A Smooth Binary Mechanism for Efficient Private Continual Observation

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

In privacy under continual observation we study how to release differentially private estimates based on a dataset that evolves over time. The problem of releasing private prefix sums of $x_1, x_2, x_3, \dots \in \{0,1\}$ (where the value of each x_i is to be private) is particularly well-studied, and a generalized form is used in state-of-the-art methods for private stochastic gradient descent (SGD). The seminal *binary mechanism* privately releases the first t prefix sums with noise of variance polylogarithmic in t. Recently, Henzinger et al. and Denisov et al. showed that it is possible to improve on the binary mechanism in two ways: The variance of the noise can be reduced by a (large) constant factor, and also made more even across time steps. However, their algorithms for generating the noise distribution are not as efficient as one would like in terms of computation time and (in particular) space. We address the efficiency problem by presenting a simple alternative to the binary mechanism in which 1) generating the noise takes constant average time per value, 2) the variance is reduced by a factor about 4 compared to the binary mechanism, and 3) the noise distribution at each step is *identical*. Empirically, a simple Python implementation of our approach outperforms the running time of the approach of Henzinger et al., as well as an attempt to improve their algorithm using high-performance algorithms for multiplication with Toeplitz matrices.

1 Introduction

There are many actors in society that wish to publish aggregate statistics about individuals, be it for financial or social utility. Netflix might recommend movies based on other users' preferences, and policy might be driven by information on average incomes across groups. Whatever utility one has in mind however, it should be balanced against the potential release of sensitive information. While it may seem anodyne to publish aggregate statistics about users, doing it without consideration to privacy may expose sensitive information of individuals (Dinur & Nissim, 2003). Differential privacy offers a framework for dealing with these problems in a mathematically rigorous way.

A particular setting is when statistics are updated and released continually, for example a website releasing its number of visitors over time. Studying differential privacy in this setup is referred to as *differential privacy under continual observation* (Dwork et al., 2010; Dwork & Roth, 2013). A central problem in this domain is referred to as *differentially private counting under continual observation* (Chan et al., 2011; Dwork et al., 2010), *continual counting* for short. It covers the following problem: a binary stream $x_1, x_2, x_3, \dots$ is received one element at a time such that x_t is received in round t. The objective is to continually output the number of 1s encountered up to each time step while maintaining differential privacy. The neighboring relation considered is that streams x and x' are neighboring if they are identical except for a single index i where $x_i \neq x'_i$. It suffices to study the setting in which there is a known upper bound T on the number of prefix sums to release — algorithms for the case of unbounded streams then follow by a general reduction (Chan et al., 2011).

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

Table 1: Comparison between different ρ-zCDP mechanisms for continual counting.

Mechanism $\mathcal{M}$	$\mathrm{Var}[\mathcal{M}(t)] \cdot \frac{2\rho}{\log_2^2 T}$	**Time**$/T$	**Space**	**Identically distributed**	**Streaming support**	**Matrix type**
Binary mech.	1	$O(1)$	$O(\log T)$	no	yes	sparse
Honaker	$< 1^{***}$	$O(T)$	$O(T)$	no	no	dense
Denisov et al.	$0.0973\ldots + o(1)^{**}$	$O(T)$	$O(T^2)$	no	yes	dense
Henzinger et al.	$0.0973\ldots + o(1)$	$O(T)^*$	$O(T)$	no	yes	Toeplitz
Our mech.	$0.25 + o(1)$	$O(1)$	$O(\log T)$	yes	yes	sparse

The time usage given in Henzinger et al. (2023) is as stated in the table *, but it is possible to achieve time $O(\log T)$ by implementing the matrix-vector product using FFT, at the expense of not supporting streaming. Leveraging FFT for continual observation is not a novel approach (Choquette-Choo et al., 2022). There is no explicit bound on variance in Denisov et al. (2022), but the method finds an optimal matrix decomposition so it should achieve same variance ** as Henzinger et al. (2023) up to lower order terms. Similarly, detailed analysis of the variance *** is not provided in Honaker (2015), but empirically variance is reduced by some constant (see experiments in Denisov et al. (2022)). All sparse matrices have $O(\log T)$ nonzero entries per row or column, and all dense matrices have $\Omega(T^2)$ nonzero entries — a Toeplitz matrix can be seen as intermediate in the sense of having $O(T)$ unique diagonals, allowing for efficient storage and multiplication.

Aside from the natural interpretation of continual counting as the differentially private release of user statistics over time, mechanisms for continual counting (and more generally for releasing prefix sums) are used as a subroutine in many applications. Such a mechanism is for example used in Google's privacy-preserving federated next word prediction model (McMahan & Thakurta, 2022; Kairouz et al., 2021), in non-interactive local learning (Smith et al., 2017), in stochastic convex optimization (Han et al., 2022) and in histogram estimation (Cardoso & Rogers, 2022; Chan et al., 2012; Huang et al., 2022; Upadhyay, 2019) among others.

Given the broad adoption of continual counting as a primitive, designing algorithms for continual counting that improve constants in the error while scaling well in time and space is of practical interest.

1.1 Our contributions

In this paper we introduce the *Smooth Binary Mechanism*, a differentially private algoritm for the continual counting problem that improves upon the original binary mechanism by Chan et al. (2011); Dwork et al. (2010) in several respects, formalized in Theorem 1.1 and compared to in Table 1.

Theorem 1.1 (Smooth Binary Mechanism)**.** *For any $\rho > 0$, $T > 1$, there is an efficient ρ-zCDP continual counting mechanism $\mathcal{M}$, that on receiving a binary stream of length T satisfies*

$$\mathrm{Var}[\mathcal{M}(t)] = \frac{1 + o(1)}{8\rho} \log_2(T)^2$$

where $\mathcal{M}(t)$ is the output prefix sum at time t, while only requiring $O(\log T)$ space, $O(T)$ time to output all T prefix sums, and where the error is identically distributed for all $1 \leq t \leq T$.

Our mechanism retains the scalability in time and space of the binary mechanism while offering an improvement in variance by a factor of $4 - o(1)$. Unlike existing mechanisms it has the *same* error distribution in every step, which could make downstream applications easier to analyze.

Sketch of technical ideas. Our starting point is the binary mechanism which, in a nutshell, uses a complete binary tree with $\geq T + 1$ leaves (first T leaves corresponding to $x_1, \ldots, x_T$) in which each node contains the sum of the leaves below, made private by adding random noise (e.g. from a Gaussian distribution). To estimate a prefix sum $\sum_{i=1}^{t} x_i$ we follow the path from the root to the leaf storing x_{t+1}. Each time we go to a right child the sum stored in its sibling node is added to a counter. An observation, probably folklore, is that it suffices to store sums for nodes that are *left children*, so suppose we do not store any sum in nodes that are right children. The number of terms added when computing the prefix sum is the number of 1s in the binary representation $\mathrm{bin}(t)$ of t, which encodes the path to the leaf storing x_{t+1}. The *sensitivity* of the tree with respect to x_t, i.e., the number of node counts that change by 1 if x_t changes, is the number of 0s in $\mathrm{bin}(t-1)$. Our idea is to only use leaves that have *balanced* binary representation, i.e. same number of 0s and 1s (assuming the height h of the tree is an even integer). To obtain T useful leaves we need to make the tree slightly deeper — it turns out that height h slightly more than $\log_2(T)$ suffices. This has the effect of making the sensitivity of every leaf $h/2$, and the noise in every prefix sum a sum of $h/2$ independent noise terms.

Limitations. As shown in Table 1 and Section 4, the smooth binary mechanism, while improving on the original binary mechanism, cannot achieve as low variance as matrix-based mechanisms such as Henzinger et al. (2023). However, given that the scaling of such methods can keep them from being used in practice, our variant of the binary mechanism has practical utility in large-scale settings.

1.2 Related work

All good methods for continual counting that we are aware of can be seen as instantiations of the *matrix mechanism* (Li et al., 2015). These methods perform a linear transformation of the data, given by a matrix R, then add noise according to the sensitivity of Rx, and finally obtain the sequence of estimates by another linear transformation given by a matrix L. To obtain unbiased estimates, the product LRx needs to be equal to the vector of prefix sums, that is, LR must be the lower triangular all 1s matrix.

Seminal works introducing the first variants of the binary mechanism are due to Chan, Shi, and Song (2011) and Dwork, Naor, Pitassi, and Rothblum (2010), but similar ideas were proposed independently in Hay, Rastogi, Miklau, and Suciu (2010) and Gehrke, Wang, and Xiao (2010). Honaker (2015) noticed that better estimators are possible by making use of *all* information in the tree associated with the binary tree method. Some parts of his method are incompatible with a streaming setting since they require all values $x_1, \dots, x_T$ to be known, though a streaming variant has later been devised by Denisov et al. (2022).

A number of recent papers have studied improved choices for the matrices L, R. Denisov, McMahan, Rush, Smith, and Thakurta (2022) showed that the problem of finding matrices leading to minimum largest variance on the estimates is a convex optimization problem, and that (at least for T up to 2048) it is feasible to solve it. They further show that the smallest error can be obtained with lower triangular matrices, making it possible to support streaming settings where values of x_i are revealed one at a time and a prefix sum estimate must be released immediately. To handle a larger number of time steps they consider a similar setting where a restriction to *banded* matrices makes the method scale better, empirically with good error, but no theoretical guarantees are provided.

Fichtenberger, Henzinger, and Upadhyay (2022) gave an explicit decomposition into lower triangular matrices, and analyzed its error in the ℓ_∞ metric. The matrices employed are Toeplitz (banded) matrices. Henzinger, Upadhyay, and Upadhyay (2023) analyzed the same decomposition with respect to mean squared error of the noise, and showed that it obtains the best possible error among matrix decompositions where L and R are square matrices, up to a factor $1 + o(1)$ where the $o(1)$ term vanishes when T goes to infinity.

A breakdown of how our mechanism compares to existing ones is shown in Table 1. While not achieving as small an error as the matrix mechanism of Henzinger et al. (2023), its runtime and small memory footprint allows for better scaling for longer streams. For concreteness we consider privacy under ρ-zero-Concentrated Differential Privacy (zCDP) (Bun & Steinke, 2016), but all results can be expressed in terms of other notions of differential privacy.

2 Preliminaries

Binary representation of numbers. We will use the notation $\mathrm{bin}(n)$ to refer to the binary representation of a number $n \in [2^h]$, where $h > 0$ is an integer, and let $\mathrm{bin}(n)$ be padded with leading to h digits. For example, $\mathrm{bin}(2) =$ `0b010` for a tree of height $h = 3$. When indexing such a number we let index i refer to the i^{th} least significant bit, e.g. $\mathrm{bin}(2)_1 = 1$. We will also refer to prefixes and suffixes of binary strings, and we use the convention that a prefix of a binary string includes its most significant bits.

Partial sums (p-sums). To clear up the notation we use the concept of p-sums $\Sigma[i,j]$ where $\Sigma[i,j] = \sum_{t=i}^{j} x_t$. We will furthermore use the concept of noisy p-sums

$$\widehat{\Sigma}[i,j] = \Sigma[i,j] + X[i,j],\ X[i,j] \sim \mathcal{F}$$

where $\mathcal{F}$ is a suitable distribution for the DP paradigm, e.g. Laplacian or Gaussian. For convenience we also define $\hat{x}_t = \widehat{\Sigma}[t,t]$, i.e. $\hat{x}_t$ is the single stream element with noise added.

2.1 Continual observation of bit stream

Given an integer $T > 1$ we consider a finite length binary stream $x = (x_1, x_2, \dots, x_T)$, where $x_t \in \{0,1\}$, $1 \le t \le T$, denotes the bit appearing in the stream at time t.

Definition 2.1 (Continual Counting Query)**.** Given a stream $x \in \{0,1\}^T$, the *count* for the stream is a mapping $c : \{1,\dots,T\} \to \mathbb{Z}$ such that $c(t) = \sum_{i=1}^{t} x_i$.

Definition 2.2 (Counting Mechanism)**.** A *counting mechanism* $\mathcal{M}$ takes a stream $x \in \{0,1\}^T$ and produces a (possibly random) vector $\mathcal{M}_x \in \mathbb{R}^T$ where $(\mathcal{M}_x)_t$ is a function of the first t elements of the stream. For convenience we will write $\mathcal{M}(t)$ for $(\mathcal{M}_x)_t$ when there is little chance for ambiguity.

To analyze a counting mechanism from the perspective of differential privacy, we also need a notion of neigboring streams.

Definition 2.3 (Neighboring Streams)**.** Streams $x, x' \in \{0,1\}^T$ are said to be *neighboring*, denoted $x \sim x'$, if $|\{i \mid x_i \neq x'_i\}| = 1$.

Intuitively, for a counting mechanism to be useful at a given time t, we want it to minimize $|\mathcal{M}(t) - c(t)|$. We consider unbiased mechanisms that return the true counts in expectation and we focus on minimizing $\mathrm{Var}[\mathcal{M}(t) - c(t)]$.

2.2 Differential privacy

For a mechanism to be considered differentially private, we require that the outputs for any two neighboring inputs are are indistinguishable. We will state our results in terms of ρ-zCDP:

Definition 2.4 (Concentrated Differential Privacy (zCDP) (Bun & Steinke, 2016))**.** For $\rho > 0$, a randomized algorithm $\mathcal{A} : \mathcal{X}^n \to \mathcal{Y}$ is ρ-zCDP if for any $\mathcal{D} \sim \mathcal{D}'$, $D_\alpha(\mathcal{A}(\mathcal{D}) || \mathcal{A}(\mathcal{D}')) \le \rho\alpha$ for all $\alpha > 1$, where $D_\alpha(\mathcal{A}(\mathcal{D}) || \mathcal{A}(\mathcal{D}'))$ is the α-Rényi divergence between $\mathcal{A}(\mathcal{D})$ and $\mathcal{A}(\mathcal{D}')$.

In the scenario where we are looking to release a real-valued function $f(\mathcal{D})$ taking values in $\mathbb{R}^d$, we can achieve zCDP by adding Gaussian noise calibrated to the ℓ_2-sensitivity of f.

Lemma 2.5 (Gaussian Mechanism (Bun & Steinke, 2016))**.** *Let $f : \mathcal{X}^n \to \mathbb{R}^d$ be a function with global ℓ_2-sensitivity $\Delta := \max_{\mathcal{D} \sim \mathcal{D}'} \|f(\mathcal{D}) - f(\mathcal{D}')\|_2$. For a given data set $\mathcal{D} \in \mathcal{X}^n$, the mechanism that releases $f(\mathcal{D}) + \mathcal{N}(0, \frac{\Delta^2}{2\rho})$ satisfies ρ-zCDP.*

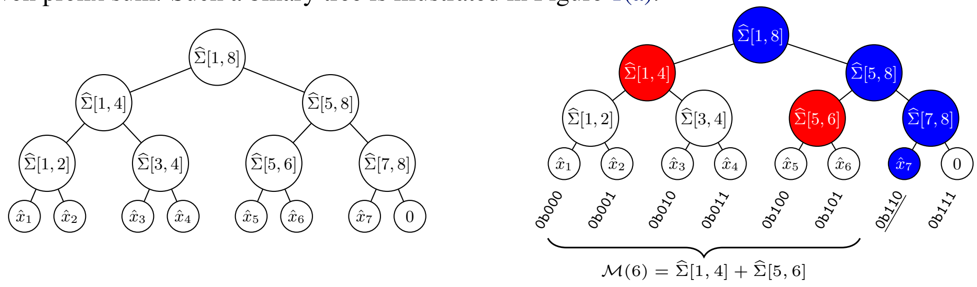
It is known that ρ-zCDP implies $(\rho + 2\sqrt{\rho \ln(1/\delta)}, \delta)$-differential privacy for every $\delta > 0$ (Bun & Steinke, 2016).

Lastly, when comparing counting mechanisms based on Gaussian noise, we note that it is sufficient to look at the variance. For a single output $\mathcal{M}(t)$, the variance $\mathrm{Var}[\mathcal{M}(t)]$ is the only relevant metric, as it completely describes the output distribution. For a series of outputs $\mathcal{M}_x$, and related norms $\|\mathcal{M}_x\|_p$, we note that a mechanism with lower variance will be more concentrated in each coordinate and have lower p^{th} moment, allowing for tighter bounds on the norm.

2.3 Differentially private continual counting

We next describe two approaches to continual counting.

Binary mechanism. The binary mechanism (Chan et al., 2011; Dwork et al., 2010; Gehrke et al., 2010; Hay et al., 2010) can be considered the canonical mechanism for continual counting. In this section we present a variant of it where only left subtrees are used. The mechanism derives its name from the fact that a binary tree is built from the input stream. Each element from the stream is assigned a leaf in the binary tree, and each non-leaf node in the tree represents a p-sum of all elements in descendant leaves. All values are stored noisily in nodes, and nodes are added together to produce a given prefix sum. Such a binary tree is illustrated in Figure 1(a).



(a) Binary mechanism tree for $T = 7$.

(b) Computation of $\mathcal{M}(6)$ using Algorithm 1.

Figure 1: Binary trees for a sequence of length $T = 7$. In Figure 1(b) each leaf is labeled by $\text{bin}(t-1)$, and it illustrates how the prefix sum up to $t = 6$ can be computed from $\text{bin}(t)$. Blue nodes describe the path taken by Algorithm 1, and the sum of red nodes form the desired output $\mathcal{M}(6)$.

A more detailed exposition of the binary mechanism is given in the appendix and Section 3.1. Here we settle for stating that the ρ-zCDP binary mechanism achieves $\text{Var}[\mathcal{M}(t)] = O(\log(T)^2)$ and is known to be computationally efficient: to release all prefix sums up to a given time $t \leq T$ requires only $O(\log(T))$ space and $O(t)$ time.

Matrix mechanism. The binary mechanism belongs to a more general class of mechanisms called *matrix mechanisms* (Li et al., 2015). Computing a prefix sum is a linear operation on the input stream $x \in \mathbb{R}^T$, and computing all prefix sums up to a given time T can therefore be represented by a matrix $A \in \mathbb{R}^{T\times T}$, where $c(t) = (Ax)_t$. A is here a lower-triangular matrix of all 1s. The matrix mechanism characterizes solutions to the continual counting problem by factorizing A as $A = LR$ with corresponding mechanism $\mathcal{M}_x = L(Rx + z)$, $z \sim \mathcal{F}^n$ where n is the dimension of Rx.

Intuitively Rx represents linear combinations of the stream, which are made private by adding noise z, and which lastly are aggregated by the linear transformation L. To achieve ρ-zCDP for this mechanism, we let $z \sim \mathcal{N}(0, \frac{\Delta^2}{2\rho})^n$ where $\Delta = \max_i \|Re_i\|_2$, e_i being the i^{th} unit vector. It follows that the corresponding output noise becomes Lz where $(Lz)_i \sim \mathcal{N}(0, \frac{\Delta^2}{2\rho}\|L_i\|_2^2)$

3 Our mechanism

To introduce our variant of the binary mechanism, we need to return to the original mechanism.

3.1 Closer analysis of the binary mechanism

As has been pointed out in earlier work (Denisov et al., 2022; Henzinger et al., 2023), a naive implementation of the binary mechanism will lead to variance that is non-uniform with respect to time: the number of 1s in the bitwise representation of t determines the variance. To underline this, consider the pseudo-code in Algorithm 1 for computing a prefix sum given a binary mechanism tree structure. The code assumes that the tree has $\geq t + 1$ leaves, a detail that only matters when t is a power of 2 and allows us to never use the root node. To illustrate a simple case, see Figure 1(b).

We can make the following observations:

- $\text{bin}(t-1)$ encodes a path from the root to the leaf where x_t is stored.
- To compute prefix sums using Algorithm 1, we only need to store values in "left children".

Combining these observations, we get the following result:

Proposition 3.1. *The squared ℓ_2-sensitivity Δ^2 of x_t is equal to the number of 0s in* $\text{bin}(t-1)$.

To see this, note that the number of 0s in $\text{bin}(t-1)$ is equal to the number of left-children that are passed through to reach the given node. Changing the value of x_t impacts all its ancestors, and since

Algorithm 1 Prefix Sum for Binary Mechanism

```
1: Input: binary tree of height $h$ storing $\widehat{\Sigma}[a, b]$ for $b \leq t$, time $t \leq 2^h - 1$
2: $output \leftarrow 0$
3: $s \leftarrow \mathrm{bin}(t)$ {padded to $h$ bits by adding zeros}
4: $a \leftarrow 1; b \leftarrow 2^h$
5: for $i = h - 1$ to $0$ do
6:     $d = \lfloor \frac{a+b}{2} \rfloor$
7:     if $s_i = 1$ then
8:         $output \leftarrow output + \widehat{\Sigma}[a, d]$
9:         $a \leftarrow d + 1$
10:    else
11:        $b \leftarrow d$
12:    end if
13: end for
14: return $output$
```

only left-children are used for prefix sums, the result immediately follows. This does not address the volatility of variance with respect to time. However, studying Algorithm 1 gives the reason:

Proposition 3.2. $\mathrm{Var}[\mathcal{M}(t)]$ *is proportional to the number of 1s in* $\mathrm{bin}(t)$.

The result follows from the fact that the number of terms added together for a given prefix sum $c(t)$ is equal to the number of 1s in $\mathrm{bin}(t)$, since Line 8 is executed for each such 1. In this view, each node that is used for the prefix sum at time t can be identified as a prefix string of $\mathrm{bin}(t)$ that ends with a 1.

We will return to Proposition 3.1 and Proposition 3.2 when constructing our smooth mechanism, but for now we settle for stating that combined they give the exact variance at each time step for the binary mechanism. Following Li et al. (2015), to make the mechanism private we have to accommodate for the worst sensitivity across all leaves, which yields Theorem 3.3.

Theorem 3.3 (Exact Variance for Binary Mechanism (Chan et al., 2011; Dwork et al., 2010))**.** *For any* $\rho > 0$, $T > 1$*, the* ρ*-zCDP binary mechanism* $\mathcal{M}$ *based on Algorithm 1 achieves variance*

$$\mathrm{Var}[\mathcal{M}(t)] = \frac{\lceil \log(T+1) \rceil}{2\rho} \left\| \mathrm{bin}(t) \right\|_1$$

for all $1 \leq t \leq T$*, where* $\left\| \mathrm{bin}(t) \right\|_1$ *is equal to the number of 1s in* $\mathrm{bin}(t)$.

3.2 A smooth binary mechanism

Based on the analysis, a naive idea to improve the binary mechanism would be to only consider leaves with "favorable" time indices. To make this a bit more precise, we ask the following question: could there be a better mechanism in which we store elements in only a subset of the leaves in the original binary tree, and then use Algorithm 1 to compute the prefix sums? We give an affirmative answer.

Consider a full binary tree of height h where we let the leaves be indexed by i (1-indexed). Given a sequence of elements $x \in \{0, 1\}^T$ (we assume a great enough height h to accommodate for all elements) and an integer $0 \leq k \leq h$, we conceptually do the following for each leaf with index i:

- If $\mathrm{bin}(i - 1)$ has k 0s, store the next element of the stream in the leaf.
- Otherwise store a token 0 in the leaf.

More rigorously stated, we are introducing an injective mapping from time to leaf-indices $m : \{1, \ldots, T + 1\} \rightarrow [2^h + 1]$, such that $m(t)$ is the $(t - 1)^{st}$ smallest h-bit integer with k 0s in its binary representation. x_t gets stored in the leaf with index $m(t)$, and to compute $\mathcal{M}(t)$, we would add p-sums based on $\mathrm{bin}(m(t + 1))$. The resulting algorithm works analogously to Algorithm 1 but the details differ slightly. See Algorithm 2 in the appendix for the full specification.

It follows that the maximum sensitivity is proportional to k, and that any prefix sum will be a sum of $h - k$ nodes. Importantly, the latter fact removes the dependence on t for the variance.

Choosing a k. The optimal choice of k depends on the differential privacy paradigm. Here we only consider ρ-zCDP. Since each element will appear in at most k p-sums, the corresponding ℓ_2-sensitivity becomes $\Delta = \sqrt{k}$. Furthermore, since each prefix sum is computed as a sum of $h-k$ nodes, the variance for a given prefix sum becomes $\mathrm{Var}[\mathcal{M}(t)] = (h-k)\cdot\frac{k}{2\rho}$, which for $k = h/2$ gives a leading constant of $1/4$ compared to the maximum variance of the binary mechanism. This choice of k is valid if the tree has an even height h, and it maximizes the number of available leaves.

A penalty in height. The analysis above assumes that we have a tree of sufficient height. If we before had a tree of height $\lceil\log(T+1)\rceil$, we now need a tree of height $h \geq \lceil\log(T+1)\rceil$ to have enough leaves with the right ratio of 1s in their index. To account for this, we let h be the smallest even integer such that $h \geq \lceil\log(T)\rceil + a\log\log T$, where a is a constant. For our new tree to support as many prefix sums, we need that $\binom{h}{h/2} \geq T+1$. This holds for $a > 1/2$ and sufficiently large T, but we show it for $a = 1$ next. Using Stirling's approximation in the first step, we can establish that

$$\binom{h}{h/2} \geq \frac{1}{\sqrt{2h}}2^h \geq \frac{2^{\log T+\log\log T}}{\sqrt{2}\sqrt{\lceil\log T\rceil+\log\log T}} = \frac{\log T}{\sqrt{2}\sqrt{\lceil\log T\rceil+\log\log T}}T \ ,$$

which is at least $T+1$ for $T \geq 13$.

Resulting variance. This height penalty makes $\mathrm{Var}[\mathcal{M}(t)]$ no longer scale as $\log(T+1)^2$, but $(\log(T) + a\log\log(T))^2$. Nevertheless, we can state the following:

Lemma 3.4. *For any $\rho > 0$, $T \geq 13$, the ρ-zCDP smooth binary mechanism $\mathcal{M}$ achieves variance*

$$\mathrm{Var}[\mathcal{M}(t)] = \frac{1+o(1)}{8\rho}\log(T)^2$$

where $1 \leq t \leq T$, and the $o(1)$ term is $\frac{2\log\log(T)}{\log(T)} + \left(\frac{\log\log(T)}{\log(T)}\right)^2$.

which is an improvement over the original binary mechanism by a factor of $1/4$ with regard to the leading term. This improvement is shown empirically in Section 4.

Constant average time per output. When outputting T prefix sums continuously while reading a stream, we only have to store the noise of the nodes, not the p-sums themselves. To make this more explicit, let S_t describe the set of nodes (p-sum indices) that the smooth mechanism adds together to produce the output at time t. To produce $\mathcal{M}(t+1)$ given $\mathcal{M}(t)$, we effectively do:

$$\mathcal{M}(t+1) = \mathcal{M}(t) + x_{t+1} + \sum_{(i,j)\in S_{t+1}\setminus S_t} X[i,j] - \sum_{(i,j)\in S_t\setminus S_{t+1}} X[i,j] \ .$$

To quantify the cost, we need to deduce how many nodes are replaced from t to $t+1$, which means reasoning about S_t and S_{t+1}. Recalling that each element in S_t can be identified by a prefix string of $\mathrm{bin}(m(t+1))$ terminating with a 1, consider Figure 2. Based on the pattern shown in Figure 2, where the leaf indices only differ in their least significant bits, we get that $|S_{t+1}\setminus S_t| = |S_t\setminus S_{t+1}| = n$, where n is the number of 1s in the least significant block of 1s. We formalize this observation next to give a bound on the average cost when outputting a sequence of prefix sums.

Lemma 3.5. *Assuming the cost of replacing a node in a prefix sum is 1, then the cost to release all $\binom{2k}{k} - 1$ prefix sums in the tree of height $2k$ using the smooth binary mechanism is at most $2\binom{2k}{k}$.*

Proof. As argued before, to compute $\mathcal{M}(t+1)$ given $\mathcal{M}(t)$ we need to replace a number of nodes equal to the size n of the least significant block of 1s in $\mathrm{bin}(m(t+1))$. We can therefore directly

$$\mathrm{bin}(m(t+1)) = \boxed{\cdots}\boxed{0}\underbrace{\boxed{1\cdots1}}_{n}\boxed{0\cdots0} \ ; \ \mathrm{bin}(m(t+2)) = \boxed{\cdots}\boxed{1}\boxed{0\cdots0}\underbrace{\boxed{1\cdots1}}_{n-1}$$

Figure 2: The least significant bits of two leaf indices in a full binary tree that are neighboring with respect to time when used in the smooth binary mechanism. If the first cluster of 1s in $\mathrm{bin}(m(t+1))$, counted from the least significant bit, has n 1s then n nodes in total will be replaced from t to $t+1$.

compute the total cost by enumerating all valid indices with different block sizes as

$$\begin{aligned}\text{cost} &= -k + \sum_{n=1}^{k} \sum_{i=k-n}^{2k-n-1} n \binom{i}{k-n} = -k + \sum_{n=1}^{k} n \binom{2k-n}{k-n+1} \\ &= -k + \sum_{i=1}^{k} \sum_{j=1}^{i} \binom{k-1+j}{k-1} = -2k + \sum_{i=1}^{k} \binom{k+i}{k} = \binom{2k+1}{k+1} - (2k+1) \leq 2\binom{2k}{k}\end{aligned}$$

where the initial $-k$ term comes from excluding the last balanced leaf index in the tree. □

In particular, this implies that the average cost when releasing all prefix sums in a full tree is ≤ 2. For T that does not use all leaves of a tree, comparing to the closest T' where $T \leq T' = \binom{2k}{k} - 1$ also implies an average cost of ≤ 4 for arbitrary T. For a single output the cost is $O(\log(T))$. It is not hard to check that computing $\mathrm{bin}(m(t+1))$ from $\mathrm{bin}(m(t))$ can be done in constant time using standard arithmetic and bitwise operations on machine words (see our supplied code for details).

Logarithmic space. The argument for the binary mechanism only needing $O(\log(T))$ space extends to our smooth variant. We state this next in Lemma 3.6, and supply a proof in the appendix.

Lemma 3.6. *The smooth binary mechanism computing prefix sums runs in $O(\log(T))$ space.*

Finishing the proof for Theorem 1.1. The last, and more subtle, property of our mechanism is that the noise at each time step not only has constant variance, but it is identically distributed. When working with the Gaussian mechanism, as we do in this paper, this is not surprising: a sum of Gaussians is still Gaussian. But there is nothing precluding our revised binary mechanism from being used with different noise distributions, e.g. Laplacian for the purpose of achieving ε-DP. By fixing the number of 1s in each leaf index, we are in fact fixing the number of noise terms that get added together for each count, and by extension the final error distribution at each t. Combining Lemmas 3.4, 3.5 and 3.6 together with the last observation, we arrive at Theorem 1.1.

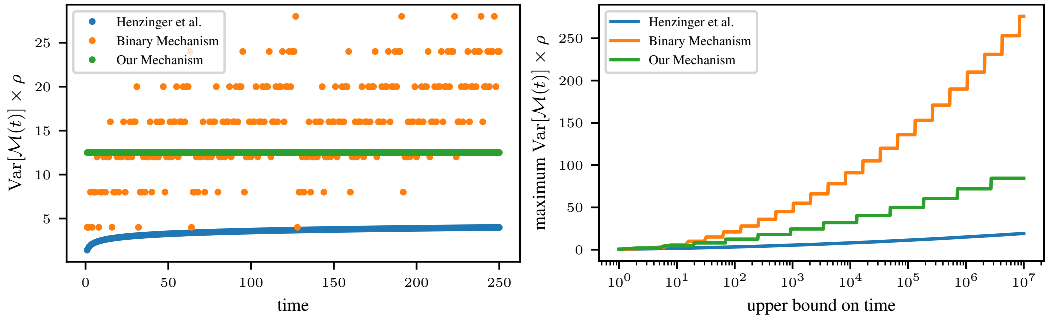
4 Comparison of mechanisms

In this section we review how the smooth binary mechanism compares to the original binary mechanism, and the matrix mechanism of Henzinger et al. (2023). We do not compare to Denisov et al. (2022) since their method is similar to that of Henzinger et al. (2023) in terms of error, and less efficient in terms of time and space usage.

Variance comparison. To demonstrate how the variance behaves over time for our smooth binary mechanism, see Figure 3. Given a fix T, the tree-based mechanisms compute the required tree height to support all elements, and the matrix mechanism a sufficiently large matrix. The volatility of the error in the regular binary mechanism is contrasted by the stable noise distribution of our smooth binary mechanism, as demonstrated in Figure 3(a). In terms of achieving the lowest variance, the matrix mechanism in Henzinger et al. (2023) is superior, as expected. This result is replicated in Figure 3(b) where our mechanism offers a substantial improvement in terms of maximum variance over the original binary mechanism, but does not improve on Henzinger et al. (2023).

Computational efficiency comparison. While our mechanism does not achieve as low noise as the mechanism of Henzinger et al. (2023), it scales well in time and space, and with respect to the dimensionality of the stream elements. This is empirically demonstrated in Figure 4. Since the matrix used in Henzinger et al. (2023) is a Toeplitz matrix, the scipy method "matmul_toeplitz" (based on FFT and running in time $O(dT \log T)$ to produce a d-dimensional output) was used to speed up the matrix multiplication generating the noise in Figure 4(a). However, note that using a single matrix multiplication to produce $\{\mathcal{M}(1), \ldots, \mathcal{M}(T)\}$ means that we are not in the "streaming" setting. Even then, their matrix-based mechanism does not scale as well as ours.

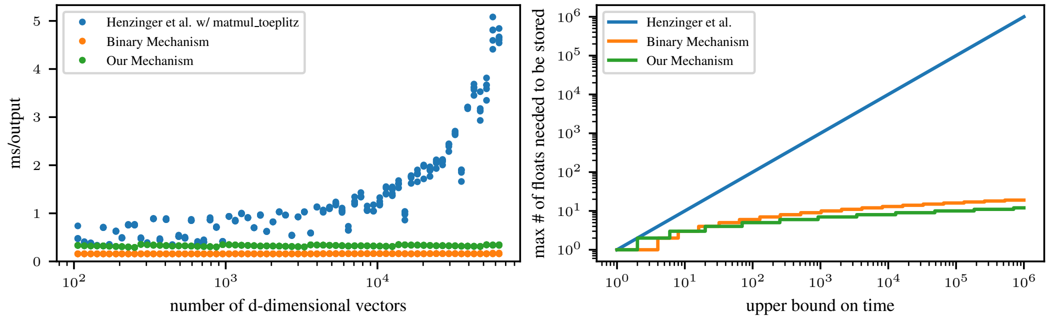
The discrepancy in the computation time scaling can largely be attributed to space: the needed space for tree-based methods scales logarithmically with T, and linearly with T for matrix-multiplication based methods. This is demonstrated in Figure 4(b). As to the difference in computation time between the tree-based methods, the smooth binary mechanism generates twice as much fresh noise per time step on average, which likely is the dominating time sink in this setup.



(a) Running variance for $T = 250$.

(b) Maximum variance vs. upper bound on time.

Figure 3: Comparison of variance between the mechanism in Henzinger et al. (2023), the standard binary mechanism and our mechanism. Figure 3(a) shows $\mathrm{Var}[\mathcal{M}(t)]$ for $1 \leq t \leq T$ for $T = 250$, whereas Figure 3(b) shows the maximum variance that each mechanism would attain for a given upper bound on time. At the last time step in Figure 3(b), our mechanism reduces the variance by a factor of 3.27 versus the binary mechanism. Note: the variance for the standard binary mechanism shown in these plots is slightly lower than what was shown in the submitted version.



(a) Average computational time per output.

(b) Maximum space needed vs upper bounds on time.

Figure 4: Comparison of computational efficiency between the mechanism in Henzinger et al. (2023), the binary mechanism and our mechanism. Figure 4(a) shows the computation time spent per d-dimensional input. The simulation was run 5 times for each method, meaning each method has 5 data points in the plot per time step. The computation was performed for elements of dimension $d = 10^4$, was run on a Macbook Pro 2021 with Apple M1 Pro chip and 16 GB memory using Python 3.9.6, scipy version 1.9.2, and numpy version 1.23.3. Figure 4(b) shows the maximum number of floats that has to be stored in memory when outputting all prefix sums up to a given time, assuming binary input.

5 Conclusion and discussion

We presented an improved "smooth" binary mechanism that retains the low time complexity and space usage while improving the additive error and achieving stable noise at each time step. Our mechanism was derived by performing a careful analysis of the original binary mechanism, and specifically the influence of the binary representation of leaf indices in the induced binary tree. Our empirical results demonstrate the stability of the noise and its improved variance compared to the binary mechanism. The matrix mechanism of Henzinger et al. (2023) offers better variance, but is difficult to scale to a large number of time steps, especially if we need high-dimensional noise.

We note that the smooth binary mechanism can be extended to ε-DP. The optimal fraction of 1s in the leaves of the binary tree would no longer be $1/2$. An interesting problem is to find mechanisms that have lower variance and attractive computational properties. It is possible that the dependence on T can be improved by leaving the matrix mechanism framework, but in absence of such a result the best we can hope is to match the variance obtained by Henzinger et al. (2023).

References

Bun, M. and Steinke, T. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In Hirt, M. and Smith, A. (eds.), *Theory of Cryptography*, pp. 635–658, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-53641-4.

Cardoso, A. R. and Rogers, R. Differentially private histograms under continual observation: Streaming selection into the unknown. In Camps-Valls, G., Ruiz, F. J. R., and Valera, I. (eds.), *International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event*, volume 151 of *Proceedings of Machine Learning Research*, pp. 2397–2419. PMLR, 2022. URL `https://proceedings.mlr.press/v151/rivera-cardoso22a.html`.

Chan, T.-H. H., Shi, E., and Song, D. Private and Continual Release of Statistics. *ACM Transactions on Information and System Security*, 14(3):26:1–26:24, November 2011. ISSN 1094-9224. doi: 10.1145/2043621.2043626. URL `https://doi.org/10.1145/2043621.2043626`.

Chan, T. H. H., Li, M., Shi, E., and Xu, W. Differentially Private Continual Monitoring of Heavy Hitters from Distributed Streams. In Fischer-Hübner, S. and Wright, M. (eds.), *Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pp. 140–159, Berlin, Heidelberg, 2012. Springer. ISBN 978-3-642-31680-7. doi: 10.1007/978-3-642-31680-7_8.

Choquette-Choo, C. A., McMahan, H. B., Rush, K., and Thakurta, A. Multi-epoch matrix factorization mechanisms for private machine learning. *CoRR*, abs/2211.06530, 2022. doi: 10.48550/arXiv.2211.06530. URL `https://doi.org/10.48550/arXiv.2211.06530`.

Denisov, S., McMahan, H. B., Rush, J., Smith, A. D., and Thakurta, A. G. Improved differential privacy for SGD via optimal private linear operators on adaptive streams. In *NeurIPS*, 2022. URL `http://papers.nips.cc/paper_files/paper/2022/hash/271ec4d1a9ff5e6b81a6e21d38b1ba96-Abstract-Conference.html`.

Dinur, I. and Nissim, K. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '03, pp. 202–210, New York, NY, USA, June 2003. Association for Computing Machinery. ISBN 978-1-58113-670-8. doi: 10.1145/773153.773173. URL `https://doi.org/10.1145/773153.773173`.

Dwork, C. and Roth, A. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3-4):211–407, 2013. ISSN 1551-305X, 1551-3068. doi: 10.1561/0400000042.

Dwork, C., Naor, M., Pitassi, T., and Rothblum, G. N. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing*, STOC '10, pp. 715–724, New York, NY, USA, June 2010. Association for Computing Machinery. ISBN 978-1-4503-0050-6. doi: 10.1145/1806689.1806787. URL `https://doi.org/10.1145/1806689.1806787`.

Fichtenberger, H., Henzinger, M., and Upadhyay, J. Constant matters: Fine-grained Complexity of Differentially Private Continual Observation, December 2022. URL `http://arxiv.org/abs/2202.11205`. arXiv:2202.11205 [cs].

Gehrke, J., Wang, G., and Xiao, X. Differential privacy via wavelet transforms. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pp. 225–236, Los Alamitos, CA, USA, mar 2010. IEEE Computer Society. doi: 10.1109/ICDE.2010.5447831. URL `https://doi.ieeecomputersociety.org/10.1109/ICDE.2010.5447831`.

Han, Y., Liang, Z., Liang, Z., Wang, Y., Yao, Y., and Zhang, J. Private Streaming SCO in ℓ_p geometry with Applications in High Dimensional Online Decision Making. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 8249–8279. PMLR, June 2022. URL `https://proceedings.mlr.press/v162/han22d.html`. ISSN: 2640-3498.

Hay, M., Rastogi, V., Miklau, G., and Suciu, D. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.*, 3(1–2):1021–1032, sep 2010. ISSN 2150-8097. doi: 10.14778/1920841.1920970. URL `https://doi.org/10.14778/1920841.1920970`.

Henzinger, M., Upadhyay, J., and Upadhyay, S. Almost tight error bounds on differentially private continual counting. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 5003–5039, 2023. doi: 10.1137/1.9781611977554.ch183. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611977554.ch183`.

Honaker, J. Efficient use of differentially private binary trees. *Theory and Practice of Differential Privacy (TPDP 2015), London, UK*, 2:26–27, 2015.

Huang, Z., Qiu, Y., Yi, K., and Cormode, G. Frequency estimation under multiparty differential privacy: one-shot and streaming. *Proceedings of the VLDB Endowment*, 15(10):2058–2070, September 2022. ISSN 2150-8097. doi: 10.14778/3547305.3547312. URL `https://doi.org/10.14778/3547305.3547312`.

Kairouz, P., Mcmahan, B., Song, S., Thakkar, O., Thakurta, A., and Xu, Z. Practical and Private (Deep) Learning Without Sampling or Shuffling. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 5213–5225. PMLR, July 2021. URL `https://proceedings.mlr.press/v139/kairouz21b.html`. ISSN: 2640-3498.

Li, C., Miklau, G., Hay, M., McGregor, A., and Rastogi, V. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB Journal*, 24(6):757–781, December 2015. ISSN 0949-877X. doi: 10.1007/s00778-015-0398-x. URL `https://doi.org/10.1007/s00778-015-0398-x`.

McMahan, H. B. and Thakurta, A. Federated Learning with Formal Differential Privacy Guarantees, 2022. URL `https://ai.googleblog.com/2022/02/federated-learning-with-formal.html`. Accessed online 2023-05-15.

Smith, A., Thakurta, A., and Upadhyay, J. Is Interaction Necessary for Distributed Private Learning? In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 58–77, May 2017. doi: 10.1109/SP.2017.35. ISSN: 2375-1207.

Upadhyay, J. Sublinear Space Private Algorithms Under the Sliding Window Model. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 6363–6372. PMLR, May 2019. URL `https://proceedings.mlr.press/v97/upadhyay19a.html`. ISSN: 2640-3498.

A A more detailed introduction to the binary mechanism

To introduce the binary mechanism, it will be useful to first consider intuitive but sub-optimal candidate solutions to the problem. This step-wise introduction is inspired by Chan et al. (2011).

First naive approach. A first naive solution to the private continual release problem would be to directly add noise to the output. Formalized, a mechanism $\mathcal{M}$ that releases the corresponding noisy p-sum for each time step t:

$$\mathcal{M}(t) = \widehat{\Sigma}[1, t] \ .$$

The problem with this approach is that if you flip the value of an element x_t then for every $t' \geq t$, $\widehat{\Sigma}[1, t']$ is impacted. This implies a sensitivity scaling with T, and consequently a high level of noise needed to preserve privacy. To achieve ρ-zCDP you would thus need to add noise scaling with T, and therefore we would have $\mathrm{Var}[\mathcal{M}(t)] = \frac{T}{2\rho}$.

Second naive approach. Based on this observation, a different approach would be to instead add noise to the input, i.e. store x_t as $\widehat{\Sigma}[t, t]$ and construct $\mathcal{M}$ as

$$\mathcal{M}(t) = \sum_{i=1}^{t} \widehat{\Sigma}[i, i] \ .$$

This makes the sensitivity of any element x_t constant, as any element x_t impacts a single noisy p-sum $\widehat{\Sigma}[t, t]$. Unfortunately we are instead forced to add up a number of noisy p-sums that increase linearly in T. For ρ-zCDP, we thus get $\mathrm{Var}[\mathcal{M}(t)] = \frac{t}{2\rho}$.

The binary mechanism. The binary mechanism (Chan et al., 2011; Dwork et al., 2010; Gehrke et al., 2010; Hay et al., 2010) represents a compromise between these two approaches of adding noise to the input and adding noise to the output. It is a compromise because it achieves a logarithmic sensitivity and adds a logarithmic number of noise terms to produce an output, thus achieving a variance of $O(\log(T)^2)$ rather than $O(T)$.

The intuition behind the binary mechanism is that it partitions the input stream into overlapping p-sums which guarantee that no element is part of more than $\lceil \log(T+1) \rceil$ p-sums, and that every prefix sum can be expressed as a sum of at most $\lceil \log(T+1) \rceil$ p-sums. The dependence on $T+1$ rather than T is a consequence of only using left subtrees.

We briefly sketch the argument for the variance next. To make the mechanism differentially private every p-sum stored in the tree is stored noisily. Focusing on ρ-zCDP, we get that the ℓ_2-sensitivity is $\sqrt{\lceil \log(T+1) \rceil}$, and consequently that adding Gaussian noise with variance $\frac{\lceil \log(T+1) \rceil}{2\rho}$ suffices to make the output satisfy ρ-zCDP. Since every final prefix sum is a sum of at most $\lceil \log(T+1) \rceil$ p-sums, the resulting error becomes Gaussian with variance at most $\frac{\lceil \log(T+1) \rceil^2}{2\rho}$.

B Algorithm for prefix sum using smooth binary mechanism

For completion, the smooth binary mechanism variant of Algorithm 1 is given below as Algorithm 2. Recall that $m(t)$ denotes the (1-based) index of the leaf where x_t is stored. The only change between this variant and the one for the binary mechanism is that the size of the tree is decided by the time-to-leaf-index mapping $m(t)$ rather than t, and that the string s on Line 3 is $\mathrm{bin}(m(t+1))$ rather than $\mathrm{bin}(t)$.

C Proof of Lemma 3.6

The proof of Lemma 3.6 was omitted from the main text, it is given next.

Proof. The critical observation to make is that once a given p-sum is removed from a prefix sum, then it will never re-appear. Let its associated prefix string at time t be s of length l. If we look at the same l bit positions in $\mathrm{bin}(m(t))$ as t increases and interpret it as a number, then it is monotonously

Algorithm 2 Prefix Sum for Smooth Binary Mechanism

```
Input: binary tree of height $h$ storing $\widehat{\Sigma}[a,b]$ for $b \leq m(t)$, time $t$ where $m(t+1) \leq 2^h$
$output \leftarrow 0$
$s \leftarrow \mathrm{bin}(m(t+1))$ {padded to $h$ bits by adding zeros}
$a \leftarrow 1$; $b \leftarrow 2^h$
for $i = h-1$ to $0$ do
    $d = \lfloor \frac{a+b}{2} \rfloor$
    if $s_i = 1$ then
        $output \leftarrow output + \widehat{\Sigma}[a,d]$
        $a \leftarrow d+1$
    else
        $b \leftarrow d$
    end if
end for
return $output$
```

increasing with t. This implies that once a given prefix-string s in $\mathrm{bin}(m(t))$ disappears at $t' > t$ then it will not be encountered again. We can therefore free up the memory used for storing any p-sum the moment it is no longer used. Because of this, it suffices to only store the p-sums in the active prefix sum at any time, of which there are at most $O(\log(T))$, proving the statement. $\square$