantees against *intrusions*, for example when the algorithm's internal state is exposed to an adversary, or a subpoena entails the disclosure of the algorithm's internal state.

Pan-private algorithms aim to ensure differential privacy even when the internal state of the algorithm is exposed via an intrusion. Additionally, pan-privacy protects against situations of *mission creep* where data collected for one purpose is used for another purpose [10].

The following definition of user-level pan-privacy is due to Dwork & Roth [23]:

Definition 5. *Say that an algorithm **Alg** mapping data stream prefixes to $I \times \sigma$ is **pan-private against a single intrusion** if for all subsets $I' \subseteq I$ of internal algorithm states & all subsets $\sigma' \subseteq \sigma$ of output sequences, as well for all pairs S, S' of adjacent data-stream prefixes, we have that:*

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

where the probability is taken over the internal coin flips (randomization) of the algorithm Alg.

This definition assumes that the adversary can observe internal states in I and output sequences in σ, but it cannot see the individual data records in the stream. However, the adversary may have auxiliary knowledge about the data obtained from other sources. Additionally, the adversary has arbitrary computational power.

One should also note that this definition refers only to one single intrusion. If there are multiple (unannounced) intrusions, we need to consider the different ways in which the observations of internal algorithm states and output sequences can be interleaved.

In the following section, we discuss the problem of pan-private density estimation and illustrate how pan-privacy can be achieved.

3.1.1 Pan Private Density Estimation

The problem of **density estimation** is as follows: Given a universe $\mathcal{X}$ of data records and a data stream σ, we wish to estimate the fraction of items in $\mathcal{X}$ which appear at least once in σ. Dwork et al. propose a density estimation algorithm, which aims for *user-level* privacy [13]. The challenge of density estimation is that the algorithm has to maintain information regarding items' previous appearances, without the algorithm's internal state leaking significant information about the appearance or absence of individual items.

Note that a stream is a sample of the universe $\mathcal{X}$ (with replacement). Thus, we can can sample a random set $M \subseteq \mathcal{X}$ consisting of m items, where m is sufficiently large such that with high probability, the density of M's items in the data stream approximate the density of X's items.

Then, we maintain a table, with one entry per item in M. For each item $x \in M$ that is absent from the stream, its corresponding table entry is drawn from a distribution $\mathcal{D}_0(\varepsilon)$ over $\{0, 1\}$. For items that have appeared at least once in the stream, their entry is drawn from another distribution $\mathcal{D}_1(\varepsilon)$ over $\{0, 1\}$ regardless of the item's no. of appearances.

[1] Unless otherwise stated, all logarithms discussed in this document are base-2 logarithms. That is, we write $\log(n)$ to denote $\log_2(n)$.

The two distributions $\mathcal{D}_0(\varepsilon)$ and $\mathcal{D}_1(\varepsilon)$ are defined as follows:

$$\mathcal{D}_0(\varepsilon) = \begin{cases} 0 & w.p.\ \frac{1}{2} \\ 1 & w.p.\ \frac{1}{2} \end{cases} \qquad \mathcal{D}_1(\varepsilon) = \begin{cases} 1 & w.p.\ \frac{1}{2} + \frac{\varepsilon}{4} \\ 0 & w.p.\ \frac{1}{2} - \frac{\varepsilon}{4} \end{cases}$$

Then, the algorithm computes θ, the fraction of entries in the table with value 1 (call these entries *1-entries*). Lastly, the algorithm outputs the density value f' defined as follows:

$$f' = \frac{4(\theta - \frac{1}{2})}{\varepsilon} + \text{Lap}\left(\frac{1}{\varepsilon m}\right)$$

We now discuss the intuition for this algorithm's privacy and correctness. Since we re-draw an item x's entry from $\mathcal{D}_1(\varepsilon)$ whenever x appears in the data stream, the distribution of x's entry is unaffected by its no. of appearances in the stream (beyond the first appearance). This means that the entry corresponding to x does not yield any information about the no. of times x occured in the stream.

Dwork et al. show that for $\varepsilon \leq 1/2$, the two distributions $D_0(\varepsilon)$ and $D_1(\varepsilon)$ are ε-differentially private, i.e. sufficiently close to each other [13]. We formalize this intuition using the following lemma:

Lemma 1 (Theorem 3.1 in [13])**.** *For $\varepsilon \leq 1/2$, the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are ε-differentially private. That is, for both values $b \in \{0,1\}$, we have that:*

$$e^{-\varepsilon} \leq \frac{\Pr_{\mathcal{D}_1}[b]}{\Pr_{\mathcal{D}_0}[b]} \leq e^{\varepsilon}$$

Proof. We first examine the ratio of the probability weight of 1 in the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$, and note that for any $0 \leq \varepsilon \leq 1/2$, we have that:

$$\frac{\Pr_{\mathcal{D}_1}[1]}{\Pr_{\mathcal{D}_0}[1]} = 1 + \frac{\varepsilon}{2} \leq e^{\varepsilon}$$

Similarly, the ratio of the probability weights of 0 in $\mathcal{D}_0$ and $\mathcal{D}_1$ is:

$$\frac{\Pr_{\mathcal{D}_1}[0]}{\Pr_{\mathcal{D}_0}[0]} = 1 - \frac{\varepsilon}{2} \leq e^{\varepsilon}$$

Together, the above two inequalities demonstrate ε-differential privacy. □

Dwork et al. then present the following result regarding this algorithm's privacy and accuracy:

Theorem 6. *The density estimation algorithm ensures 2ε-differential pan-privacy for $\varepsilon \leq 1/2$. For a fixed input, with probability $1 - \beta$ over the algorithm's internal randomization, the algorithm's output is α-close to the fraction of items in X that appear in the data stream. The algorithm uses space* $\text{poly}(1/\varepsilon, 1/\alpha, \log(1/\beta))$.

Proof Sketch Here we provide a sketch of the proof (we refer the reader to [13] for the complete proof). We know that ε is chosen such the two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are sufficiently close to each other. This means that users in M are guaranteed ε-differential privacy. Moreover, the users not in M enjoy full privacy as they do not appear in the algorithm's internal state. These two findings ensure that the algorithm's internal state remains private.

Now, when the algorithm generates its output, the sensitivity of this output to the presence/absence of any user is at most $\frac{1}{m}$ if the user is in M, or 0 if the user isn't in M. Moreover, we can achieve ε-differential privacy for the output by adding noise sampled from the Laplace distribution. Thus, by composition theorems, the entire computation achieves 2ε-differential privacy.

We set the size of our sample M to be $m = \text{poly}(1/\alpha, \log(1/\beta))$. Using Chernoff bounds, we can establish that with high probability, most of the items that appear atleast once in the stream also appear atleast once in the sample M. Next, we use Chernoff bounds again to show that the empirical fraction f' of 1-entries in the table contributed via draws from $\mathcal{D}_1$ is close to the true fraction f of 1-entries in the table. This leads to a bound on the error for the density in the form $|f - f'| \leq \alpha$.

Extensions The density estimation algorithm can be extended to provide multiplicative errors and to handle multiple announced intrusions. To obtain a multiplicative error, Dwork et al. [13] propose hashing via 4-wise independent hash functions (see section 2.5). The authors hash the universe X into a chain of $l = \log|X|$ smaller sets

$$X_0 \subset X_1 \subset \ldots \subset X_l = X$$

where $|X_i| = 2^i$. Then, the density estimation algorithm is run independently for each X_i (each with its own choice of m) with noise drawn from $\text{Lap}\left(\frac{l}{\varepsilon \cdot m}\right)$ in the output step. An appropriate output is then chosen from the l outputs, and examining the choice of hash function via Chebyshev's inequality yields the desired multiplicative error term [13].

Handling multiple intrusions We have shown that the density of a stream can be estimated in a pan-private way for a single unannounced intrusion. To handle multiple *announced* intrusions, the algorithm re-randomizes the aforementioned distributions in order to inject new noise and maintain privacy.

However, two *unannounced* intrusions are far more challenging to tackle. Dwork et al. [13] demonstrate how the threat of multiple intrusions forces the algorithm to behave in a way where it cannot distinguish between a stream containing just one element repeated for the entire stream, and a stream which contains individual elements for a large time period and then just contains one elements repeated for the rest of the stream.

3.2 Sketch Algorithms

Sketch algorithms are a class of algorithms that are used to process and analyze large datasets in a privacy-preserving manner. They work by creating a "sketch" (summary) of the data that retains certain key properties of the original dataset. These sketches can then be used to estimate various statistics without revealing sensitive information about the individual data. In this section, we first present several definitions regarding sketch vectors, discuss the Laplace mechanism for sketches and lastly explore how sketch algorithms can be used for distinct count estimation over streams.

The following definition of a *sketch vector* is due to Mir et al. [23]:

Definition 6. *Let X be a matrix of random values of dimension $m \times r$, where each entry of the matrix $X_{i,j}, 1 \leq i \leq m$, and $1 \leq j \leq r$, is drawn independently from a random stable distribution with parameter p, with p as small as possible. The **sketch vector** $sk(a)$ is defined as the dot product of matrix X^T with $\mathbf{a}$, so*

$$\text{sk}(\mathbf{a})_j = \sum_{i=1}^{m} X_{i,j} a_i = X_j \cdot \mathbf{a}$$

where X_j is a m-dimensional vector composed of the following elements: $\left(X_{1,j}, X_{2,j}, \ldots X_{m,j}\right)$.

To elucidate the definition above, we also present the following definition of a p-stable distribution, also due to Mir et al. [23]:

Definition 7. *A distribution $\mathcal{P}$ over $\mathbb{R}$ is said to be **p-stable**, if there exists $p \geq 0$ such that for any n real numbers $b_1, \ldots, b_m$ and i.i.d. variables $Y_1, \ldots, Y_m$ with distribution $\mathcal{P}$, the random variable $\sum_i b_i Y_i$ has the same distribution as the random variable $(\sum_i |b_i|^p)^{1/p} Y$, where Y is a random variable with distribution $\mathcal{P}$.*

Stable distributions are a class of probability distributions that are defined by a set of specific mathematical properties, including the fact that the mean of the distribution is stable over time. The parameter p is used to describe the degree of stability of the mean of the distribution. A distribution with a high value of p will have a more stable mean, while a distribution with a low value of p will have a less stable mean.

Updating the sketch vector Starting from the null vector, for each update encountered in the data stream, the sketch vector is updated as follows:

$$\forall j \in [r] : \text{sk}(\mathbf{a})_j \leftarrow \text{sk}(\mathbf{a})_j + d_k X_{i,j}$$

Here, the $X_{i,j}$ are values drawn from a p-stable distribution parameterized by r_1, r_2, p where r_1, r_2 are random numbers obtained from a pseudorandom generator. The p-stable distribution is then defined as:

$$stable(1/2 + \theta, r_2, p) = \frac{\sin p\theta}{\cos^{1/p}\theta} \left(\frac{\cos(\theta(1-p))}{-\ln r_2} \right)^{\frac{1-p}{p}}$$

3.2.1 Laplace mechanism for sketches

Sensitivity of a sketch Let $\mathbf{a}$ and $\mathbf{a}'$ be internal states of a sketch algorithm corresponding to two adjacent data streams S and S' (see definition 3). Mir et al. [23] demonstrate that for any S and S', the following inequality can be obtained:

$$\left\| X_j \cdot \mathbf{a} - X_j \cdot \mathbf{a}' \right\|_1 \leq \left| X_{i,j} a_i - X_{i,j} a'_i + X_{k,j} a_k - X_{k,j} a'_k \right| \leq 2 \cdot Z \left\| X_j \right\|_\infty$$

This inequality demonstrates that the global sensitivity of a sketch $\text{sk}(\mathbf{a})_j$ is:

$$GS_j = 2 \cdot Z \left\| X_j \right\|_\infty$$

We need to add Laplacian noise based on this sensitivity value in order to have a differentially private description of the algorithm's internal state at any point. The Laplacian distribution used will have mean 0 and scaling factor of GS_j/ε, where ε is the privacy parameter.

3.2.2 Distinct Count Estimation using sketch algorithms

We now show that it is possible to count the number of occurrences of elements in a stream using a sketch algorithm. Mathematically, the distinct count of a stream is defined as:

$$D^{(t)} = \left\{ i \;\middle|\; a_i^{(t)} \neq 0 \right\}$$

Now, recall that the L_p norm of a vector $\mathbf{x}$ is defined as $\|\mathbf{x}\|_p := (\Sigma_i |a_i|^p)^{1/p}$. Then, using results from Cormode et al. [8], the distinct count can be estimated using the following expression for sufficiently small p:

$$D^{(t)} \leq \sum_i |a_i|^p \leq (1+\varepsilon) D^{(t)}$$

Using p-stable distributions, we can calculate the L_p norm to then estimate the distinct count.

Mir et al.'s algorithm involves updating the sketch vector $\text{sk}(\mathbf{a})_j$ by adding noisy samples of the data records d_t at every time step t. The algorithm then takes a median of the sketch and scales it appropriately. Mir et al. [23] provide the following bound on the distinct count estimate:

Theorem 7. *With probability* $1-(r+1)\delta$*, Algorithm 1 computes an* α'*-pan-private approximation* $\tilde{\mathcal{D}}$ *of* $D^{(t)}$ *such that*

$$(1-\varepsilon)D^{(t)} - O\left(\text{poly}\left(\log(m)\cdot(1+\varepsilon)\log\left(\frac{1}{\delta}\right)\frac{1}{\alpha'}\right)\right) \leq \tilde{\mathcal{D}} \leq (1+\varepsilon)D^{(t)} + O\left(\text{poly}\left(\log(m)\cdot(1+\varepsilon)\log\left(\frac{1}{\delta}\right)\frac{1}{\alpha'}\right)\right)$$

We refer the reader to [23] for the full proof.

3.3 Adversarially Robust Streaming Algorithms

3.3.1 Existing work

Most of the streaming algorithms we have seen so far focus on the *oblivious* setting, i.e. they assume that the entire stream is *fixed* in advance (and is just fed to the algorithm one item at a time), or that the choice of items in the stream is *independent* of the internal state and the internal randomization of the algorithm.

In this section, we focus on the setting where the items in the stream and the queries fed to the algorithm are chosen by an *adaptive adversary*. That is, every item in the stream and each query is chosen by the adversary, and is a function of the previous stream items, the previous queries and the algorithm's previous answers. This means that the items in the stream are *not* independent of the algorithm's internal state. In this setting, oblivious stream algorithms do not provide any meaningful guarantees on their utility (since the stream items are no longer independent of the algorithm's previous answers). Instead, one would like to design adversarially robust streaming algorithms that maintain (provable) accuracy against adaptive adversaries.

There is a game-theoretic statement of this adversarial streaming model due to Hassidim et al. [18]: Consider a two-player game between a randomized StreamingAlgorithm and an Adversary. Fix a function g. Then proceed in rounds, where in the i-th round:

1. The Adversary chooses an update u_i for the stream, which can depend on all previous stream updates and the StreamingAlgorithm's output
2. StreamingAlgorithm processes the new update u_i and outputs a response z_i

The Adversary's goal is to make the StreamingAlgorithm output an incorrect response z_i for some time step $i \in [1..m]$, where m is the length of the stream.

The above game-theoretic statement is helpful for understanding the challenges of making streaming algorithms adversarially robust, some of which we highlight below. Hardt & Woodruff [17] show that no randomized linear sketch algorithm can be made adversarially robust. Moreover, Stemmer [26] points out that while *deterministic* streaming algorithms are adversarially robust, many streaming algorithms *must* be *randomized*

in order for the space complexity to not be linear in the length of the stream. For example, given a stream of length m, Alon, Matias & Szegedy [1] showed that to estimate the second moment F_2 of a stream, any *deterministic* streaming algorithm must use $\Omega(m)$ space, whereas a *randomized* algorithm can use $O(\alpha^{-2}\log m)$ space for some error parameter α. Stemmer [26] remarks that the challenge posed by the above adversarial streaming model is that over time, the adversary can learn the internal randomness of the algorithm and force the algorithm to become deterministic, resulting in a space blowup. This result demonstrates that randomization for stream algorithms is necessary in order for the space complexity to be sublinear in the stream length.

3.3.2 Adversarial Robust Streaming via Differential Privacy

Most of the results discussed in the previous section are impossibility results. However, Hassidim et al. [18] demonstrate how differential privacy can be used to design new adversarially robust streaming algorithms, providing constructive results.

Let $\mathcal{X}$ be the universe from which data records x in the stream originate. Let $g : \mathcal{X} \to \mathbb{R}$ be a mapping from stream data records to some real value. For example, g may count the no. of distinct elements in the stream. Now, consider an oblivious (randomized) streaming algorithm $\mathcal{A}$ that computes g.

Hassidim et al. [18] present an algorithm RobustSketch which takes $\mathcal{A}$ and makes it adversarially robust, providing accuracy guarantees within a multiplicative error of $1 \pm \alpha$ for some error parameter α.

The construction of RobustSketch is as follows: We first initialize k copies $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of algorithm $\mathcal{A}$ with a collection R of k random strings (seeds), and feeds the stream data to all k copies of $\mathcal{A}$. Then, as each element u_i in the data stream arrives, u_i is fed to each of $\mathcal{A}_1, \ldots, \mathcal{A}_k$ as an input, yielding results $y_{i,1}, \ldots, y_{i,k}$. RobustSketch then outputs $z_i = \mathsf{PrivateMedian}(y_{i,1}, \ldots, y_{i,k})$, where PrivateMedian is a canonical ε-differentially private algorithm for computing the median (see Algorithm 9 in [10]).

Hassidim et al. claim that RobustSketch is differentially private with respect to the collection of random strings R, and that for the algorithm's outputs for each $\vec{u_i}$ are accurate within a multiplicative error of $1 \pm \alpha$ of $g(\vec{u_i})$ [18].

The intuition for the proof is thus: Let $\vec{u_i} = (u_1, \ldots, u_i)$ denote the first i elements in the stream. Let $\mathcal{A}(r, \vec{u_i})$ denote the output of algorithm $\mathcal{A}$ when it is initialized with the random seed r and executed on stream $\vec{u_i}$ after the arrival of the i-th stream element. Then, consider the following function:

$$f_{\vec{u_i}}(r) := \mathbf{1}\left\{(1-\alpha)g(\vec{u_i}) \leq \mathcal{A}(r, \vec{u_i}) \leq (1+\alpha)g(\vec{u_i})\right\}$$

Here, $f_{\vec{u_i}}$ is the indicator function which is 1 if $\mathcal{A}(r, \vec{u_i})$ is approximately equal to $g(\vec{u_i})$ within a multiplicative error of $1 \pm \alpha$, and 0 otherwise.

Note that z_i is the median of $y_{i,1}, \ldots, y_{i,k}$, which is computed in a differentially private manner with respect to the collection R of random strings. Moreover, the adversary chooses the stream element u_i s a function of RobustSketch's previous outputs $z_{i-1}, z_{i-2}, \ldots$. Now, since the post-processing of a differentially-private computation preserves privacy (Theorem 4), it follows that the updates in the stream $\vec{u_i}$ are the result of a differentially private computation, where the privacy is with respect to R. It follows that $f_{\vec{u_i}}$ is also the result of a differentially-private computation on R.

Then, by the generalization properties of differential privacy (Theorem 5), it follows that:

$$\frac{1}{k}\sum_{j=1}^{k} f_{\vec{u_i}}(r_j) \approx \mathbb{E}_r[f_{\vec{u_i}}(r)]$$

Let us consider the LHS and RHS of this expression separately. As Stemmer [26] points out, $\mathbb{E}_r[f_{\vec{u_i}}(r)]$ is the success probability of the randomized oblivious algorithm $\mathcal{A}$ in the classical setting, where the random seed r is independent of the stream $\vec{u_i}$. (Here, the success probability refers to the probability of $\mathcal{A}$ returning some value $g(\vec{u_i})$ which is accurate to within a multiplicative error of $1 \pm \alpha$.) This means that $\mathbb{E}_r[f_{\vec{u_i}}(r)] \approx 1$.

Now, the LHS $\frac{1}{k}\sum_{j=1}^{k} f_{\vec{u_i}}(r_j)$ represents the fraction of $y_{i,1}, \ldots, y_{i,k}$ which are accurate to $g(\vec{u_i})$ within the aforementioned

multiplicative error. This means that:

$$\frac{1}{k}\sum_{j=1}^{k}\mathbf{1}\left\{y_{i,j} \text{ is accurate w.r.t } g(\vec{u_i}) \text{ within a multiplicative error of } 1\pm\alpha\right\} \approx \frac{1}{k}\sum_{j=1}^{k} f_{\vec{u_i}}(r_j)$$
$$\approx \mathbb{E}_r[f_{\vec{u_i}}(r)]$$
$$\approx 1$$

Since most of the $y_{i,j}$'s are at most a factor of $1\pm\alpha$ away from $g(\vec{u_i})$, it follows that the (approximate) median of the $y_{i,j}$'s also lies within the same range and is also accurate within the aforementioned multiplicative error. This establishes the accuracy of RobustSketch.

Hassidim et al.'s main contributions to the adversarial streaming litearture are to demonstrate how differential privacy can be used to hide a streaming algorithm's internal randomness from the adversary, and how the generalization properties of DP allow for accuracy guarantees.

3.4 Continual observation

Now, consider scenarios in which we need to continuously produce some output based on the data stream. Consider an input stream over $\{0,1\}$, where 1 indicates the presence of some event of interest, and for each time $t = 1,\ldots,T$, the algorithm outputs the no. of 1s seen in the stream for the first t time steps.

Let S, S' be finite stream prefixes of symbols from the universe of input symbols $\mathcal{X}$. We define a notation of adjacency for stream prefixes:

Definition 8. *S is $\mathcal{X}$-**adjacent** to S' if and only if there exist $a,b\in\mathcal{X}$ such that if we change some instances of a in S to instances in b, we get S' as a result.*
Writing $Adj(S,S')$ to denote the predicate that S and S' are adjacent stream prefixes, we have that:

$$Adj(S,S') \iff \exists\ a,b\in\mathcal{X},\ \exists R\subseteq[1..|S|]\ s.t.\ S|_{R:a\to b} = S'$$

where R is a subset of indices in the stream prefix S, and $S|_{R:a\to b}$ denotes replacing all occurrences of a in the set of indices R with b.

Event-level privacy corresponds to the case when $|R|\leq 1$. User-level privacy corresponds to any generic R where $|R|$ is not constrained.

3.4.1 Binary Mechanism (Chan et al.)

One common statistic computed by streaming algorithms is the count of certain items in the stream. For this section, we will follow the terminology used by Chan et al. [5] and define *p-sums* to be a partial sum of consecutive items in a data stream.

Chan et al. describe a mechanism for computing *noisy* p-sums that ensure differential privacy. The name "Binary Mechanism" is due to the mechanism's use of the binary representation of the stream length T.

Suppose $T\in\mathbb{N}$ is a power of 2. Then, construct a complete binary tree with T leaves, where each of the T leaves is labelled 1 through T from left to right. (Here, each leaf represents a different time step.)

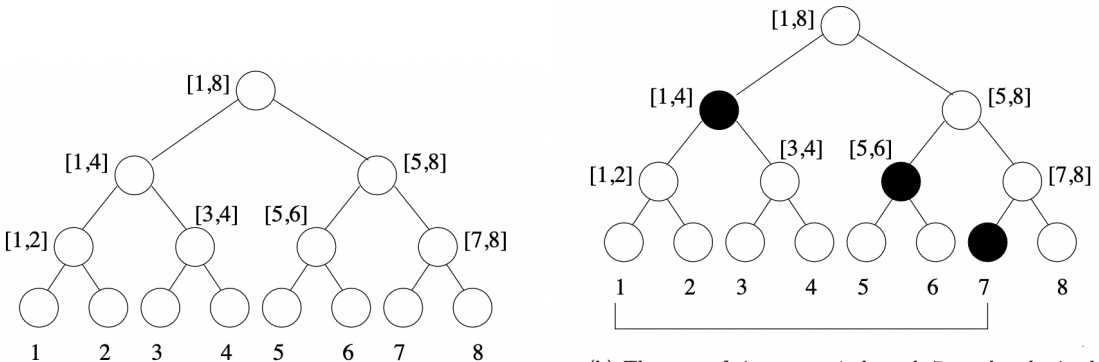
Then, each successive parent is labelled with an (integer) interval that represents the union of its children's intervals. For example, given a parent node with children labelled 1 and 2 respectively, the parent node would be labelled with the interval $[1,2]$. (See figure 1)

The aim of this mechanism is to release a noisy-count for each interval $[s,t]$, i.e. a noisy count for the no. of 1s from time steps $s, s+1,\ldots,t$. To do this, we utilize the binary representation of t as follows:

Given a number $t\in\mathbb{N}$, let $\mathsf{Bin}_i(t)\in\{0,1\}$ denote the i-th digit in the binary representation of t, where $\mathsf{Bin}_0(t)$ denotes the least significant bit of t. Note that $t = \sum_i \mathsf{Bin}_i(t)\cdot 2^i$. Then, it follows that if the i-th binary digit $\mathsf{Bin}_i(t) = 1$, then there exists a p-sum consisting of 2^i items.

The intuition of Chan et al's Binary Mechanism is thus: instead of releasing the counts for a particular time step t, the mechanism releases a sequence of noisy p-sums, from which one can estimate the count at time t.

To do this, the mechanism produces a p-sum corresponding to each node in the labelled binary tree, with random noise drawn from $\mathrm{Lap}\left(\frac{\log(T)}{\varepsilon}\right)$. Then, to recover the sum of the no. of 1s in the stream between time steps 1 and T, it suffices for one to find a set of at most $\log(T)$ nodes corresponding to disjoint sub-intervals of



(a) Every node in the binary interval tree corresponds to a **p-sum**.

(b) The sum of time steps 1 through 7 can be obtained by adding the **p-sums** corresponding to the black nodes.

Figure 1: Binary tree used in Chan et al.'s Binary Mechanism [5]

$[1..T]$ (where the union of the corresponding intervals is $[1..T]$), and compute the sum of the associated noisy p-sums.

We now discuss the intuition for how this mechanism ensures that the count is differentially private. Note that each time step $t \in [1..T]$ appears in $O(\log T)$ p-sums, and any sub-interval of $[1..T]$ can be represented with $O(\log T)$ nodes for which the union of the corresponding intervals is $[1..T]$. Moreover, since each of the $O(\log T)$ p-sums is subject to (independent) Laplace noise $\text{Lap}\left(\frac{\log T}{\varepsilon}\right)$, it follows that at time t, the error of the mechanism is the sum of $O(\log T)$ i.i.d. Laplace distributions $\text{Lap}\left(\frac{\log T}{\varepsilon}\right)$. This ensures that each intermediate p-sum (the algorithm's internal state) as well as the algorithm's output remains differentially private.

Notably, this mechanism involves adding noise to the intermediate p-sums *prior* to computing the overall count, rather than after counting (as one would typically expect). As noted in [10], thanks to the commutativity of addition, this design does not affect the mechanism's outputs.

However, as Roth points out in [10], this mechanism does not satisfy pan-privacy against one single intrusion. Suppose an intruder were to see the algorithm's internal state and the values of the data stream from time a to t. Moreover, suppose the intruder wishes to learn x_{t+1}, and they have access to the count emitted by the algorithm after some time interval $I = [a, b]$ where $a \leq t < t+1 \leq b$. Then, the intruder could just take the count emitted by the algorithm at time b and subtract $x_a, x_{a+1}, \ldots, x_t$. This would allow the intruder to learn the value of the count at time b after it has been subjected to some Laplacian noise. Then, subtracting the contribution of elements from time $[a, b]$ allows the intruder to learn x_{t+1}. This vulnerability demonstrates how the algorithm does not satisfy pan-privacy.

The detailed proof of this mechanism's utility and privacy involves various lemmas regarding the summation of i.i.d. Laplace random variables and their moment generating functions. We refer the reader to [5] for the full proof.

Chan et al. generalize the Binary Mechanism to support multi-dimensional range queries, for example SQL queries over relational databases (evolving with time) of the form:
`SELECT COUNT(*) WHERE age > 50 AND salary > 10K AND time_added <= t`

The idea behind this generalization involves representing the multi-dimensional query into the Cartesian product of 1-dimensional intervals, applying the Binary Mechanism above to compute a noisy p-sum specific to *each* dimension and aggregating the one-dimensional p-sums.

3.4.2 Cascading Buffers Counter (Dwork)

Dwork [9] discusses another approach to obtaining the count of stream elements, which involves the use of a *counter* primitive.

One naive approach to creating a counter is as follows: given data record $x_t \in \{0, 1\}$ at time t, increment the output (total count) by $x_t + \text{Lap}(1/\varepsilon)$. It is clear that this approach satisfies event-level pan privacy, by virtue of the fact that the algorithm does not remember the value of x_t in its internal state.

However, Dwork points out that for sufficiently large t, the total amount of noise added to the output via this naive approach is $O(\sqrt{t}/\varepsilon)$. This amount of noise can be problematic for *sparse* data streams where the no. of 1s is less than $\sqrt{t}$, since the output count will be dominated by the accumulated noise. Thus, the naive approach leaves much to be desired in terms of its accuracy.

To handle sparse streams, Dwork proposes a *cascading buffer counter* which achieves only poly-logarithmic error with respect to the length of the stream prefix. The key insight of this algorithm is the use of d internal *buffers* $b_1, \ldots, b_d$ which are periodically *flushed* (reset to 0) once some buffer threshold l is reached. That is, each buffer maintains a privacy-preserving *approximation* of the no. of 1s in the stream since the last flush.

Each buffer b_i also has an associated *accumulator* a_i that keeps track of the *exact* no. of 1s in the stream since the last flush. When a buffer b_i is flushed, the algorithm updates the output count by $a_i + \text{Lap}(1/\varepsilon)$ and subsequently re-initializes a_i with noise $\text{Lap}(1/\varepsilon)$. (Here, the Laplacian noise added to each accumulator is independent of other accumulators.)

The use of buffers and accumulators reduce the frequency at which the algorithm updates its output count, preventing the algorithm's update frequency from disclosing the density of 1s in the stream. Moreover, thanks to the Laplacian noise added to the accumulator, the magnitude of each update to the output count is also made private. (Announced intrusions can also be dealt with by flushing the accumulators and buffers after each intrusion.)

Moreover, the d levels of buffers control the granularity of updates to the output count, as the algorithm recursively increments each layer of buffers and keeps track of the no. of updates u_i for each buffer b_i. When u_i exceeds some threshold (i.e. one buffer has been updated excessively since the last flush), buffer b_i is also flushed. Intuitively, this design ensures that the algorithm can also maintain privacy and accuracy for dense streams with a high no. of 1s.

Dwork demonstrates how because each of the d accumulators is initialized with independent noise from $\text{Lap}(1/\varepsilon)$, and because a stream data record x_t is added to the first buffer with similar Laplacian noise, each accumulator maintains $d\varepsilon$-differential privacy. Thus, the internal state of the algorithm is $((d+1)\varepsilon)$-differentially event-level pan private. Moreover, since the output counter is only incremented by the accumulator value when a buffer is flushed and the d accumulators are reset with noise $\text{Lap}(1/\varepsilon)$ after every buffer flush, each update to the output count incurs a privacy loss $d\varepsilon$. The cascading buffer counter thus achieves $((2d + 1)\varepsilon)$-differential privacy and is event-level pan-private.

Dwork subsequently demonstrates this algorithm's accuracy by bounding the no. of updates to the output counter and the discrepancy between each buffer's approximation of the count with the true no. of 1s in the stream. Furthermore, Dwork demonstrates how the event-level private cascading buffer counter can be used to transform the density estimation algorithm discussed in section into a user-level pan-private density estimation for continuous observation purposes. We refer the reader to [9] for a detailed discussion of this process.

4 Applications & Implementations

4.1 Airavat (Differential Privacy for MapReduce)

4.1.1 MapReduce

MapReduce is a distributed computation framework that has become popular in recent years for big data applications. In the MapReduce framework, input files from a distributed file system are split into chunks. Each chunk is assigned to a *mapper* that reads the data, performs some computation, and outputs a list of (key, value) pairs. Then, *reducers* combine the value belonging to each distinct key [25]. In the literature, the two functions handling these two processes are respectively called map and reduce. These two functions can be customized to perform a wide variety of tasks over large datasets, such as counting the frequency of items or applying Boolean predicates. The MapReduce framework ensures that the execution of mappers and reducers is fault-tolerant, with their execution scheduled in parallel across nodes within a distributed system.

4.1.2 Architecture of Airavat

Airavat is a system that allows for differential privacy in MapReduce computations. Airavat uses the MapReduce framework with Apache Hadoop's distributed file system (HDFS) and the Java Virtual Machine (JVM), where SELinux is the underlying operating system. Airavat uses the JVM's Mandatory Access Control mechanisms (MAC) to enforce access privileges to the file system.

To use Airavat, a *data provider* specifies the privacy budget and desired privacy parameter ε for the dataset, and a *comptuation provider* (the user performing a query) writes code according to the MapReduce paradigm. Airavat assumes that the adversary is a malicious computation provider who has full control over the code for the mapper supplied to Airavat's MapReduce mechanism.

4.1.3 How Airavat enforces differential privacy

A malicious adversary can potentially use keys to extract information about the input. For this purpose, Airavat never outputs any keys produced by untrusted mappers. Instead, the computation provider submits a list of keys and Airavat returns noisy values associated with those keys. To return noisy values, the reducers add exponentially distributed noise to the output of the computation, with the amount of noise determined by the sensitivity of the computation.

The computation provider must also specify the range of output values (M_{min}, M_{max}) that the mapper can produce. Airavat uses this range to estimate the sensitivty of the function and to determine the appropriate amount of noise that should be added such that the perturbed value remains within the desired range. If the mapper outputs a value outside the range, the range enforcer replaces it with a value inside the range. The creators of Airavat note that this design "prioritizes privacy over accuracy" [25], and that accurate results from MapReduce computations using Airavat can only be obtained if the computation provider knows the sensitivity of their computation beforehand [25].

Airavat is also equipped with a *range enforcer*, which checks that the value within each (key, value) pair produced by the mapper lies within the declared range. This ensures that the sensitivity of the computation doesn't exceed the estimated sensitivity. This also means that if the user doesn't provide a good range of outputs the out put can end up being too noise and non-interpretable.

Airavat also defines specific behavior for the mappers. Mappers have to be independent, and only a single input record is allowed to affect the (key, value) pairs output by the mapper. The mapper is also prohibited from storing the (key, value) pairs produced from an input record for later use. Moreover, multiple mappers $M_1, \ldots, M_j$ can be composed, followed by a final reducer R_j which adds noise to the final computation result. To attenuate the reduction in the privacy budget, the computation provider can specify the max no. of keys n that each mapper can output, such that Airavat only passes n (key, value) pairs between successive mappers.

Airavat can also ensure group privacy, i.e. privacy for a user which has multiple entries or a group of users. Airavat attaches group IDs to (key, value) pairs to track the dispersal of information from each input privacy group through intermediate keys to the output.

One limitation of Airavat is that it cannot confine the computations that are performed by untrusted code that the computation provider submits to the mapper. Specifically, privacy cannot be guaranteed for untrusted mapper code which perform computations that can disclose the value of intermediate keys (since such keys originate from the dataset) [25]. Instead, for MapReduce computations that require keys to be output, the computation provider is required to declare keys in advance.

4.2 Programming Frameworks for Differentially Private Stream Queries

4.2.1 Streaming PINQ

Proposed in 2010 by McSherry [22], *PINQ* (Privacy Integrated Queries) is a programming platform which supports differentially-private database queries in the *C#* language. Waye [30] extended PINQ to support differentially private queries over streams, providing extensible interfaces for both data providers and data analysts.

Notably, Waye's implementation of *Streaming PINQ* distinguishes between user-level and event-level privacy as per Dwork et al.'s definitions (discussed in section 3.1). An abstract base class PINQStreamingAgent is implemented, with subclasses PINQUserLevelAgent and PINQEventLevelAgent that enforce user-level and event-level privacy respectively. These two agent subclasses determine whether a streaming algorithm is able to receive events from the stream, and control the resultant loss in the privacy budget. To ensure ε-event level differential privacy, PINQEventLevelAgent ensures that at most ε is learned for each event, and after the current streaming algorithm finishes, PINQEventLevelAgent allows subsequent streaming algorithms run with the initial privacy budget – note that this functionality is disallowed by PINQUserLevelAgent since user-level privacy is defined over the entire stream.

However, Streaming PINQ does not dynamically check the code of a streaming algorithm to verify the algorithms' advertised privacy guarantees. The functionality offered by Streaming PINQ is mainly focused on keeping track of the loss in the privacy budget and controlling streaming algorithms' access to the stream.

Additionally, Streaming PINQ accepts arbitrary SQL-esque logical predicates written in C#, and adversaries could take advantage of this vulnerability to discover information about the private data. For example, Waye discusses an example where an adversary could execute a very long loop when an element in the stream satisfies a predicate [30]. It is also worth noting that while Waye implements several DP algorithms in

Streaming PINQ (eg. randomized response count), the timestamps at which events are processed are exposed, which may not be ideal for time-sensitive data such as stock trades.

4.2.2 Related work

Haeberlen, Pierce and Narayan developed a functional language *Fuzz* equipped with a type system that guarantees differential privacy [15]. At a high level, this is achieved by via a special type `db` in Fuzz that represents a static database. Through static analysis, the Fuzz typechecker ensures that any database queries *cannot* return type `db`, and that queries must add noise to any values returned that are primitive data types (eg. integers, strings). However, Fuzz does not support time-evolving data or data streams, and it remains unclear how Fuzz's type system could be extended to support data streams [30].

The MapReduce-based system Airavat (section 4.1.2) for differentially private cloud computing is based off the Hadoop framework, which is most typically used for batch processing instead of streaming. Although there exists a *Hadoop Streaming* utility within the Hadoop framework, this utility should not be confused with computation frameworks such as Spark Streaming that process unbounded data streams. (As discussed in [2], the "streams" in Hadoop Streaming refer to standard Unix streams such as `stdin` and `stdout`.)

4.2.3 Attacks on Differential Privacy

Haeberlen, Pierce & Narayan showed that PINQ and Airavat are both susceptible to *timing attacks* [15]. In this setting, an adversary submits a query that pauses for a long time if a record satisfying a predicate exists. Since the distribution of query execution times change drastically when a record of interest is encountered, this is a violation of differential privacy. The creators of Airavat are cognizant of this vulnerability but state that only a small no. of bits an be transmitted to the adversary as a result of this limitation [25].

PINQ is also susceptible to *privacy budget attacks*. In this setting, an adversary exploits the query processor's decision whether to disclose the result of a computation to identify whether the query contains private data [15]. This attack exploits the dynamic analysis performed by PINQ that determines whether a query results in any remaining privacy budget. In this setting, an adversary could compute the difference in the privacy budget before and after running a query to determine whether a record satisfying a predicate is returned by the query. Notably, Airavat is not susceptible to this attack as it deducts the appropriate amount from the privacy budget prior to executing the query [15].

4.3 Private Hadamard Count Mean Sketch (Apple)

Another example of how differentially private stream algorithms have been used in industry can be found at Apple, who use the *Private Hadamard Count Mean Sketch* algorithm to compute counts in user data streams. (One such example is finding the most commonly used emoji among iOS users.) Apple's construction is based on a variant of the *Count-Min sketch* matrix, which is a probabilistic data structure defined below [7]:

Definition 9. *A **Count-Min sketch matrix** CM is a $w \times d$ matrix where each entry is initialized to 0. Moreover, d pairwise-independent hash functions $h_1, \ldots, h_d$, are chosen from a universal hash family. (The domain of each of these hash functions is the domain of the stream data, and the codomain is $[1..w]$.)*
Then, for each update (i_t, d_t) in the stream, we update the matrix as follows:

$$\text{for all } j \in [1..d], \text{ set } CM[j, h_j(i_t)] \leftarrow CM[j, h_j(i_t)] + d_t$$

(Note that each hash function h_i indexes into a different row of the matrix CM.)

In Apple's construction, for each data record arriving in the stream, a hash function $h : \mathcal{D} \to [1..m]$ is selected uniformly at random from a family of 3-wise independent hash functions $\{h_1, \ldots, h_d\}$. [2] This hashed function is used to encode the data record as a m-dimensional one-hot encoded vector. Then, to ensure differential privacy, each bit in the hashed vector representation of the stream data record is flipped with probability $1/(1 + e^{\varepsilon/2})$, where ε is the privacy parameter.

The privatized vector $\mathbf{v}$ and the index of the (randomly) selected hash function is then sent to a server, which updates a Count-Min sketch matrix M of size $d \times m$ according to the definition above. (Here, the no. of rows d in the matrix M is the same as the no. of hash functions, and the no. of columns m is equal to the size of the privatised vector transmitted to the server.) Then, to compute the frequency (count) of a data record x, for each $j \in [1..d]$, the algorithm computes the mean of all matrix entries $M[j, h_j(x)]$.

[2] See section 2.5 for a definition on k-wise independent hash functions.

Note that for large d and m, the cost of transmitting the privatized vector to the server to construct the sketch matrix may be prohibitively high. To circumvent this issue without sacrificing accuracy, Apple utilises the Hadamard transform, which is defined below:

Definition 10. *For $l \in \mathbb{N}$ where l is a power of 2, the **Hadamard transform** matrix is a $l \times l$ matrix H_l defined recursively, where:*

$$H_1 = [1]$$
$$H_l = \begin{bmatrix} H_{l/2} & H_{l/2} \\ H_{l/2} & -H_{l/2} \end{bmatrix}$$

The privatised vector $\mathbf{v}$ is left-multiplied by the Hadamard transform matrix H_m to obtain $\mathbf{v}' = H_m\mathbf{v}$, where $\mathbf{v}' \in \{\pm 1\}^m$. Then, one component of the vector $\mathbf{v}'$ is chosen uniformly at random, and this bit is flipped with probability $1/(1 + e^{\varepsilon/2})$. This single bit and its corresponding index (along with the index of the randomly chosen hash function) is then transmitted to the server. The server then populates the Count-Min sketch matrix M as before, and performs an inverse Hadamard transform to obtain the original basis for M.

Surprisingly, despite transmitting only one single privatised bit from the client to the server, accurate counts can still be computed using Apple's algorithm. We refer the reader to [28] for the detailed proof of this algorithm's utility and differential privacy, which rely on statistical properties of the estimators derived from the entries of the resultant matrix M.

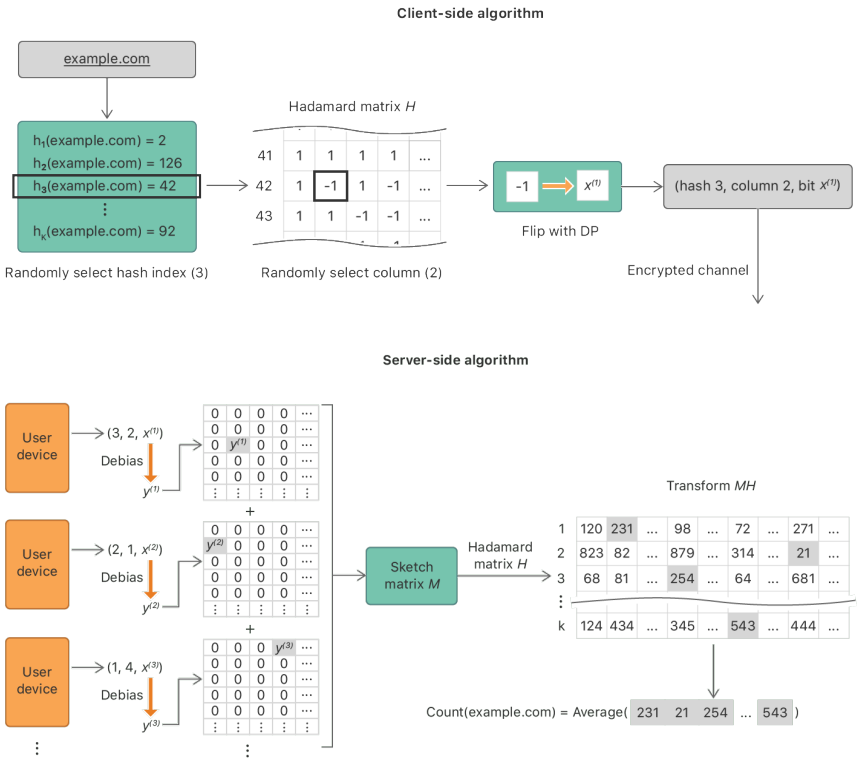


Figure 2: Apple's Private Hadamard Count Mean Sketch algorithm [29]

To identify users' most commonly used emojis, Apple uses a universal hash family consisting of $d = 65,536$ different 3-wise independent hash functions, with the privatised vectors having size $m = 1024$, and the privacy parameter set to $\varepsilon = 4$ [29]. Personal identifiable information such as IP addresses and user identifiers are stripped from the stream data, and the communication between the client (user device) and the server is encrypted using TLS [28].

Apple's deployment of differential privacy is markedly different from the previous programming frameworks explored in this survey. Note that while Airavat, Streaming PINQ and Fuzz are generic-purpose programming frameworks that accept user-defined queries and aim to enforce differential privacy, Apple's use of Count Mean Sketch is specifically used for computing aggregate statistics over user data streams.

Tang et al. reverse-engineered Apple's aforementioned DP mechanism for macOS and found that each user's

privacy budget is renewed every day. We refer the reader to their work [27] for a critique of the implementation of Apple's DP mechanisms.

5 Conclusion and Open Problems

We have examined various problems related to differential privacy in stream processing settings, ranging from pan-privacy to continual observation. We also examine sketch and adversarial streaming algorithms, as well as various programming frameworks for DP in stream processing. In this concluding section, we begin by discussing a few open problems in the literature on DP for stream processing.

5.1 Open problems

Open problems in continual observation
Chan et al. [5] discuss how the aforementioned counting mechanisms can be used for recommendation systems that continuously generate the top k suggestions for visitors to websites. Moreover, Bolot et al. [3] extend Dwork's model of continual observation and propose algorithms for estimating statistics over sliding windows and time-decayed streams where more recent items are prioritsed over older items. This work is related to that of Chen et al. [6], who propose algorithms that can support *multiple* streaming processing tasks (eg. event monitoring and sliding windows) over different resolutions of the same data stream. Furthermore, Cardoso & Rogers [4] generalize Dwork et al. and Chan et al's continual observation models (discussed in section 3.4) to settings where each event in the data stream is a *subset* of an unknown universe of items. Additionally, Jain et al. [19] focus on online convex programming algorithms and discuss methods for transforming them into DP variants while achieving sublinear regret.

However, many of these algorithms ensure only *event-level* differential privacy, i.e. protecting the privacy of one single event (eg. concealing one instance of user's interaction with a website). One could argue that sensitive information is disclosed from *multiple* events that occur in succession. To this end, Kellaris et al. [20] define a new notion of *w-event privacy*, which concerns the privacy of contiguous *event sequences* that occur in w successive periods. Kellaris et al. implement variants of canonical DP algorithms that ensure w-event privacy.

Achieving *user-level* differential privacy for continual observation problems remains an open problem due to the stronger privacy guarantees needed for user-level privacy. In fact, Dwork et al. [12] demonstrate that for stream processing algorithms that aggregate the advice of multiple experts, it is *impossible* to obtain any non-trivial result while preserving user level privacy.

Clustering problems
Har-Peled and Mazumdar [16] define the notion of *coresets* (weighted sets of points in $\mathbb{R}^d$), which can be used to derive streaming equivalents of clustering algorithms such as k-means clustering. Here, the data stream consists of points in $\mathbb{R}^d$ which are successively added to a subset $P \subseteq \mathbb{R}^d$. Feldman et al. [14] present differentially-private algorithms for computing coresets for k-means and k-median clustering in streaming settings.

5.2 Addendum

Lastly, we conclude this survey with a few thoughts on sketching algorithms and pan-privacy for MapReduce.

Sketch algorithms Non-sketching pan private algorithms generally use sampling and adaptive randomized response techniques. On the other hand, sketching algorithms maintain random "sketches" of the data, i.e. linear projections of data along random directions. Dwork et al. [13] show that multi-party sketch algorithms would not be suitable for building pan-private algorithms. This is because in a multi-party system, the algorithm's correctness requirement would cause some participant to disclose information that would allow one to guess the correct input with non-trivial probability, thus violating differential privacy. However, Mir et al. [23] show that there is a method to use single-party sketch algorithms to build pan-private algorithms. This result demonstrates the utility of sketch algorithms for private computation. Traditional sketch algorithms already have randomness built into them, so they can be made pan-private using the Laplace mechanism and other differentially-private techniques.

Pan-Privacy for MapReduce We saw in section 4.1.2 that Roy et al.'s Airavat framework allowed for differentially private queries to be implemented using the MapReduce framework [25]. Airavat introduces the use of Mandatory Access Control mechanisms (MACs) for ensuring private distributed computation. We saw how

MACs in tandem with Airavat's various enforcing mechanisms prevent the internal state of algorithms from being disclosed. This is in some sense similar to what pan-privacy is trying to achieve in an algorithmic manner. Perhaps one day, pan-private algorithms could be integrated into the MapReduce framework to create an Airavat-esque platform without the need for MACs.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *Journal of Computer and System Sciences* 58.1 (1999), pp. 137–147. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1545. URL: https://www.sciencedirect.com/science/article/pii/S0022000097915452.

[2] Benjamin Bengfort and Jenny Kim. *Data Analytics with Hadoop*. O'Reilly Media, Inc., 2016, p. 43.

[3] Jean Bolot et al. "Private Decayed Sum Estimation under Continual Observation". In: *CoRR* abs/1108.6123 (2011). arXiv: 1108.6123. URL: http://arxiv.org/abs/1108.6123.

[4] Adrian Rivera Cardoso and Ryan Rogers. "Differentially Private Histograms under Continual Observation: Streaming Selection into the Unknown". In: *CoRR* abs/2103.16787 (2021). arXiv: 2103.16787. URL: https://arxiv.org/abs/2103.16787.

[5] Hubert Chan, Elaine Shi, and Dawn Song. "Private and Continual Release of Statistics". In: *ACM Transactions on Information and System Security* (2010).

[6] Yan Chen et al. "PeGaSus: Data-Adaptive Differentially Private Stream Processing". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, 1375–1388. ISBN: 9781450349468. DOI: 10.1145/3133956.3134102. URL: https://doi.org/10.1145/3133956.3134102.

[7] Graham Cormode. *Count-Min Sketch*. http://dimacs.rutgers.edu/~graham/pubs/papers/cmencyc.pdf.

[8] Graham Cormode et al. "Comparing Data Streams Using Hamming Norms (How to Zero In)". In: *Proceedings of the 28th International Conference on Very Large Data Bases*. VLDB '02. Hong Kong: VLDB Endowment, 2002, 335–345.

[9] Cynthia Dwork. "Differential Privacy in New Settings". In: *Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 2010. URL: https://www.microsoft.com/en-us/research/publication/differential-privacy-in-new-settings/.

[10] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Foundational Trends in Theoretical Computer Science* 9.3–4 (2014), 211–407. ISSN: 1551-305X. DOI: 10.1561/0400000042. URL: https://doi.org/10.1561/0400000042.

[11] Cynthia Dwork et al. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Theory of Cryptography*. Ed. by Shai Halevi and Tal Rabin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 265–284. ISBN: 978-3-540-32732-5.

[12] Cynthia Dwork et al. "Differential Privacy under Continual Observation". In: *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*. STOC '10. Cambridge, Massachusetts, USA: Association for Computing Machinery, 2010, 715–724. ISBN: 9781450300506. DOI: 10.1145/1806689.1806787. URL: https://doi.org/10.1145/1806689.1806787.

[13] Cynthia Dwork et al. "Pan-Private Streaming Algorithms". In: *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*. Ed. by Andrew Chi-Chih Yao. Tsinghua University Press, 2010, pp. 66–80. URL: http://conference.iiis.tsinghua.edu.cn/ICS2010/content/papers/6.html.

[14] Dan Feldman et al. "Private Coresets". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC '09. Bethesda, MD, USA: Association for Computing Machinery, 2009, 361–370. ISBN: 9781605585062. DOI: 10.1145/1536414.1536465. URL: https://doi.org/10.1145/1536414.1536465.

[15] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. "Differential Privacy under Fire". In: *Proceedings of the 20th USENIX Conference on Security*. SEC'11. San Francisco, CA: USENIX Association, 2011, p. 33.

[16] Sariel Har-Peled and Soham Mazumdar. "Coresets for k-Means and k-Median Clustering and their Applications". In: *CoRR* abs/1810.12826 (2018). arXiv: 1810.12826. URL: http://arxiv.org/abs/1810.12826.

[17] Moritz Hardt and David P. Woodruff. "How Robust are Linear Sketches to Adaptive Inputs?" In: *CoRR* abs/1211.1056 (2012). arXiv: 1211.1056. URL: http://arxiv.org/abs/1211.1056.

[18] Avinatan Hassidim et al. "Adversarially Robust Streaming Algorithms via Differential Privacy". In: *CoRR* abs/2004.05975 (2020). arXiv: 2004.05975. URL: https://arxiv.org/abs/2004.05975.

[19] Prateek Jain, Pravesh Kothari, and Abhradeep Thakurta. "Differentially Private Online Learning". In: *CoRR* abs/1109.0105 (2011). arXiv: 1109.0105. URL: http://arxiv.org/abs/1109.0105.

[20] Georgios Kellaris et al. "Differentially Private Event Sequences over Infinite Streams". In: *Proc. VLDB Endow.* 7.12 (2014), 1155–1166. ISSN: 2150-8097. DOI: 10.14778/2732977.2732989. URL: https://doi.org/10.14778/2732977.2732989.

[21] Shachar Lovett. *Constructions of pairwise independent variables*. https://cseweb.ucsd.edu/classes/fa13/cse290-b/notes-lecture2.pdf. 2013.

[22] Frank D. McSherry. "Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis". In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. SIGMOD '09. Providence, Rhode Island, USA: Association for Computing Machinery, 2009, 19–30. ISBN: 9781605585512. DOI: 10.1145/1559845.1559850. URL: https://doi.org/10.1145/1559845.1559850.

[23] Darakhshan J. Mir et al. "Pan-private Algorithms: When Memory Does Not Help". In: *CoRR* abs/1009.1544 (2010). arXiv: 1009.1544. URL: http://arxiv.org/abs/1009.1544.

[24] Aleksander Madry. *CS-621 (Theory Gems) Lecture 17: L_p norm estimation problem*. 2012.

[25] Indrajit Roy et al. "Airavat: Security and Privacy for MapReduce". In: *Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX - Advanced Computing Systems Association, 2010. URL: https://www.microsoft.com/en-us/research/publication/airavat-security-and-privacy-for-mapreduce/.

[26] Uri Stemmer. *Adversarial Streaming, Differential Privacy, and Adaptive Data Analysis*. 2021. URL: https://www.youtube.com/watch?v=Whu-6lVYFXc (visited on 12/25/2022).

[27] Jun Tang et al. "Privacy Loss in Apple's Implementation of Differential Privacy on MacOS 10.12". In: *CoRR* abs/1709.02753 (2017). arXiv: 1709.02753. URL: http://arxiv.org/abs/1709.02753.

[28] Apple Differential Privacy Team. *Learning with Privacy at Scale (article)*. https://docs-assets.developer.apple.com/ml-research/papers/learning-with-privacy-at-scale.pdf. 2017.

[29] Apple Differential Privacy Team. *Learning with Privacy at Scale (blog post)*. https://machinelearning.apple.com/research/learning-with-privacy-at-scale. 2017.

[30] Lucas Waye. "Privacy Integrated Data Stream Queries". In: *Proceedings of the 2014 International Workshop on Privacy Security in Programming*. PSP '14. Portland, Oregon, USA: Association for Computing Machinery, 2014, 19–26. ISBN: 9781450322966. DOI: 10.1145/2687148.2687150. URL: https://doi.org/10.1145/2687148.2687150.