Artifact Evaluated NDSS
Available
Functional
Reproduced

Delay-allowed Differentially Private Data Stream Release

Xiaochen Li[†], Zhan Qin[*],[✉], Kui Ren[*], Chen Gong[†], Shuya Feng[‡], Yuan Hong[‡], Tianhao Wang[†]
[*]The State Key Laboratory of Blockchain and Data Security, Zhejiang University
[†]University of Virginia, [‡]University of Connecticut
{qinzhan, kuiren}@zju.edu.cn; {xiaochenli, chengong, tianhao}@virginia.edu; {shuya.feng, yuan.hong}@uconn.edu

***Abstract*—The research on tasks involving differentially private data stream releases has traditionally centered around real-time scenarios. However, not all data streams inherently demand real-time releases, and achieving such releases is challenging due to network latency and processing constraints in practical settings. We delve into the advantages of introducing a delay time in stream releases. Concentrating on the event-level privacy setting, we discover that incorporating a delay can overcome limitations faced by current approaches, thereby unlocking substantial potential for improving accuracy.**

Building on these insights, we developed a framework for data stream releases that allows for delays. Capitalizing on data similarity and relative order characteristics, we devised two optimization strategies, group-based and order-based optimizations, to aid in reducing the added noise and post-processing of noisy data. Additionally, we introduce a novel sensitivity truncation mechanism, significantly further reducing the amount of introduced noise. Our comprehensive experimental results demonstrate that, on a data stream of length $18,319$, allowing a delay of 10 timestamps enables the proposed approaches to achieve a remarkable up to a $30\times$ improvement in accuracy compared to baseline methods. Our code is open-sourced.

I. Introduction

The continuous collection and release of data streams play a crucial role in various industries. For example, Smart Grids continuously gather data from smart meters and grid sensors to optimize energy distribution and improve overall efficiency [11]; Wearable devices continuously capture patient vitals to enable healthcare professionals to monitor patients' health status [12]. Typically, data streams are gathered by local devices from data providers and then released to data consumers. This process means that once data leaves the client's device, it falls outside the data provider's control, inevitably leading to privacy concerns among the public. As a result, dedicated research efforts using Differential Privacy (DP) techniques focus on privacy-preserving data stream release [35], [42], [29], [10], [14], [26], [9].

This paper was completed while Xiaochen Li was visiting University of Virginia as a PhD student at Zhejiang University. Zhan Qin is the Corresponding author.

Network and Distributed System Security (NDSS) Symposium 2025
24–28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230828
www.ndss-symposium.org

DP is widely acknowledged as a standard privacy definition for statistical databases. It adds crafted noise to either the data or the statistical outcomes, thwarting any attempts to discern sensitive information from the perturbed results. Meanwhile, the noise inevitably reduces the utility of the released results, it poses a particularly challenging issue for releasing infinite data streams with a vast data domain. This is especially true for real-time data collection and release, an area where much of the existing research is concentrated. Therefore, the central focus of existing research, including our work, is on how to release a data stream that adheres to DP guarantees while maintaining optimal utility.

To advance the DP real-time data stream release, we propose two key questions as follows. (a) *Is real-time release required for all data streams in practice?* While many data stream tasks necessitate real-time decisions based on streaming data from client devices, e.g., identifying defects in manufacturing or potential drug interactions in healthcare, not all tasks demand absolute real-time requirements. Some tasks, like user behavior analysis for personalized product recommendations or load forecasting for utility planning, exhibit high time sensitivity but can tolerate a short or even longer delay. (b) *Can approaches theoretically designed for real-time release truly achieve real-time in practice?* The attainment of absolute real-time approaches faces formidable challenges, considering factors like network latency and processing time in practical scenarios where data collectors and consumers are distinct entities. Acknowledging these challenges, we propose exploring the feasibility of incorporating a tolerance for releasing delays introduced by local client devices, in addition to tolerances for communication and processing delays.

In this paper, we focus on the event-level DP setting, where each streaming data is protected by ϵ-DP. Reducing the amount of noise addition [42], [35] and post-processing of the noisy results [28], [9], [40], [34], [27], [5] are the only two ways to optimize the accuracy of the released data stream under the event-level privacy setting. Among these post-processing solutions, we find that they all follow the group-based post-processing framework proposed in PeGaSuS [9]. The core idea is to smooth the noisy data together with others similar to it at consecutive timestamps to reduce the error introduced by noise. However, we identify two issues with this framework. One is that it is difficult to meet the conditions for accuracy improvement brought by post-processing at consecutive timestamps. The other is that data can only be smoothed with historically released results in a group. These two issues may even lead to potential accuracy disasters in the released results but are inherently unsolvable in real-time data release scenarios. Driven by the above observations, we further contemplate: *How much could data-releasing approaches benefit from a delay time?*

Introducing a delay time during data release can be a solution to the challenges faced by the existing works, unlocking substantial potential for accuracy improvement. This is attributed to two key advantages: (a) the release strategy can be tailored based on the data to be released, eliminating the reliance on predictions from historical data. (b) processing multiple data points together within a delay time window offers an opportunity to reduce the added noise, while the estimation error in real-time scenarios is at least $\Omega(\sqrt{t})$ (t is the length of the stream prefix) as post-processing cannot mitigate the noise amount.

In this paper, we introduce a delay-allowed data-releasing framework, comprising three integral modules: *optimization module*, *noise addition module*, and *post-processing module*. Specifically, the optimization module is tailored to capture the characteristics of streaming data within a delay time, strategically aiding in noise reduction or mitigating noise impact through post-processing. Leveraging the output from the optimization module, the noise addition module introduces noise to the data or data summation. Following this, the post-processing module smooths the noisy results by applying a smoother, enhancing the overall utility of the released data.

Within the optimization module, we present two distinct strategies rooted in the similarity and order of the data. Firstly, we present two group-based strategies to process streaming data in batches. In particular, our proposed non-continuous grouping strategy greatly alleviates the limitation of group length on the effectiveness of the post-processing. Additionally, we propose two order-based optimization strategies from a novel perspective, as the order consistency of the noisy results is also a crucial factor in enhancing the utility of the released results. They are more advantageous than group-based strategies on short delay times.

In the noise addition module, we implement two enhancements aimed at reducing the amount of added noise. Firstly, we utilize the similarity of data to add noise to the data summation in the same group or bucket within a delay time. Then the noisy results are evenly distributed to each data through the post-processing module. This approach effectively reduces the total noise amount from $\Omega(\sqrt{t})$ to $\Omega(\sqrt{n_g})$, where n_g represents the number of groups/buckets. Furthermore, we design a sensitivity truncation mechanism to mitigate issues of excessive noise addition due to an excessively large data domain or outliers.

We conduct comprehensive experiments and the results validate the effectiveness of the proposed approaches. Specifically, allowing a 10 **timestamps delay** with $\epsilon = 0.1$ on a data stream of length $18,319$, BucOrder shows a remarkable **almost 30$\times$ improvement** in accuracy, and the discontinuous grouping approach also attains a notable **nearly 2.5$\times$ improvement** in accuracy compared to PeGaSuS. Moreover, our sensitivity truncation mechanism proves impactful, showcasing a substantial **2.8$\times$ improvement** in accuracy for the continuous grouping approach on a data stream with a domain size of $2,260,794$.

Contributions. We make the following contributions:

- We are the first to consider the practical implications of delay time in data stream release and its potential to surmount limitations in the accuracy achieved by existing approaches.
- We propose a novel delay-allowed framework, featuring two kinds of optimization strategies, a data sensitivity truncation mechanism, and an optimized noise addition method.
- Through comprehensive experiments, our results showcase a substantial improvement in accuracy facilitated by the introduced approaches harnessing the power of delay time.

II. Preliminary

A. Problem Statement

We assume that there exists a server that has a data stream $D = \{x_1, x_2, ...\}$ and aims to release a noisy stream $\tilde{D}$ for multiple analyses while preserving privacy. We will discuss the privacy requirement, namely, differential privacy (DP), later. We evaluate the accuracy of $\tilde{D}$ by computing the distance between $\tilde{D}$ and D.

One specific relaxation we introduce in this paper is we allow a delay time of w timestamps (instead of real-time release assumed in existing literature). Specifically, the server could release the noisy value $\tilde{x}_{i-w}$ at the i^{th} timestamp. There are two settings to process the data stream with a delay time:

- *Batch setting*. This setting is equal to partitioning the data stream into $|D|/w$ batches. The server can design release strategies and randomization methods based on all streaming data received in the entire delay time. The processing results of streaming data within a delay period are released collectively at the timestamp when the delay time concludes. It's important to note that the batched data here possess temporal relationships, which should be preserved after release.
- *Sliding window setting*. This setting releases privacy-preserving data at each timestamp, except for the initial w timestamps. At any given timestamp, all streaming data within the current delay time window is visible. As the window slides, the server continuously releases the data at the beginning timestamp of the window. This also implies that the release strategy and randomization method for each streaming data are formulated based on different windows.

In essence, processing the streaming data in batch can be considered a special case of processing in a sliding window, where each batch corresponds to a sliding window with w windows sliding next to each other. For the privacy setting in this paper, which does not involve privacy budget allocation between different timestamps, processing the streaming data in a sliding window does not necessarily offer advantages over processing in batch. Therefore, the methods proposed in this paper, unless explicitly applied for processing in sliding window, are predominantly designed for processing in batch.

B. Differential Privacy

Definition of Differential Privacy. In this paper, we focus on providing *event-level* Differential Privacy (DP) protection. We first present the definition of event-level DP.

Definition 1 *(*(ϵ, δ)-Differential Privacy*) [17]. An algorithm $\mathcal{M}$ satisfies (ϵ, δ)-differential privacy, where ϵ, $\delta \geq 0$, if and only if for any neighboring streams $\mathcal{D}$, $\mathcal{D}'$, and any possible output set $\mathcal{O} \subseteq Range(\mathcal{M})$, we have*

$$\Pr\left[\mathcal{M}(\mathcal{D}) \in \mathcal{O}\right] \leq e^{\epsilon}\Pr\left[\mathcal{M}(\mathcal{D}') \in \mathcal{O}\right] + \delta.$$

where $x_i = x'_i$ for all i except one index in any two neighboring streams $D = \{x_1, x_2, ...\}$ and $D' = \{x'_1, x'_2, ...\}$.

Here, ϵ indicates *privacy budget*, which is a non-negative parameter that measures the privacy loss in the data. The smaller ϵ means that the outputs of $\mathcal{M}$ on two neighboring streams are more similar, and thus the provided privacy guarantee is stronger. In addition, δ indicates the probability of $\mathcal{M}$ failing to satisfy DP. In this paper, we consider the case of $\delta = 0$, we write ϵ-DP for convenience.

Laplace Mechanism. In this paper, we provide DP protection by adding noise sampled from the Laplace distribution to the streaming data or statistical results. This mechanism is commonly used in the DP field to achieve ϵ-DP privacy guarantee. Specifically, the probability density function of Laplace distribution with an expected value of 0 is given by

$$\Pr(x) = \frac{1}{2\lambda}e^{\frac{-|x|}{\lambda}}.$$

Here, λ is the scale parameter, where $\lambda = \Delta_f/\epsilon$. Δ_f is the l_1-norm global sensitivity of the statistical result, which is defined as

$$\Delta_f = \max_{D_i, D'_i} \|f(D_i) - f(D'_i)\|.$$

In this paper, we add noise to streaming data, so the default sensitivity Δ_f is the size of the data domain $|D|$. To mitigate the impact of excessive Δ_f on the accuracy of noisy results, this paper also proposes method to optimize it.

Sequential Composition. For a sequence of DP mechanisms and there is a correlation between the statistical results of these mechanisms, the privacy can be calculated by composition theorem [33]. We present the definition of sequential composition as follows:

Definition 2 *(*Sequential Composition*) [33]. For a sequence of k mechanisms $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_k$ and $\mathcal{M}_i$ provides ϵ_i-DP, let D be a dataset, publishing $t = (t_1, t_2, .., t_k)$, where $t_1 = \mathcal{M}_1(D), t_2 = \mathcal{M}_2(t_1, D), ..., t_k = \mathcal{M}_k(t_1, ..., t_{k-1}, D)$, satisfies $\left(\sum_{i=1}^{k} \epsilon_i\right)$-DP.*

In this paper, we divide the privacy budget of each streaming data into several parts for computation in multiple steps. As long as the sum of these privacy budgets is equal to ϵ, the final released results still satisfy ϵ-DP according to this composition theorem.

Algorithm 1 Group-based Post Processing.

```
Input: Stream $\{x_1, x_2, \ldots\}$, privacy budget $\epsilon_g$ for grouping, threshold $\theta$.
Output: $\{\tilde{x}_1, \tilde{x}_2, \ldots\}$
 1: $G \leftarrow \emptyset, \Delta_s \leftarrow |D|, \tilde{\theta} \leftarrow \theta + \mathsf{Lap}(2\Delta_s/\epsilon_g)$
 2: for $i \leftarrow 1, 2, \ldots$ do
 3:     if $dev(G \cup x_i) + \mathsf{Lap}(4\Delta_s/\epsilon_g) < \tilde{\theta}$ then
 4:         $G \leftarrow G \cup x_i$
 5:         $\hat{x}_i \leftarrow$ perturb $x_i$
 6:         $\hat{G} \leftarrow \hat{G} \cup \hat{x}_i$
 7:         $\tilde{x}_i \leftarrow$SMOOTHER($\hat{G}$, $\hat{x}_i$)
 8:     else
 9:         $\hat{x}_i \leftarrow$ perturb $x_i$, $\tilde{x}_i \leftarrow \hat{x}_i$
10:         $G \leftarrow \emptyset, \tilde{\theta} \leftarrow \theta + \mathsf{Lap}(2\Delta_s/\epsilon_g)$
11:     end if
12: end for
13: return $\{\tilde{x}_1, \tilde{x}_2, \ldots\}$
```

C. Existing Solutions

The most basic method to release the event-level privacy-protected data streams for multiple analyses is to add noise to each data point and release the noisy results. Throughout the entire process, there are mainly two optimizations: one is truncating the data to an appropriate range to reduce the sensitivity of added noise [42], [35], and the other is applying post-processing to the noisy results to decrease the expected error [28], [9], [40], [34], [27], [5]. Existing work primarily focuses on one of the two optimizations. In this paper, we propose improvement strategies for both optimizations, while mainly focusing on designing post-processing methods.

We observe that all the current post-processing methods for streaming data [40], [27], [5] still rely on the group-based framework proposed in PeGaSuS [9]. Therefore, this paper starts by analyzing this group-based framework to identify key obstacles in the current real-time data publishing setting.

Group-based Post-Processing Framework. The framework maintains a data structure called 'group' to partition the data stream, which aggregates several consecutive similar noisy results and smoothes each current data utilizing the data already present in the group. The process is shown in Algorithm 1.

Specifically, the framework determines whether to accept the data into the group by calculating the deviation $dev(G)$ of the group at each timestamp. Denote $c_1, c_2, ...$ is data in the group G and $|G|$ is the size of the group G, $dev(G)$ is calculated as follows,

$$dev(G) = \sum_{i \in G} \left| c_i - \frac{\sum_{j \in G} c_j}{|G|} \right|. \qquad (1)$$

If involving the current data in the group causes the $dev(G)$ to exceed a pre-defined threshold, then the data does not meet the similarity requirement and cannot be added to the group. Once a data point cannot be involved, the group is closed, and an empty group is created. Since the data in the same group

TABLE I
THEORETICAL SIMILARITY THRESHOLD $dev(G)$ FOR ACCURACY IMPROVEMENT UNDER VARYING GROUP LENGTHS n AND PRIVACY BUDGETS ϵ. SMALLER n OR LARGER ϵ DEMAND HIGHER $dev(G)$.

n	2	20	40	60	80	100
$\epsilon = 0.4$	1.96	3.59	4.50	5.16	5.69	6.14
$\epsilon = 0.6$	1.31	2.39	3.00	3.44	3.79	4.09
$\epsilon = 0.8$	0.98	1.79	2.25	2.58	2.84	3.07
$\epsilon = 1.0$	0.78	1.43	1.80	2.06	2.28	2.46

must come from consecutive timestamps, we refer to this as the continuous grouping method.

To satisfy DP, the entire process is controlled through Sparse Vector Technique (SVT) [32], which adds noise to both $dev(G)$ and the threshold, preventing privacy leakage from the grouping results. Note that if a data point cannot be involved in the current group, its noisy result should be directly released, as shown in Line 9 in Algorithm 1. The reason is that it has already consumed the privacy budget in the previous group's SVT process, and participating in the new group would result in additional privacy leakage.

Each data added to the group is post-processed using the noisy data in the group already released. This process is carried out by a SMOOTHER function, which has several alternatives, e.g., AverageSmoother, which estimates the current data by averaging all the noisy data in the group, MedianSmoother, which uses the median noisy data in the group to approximate the current data. Chen et al. prove that when the following condition is met,

$$dev(G) \leq \frac{\sqrt{2(n - \ln n - 1)}}{1 + \ln(n-1)\epsilon_p} \tag{2}$$

where n is the number of streaming data in the group and ϵ_p is the privacy budget used for adding noise, the group-based post-processing framework can obtain the accuracy improvement [9, Theorem 3.6].

Observation and Inspiration. Based on the analysis of the group-based post-processing framework, we identify some challenges that are difficult to overcome in real-time settings, which may cause accuracy disasters.

Challenge 1. The condition for accuracy improvement through post-processing is difficult to satisfy, as occasional changes in the stream can result in groups being constantly closed. We apply some commonly used privacy parameters and various group lengths to calculate $dev(G)$, and the results are presented in Table I. We observe that a smaller group length or larger privacy budget demand higher data similarity, e.g., with $\epsilon_p = 1$, accuracy can only be improved when each group consists of at least 100 data, and their $dev(G)$ can not exceed 2.46, regardless of the domain size.

Challenge 2. Each newly involved data can only be smoothed with data added to the group before the current timestamp. This implies that the data earliest involved in each group inevitably cannot benefit from post-processing, as Equation 2 cannot be satisfied. Therefore, ensuring a sufficient number of data in each group is crucial to improve the accuracy of the results, but solely increasing the length of each group is still not enough.

Allowing for a delay time provides us with opportunities to address these issues. To *challenge 1*, the delay time can provide a buffer period before closing the group, alleviating the issue of premature closure of the group due to fluctuations in the stream. To *challenge 2*, data added to the group can be post-processed in delayed batches, allowing data in the group to equally obtain the accuracy improvement. *Challenge 2* is also constrained by *Challenge 1*, since the accuracy improvement effect of delayed post-processing on data still depends on the length of the group. Besides, these benefits brought by delay time can also help develop other more accurate and stable data-releasing approaches.

III. OVERVIEW OF OUR APPROACH

This paper develops the group-based post-processing framework into a general privacy-preserving and delay-allowed data-releasing framework. Our framework consists of three parts: *Optimization strategy module*, *Noise addition module*, and *Post-processing module*. We present the overview of this framework in Figure 1.

A. Optimization Strategy Module

As streaming data is continuously input into the framework, there is always w timestamps gap between the release timestamp and the current timestamp. The data from these w timestamps is initially processed in the optimization strategy module. The module analyzes and records data features with an optimization strategy to assist in subsequent noise addition and post-processing steps. Here, we refer to the strategies as optimization strategies since the strategies designed in this module aim to enhance the accuracy of the released results.

Specifically, we design two kinds of optimization strategies: one leveraging the similarity between the streaming data and the other focusing on their orders.

- **Group-based.** We preserve the continuous grouping approach in the original group-based post-processing framework and address the aforementioned *challenge 2* in subsequent noise addition and post-processing modules. To overcome the challenge of *challenge 1*, we propose a discontinuous grouping strategy, which mitigates the hindrance of small group lengths to accuracy improvement by delaying group closure and allowing discontinuous grouping.
- **Order-based.** Recognizing the significance of preserving the order relationship in the released results for the utility, we propose entirely new strategies based on orders. These include one that compares the current data with all w data in the delay time, recording the comparison results to guide the post-processing of noisy data. The other strategy employs buckets to aggregate streaming data with a similar order for noise addition and post-processing, where the 'bucket' is partitioned based on fixed intervals in the data domain.

We present the algorithmic details of all strategies, rigorous privacy, and utility proofs in Section IV.

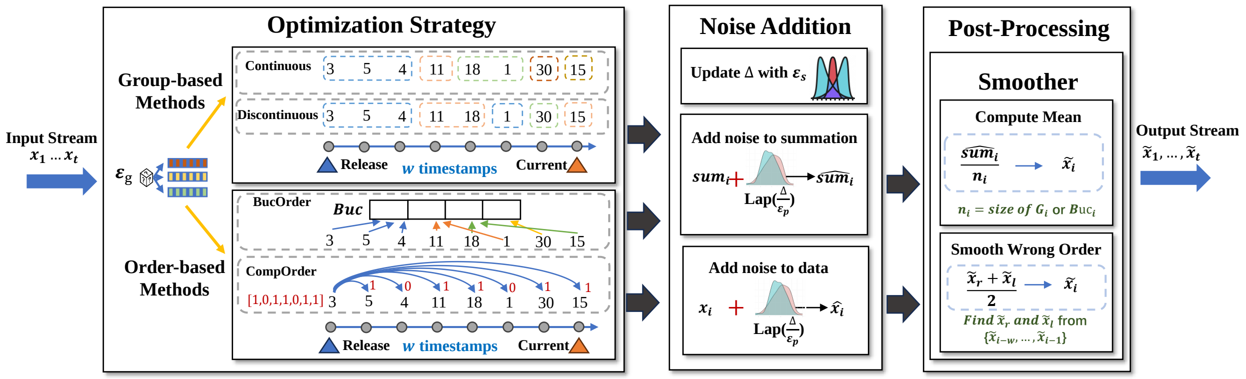

Fig. 1. Overview of the privacy-preserving and delay-allowed data releasing framework. This framework identifies the optimal strategy for the input stream, then introduces noise to comply with differential privacy (DP) standards, and finally post-processes the noised data stream before its release.

TABLE II
SUMMARY OF OPTIMIZED INSTANTIATIONS UNDER THE FRAMEWORK. FOUR DESIGNS GUIDE NOISE ADDITION AND POST-PROCESSING. METHODS OTHER THAN COMPORDER CAN ADD NOISE TO THE DATA SUMMATION TO REDUCE THE NOISE AMOUNT.

Strategies	Noise Addition	Post Processing	Type
ContinGroup	Add to data summation	Compute mean	Batch
DiscontinGroup	Add to data summation	Compute mean	Batch
BucOrder	Add to data summation	Compute mean	Batch
CompOrder	Add to single data point	Align results with order	Sliding Window

B. Noise Addition Module

The noise addition module is responsible for adding noise to the streaming data, which encompasses a mechanism for evaluating and updating data sensitivity, along with two methods for adding noise to the data.

We discussed earlier that reducing data sensitivity is one of the ways to improve the accuracy of released results under the event-level privacy setting. Estimating a reasonable data sensitivity for a data stream remains challenging in the context of the delay-allowed setting. We provide a detailed solution to this problem in Section VI.

In addition to the conventional approach of adding noise to each streaming data, the delay time provides us with the opportunity to leverage the similarity of data within the same group or bucket to reduce the amount of added noise. While this introduces additional bias as a cost, it can still achieve significant accuracy improvement when the group or bucket size is large, and data similarity is high.

C. Post-Processing Module

After noise addition, the noisy data is transmitted to the post-processing module, which incorporates a smoother function that integrates multiple smoothing methods for noisy data.

For group-based optimization strategies, noise is added to the summation of all data within the same group or bucket. The smoother function computes the mean or median of the noisy summation as the final noise result for each timestamp contained in the group or bucket. In this paper, we default to using the mean in the smoother function.

Unlike other methods, CompOrder adds noise to each data point individually and requires a special smoothing operation. Since the w timestamps before the release timestamp all record a comparison result between their data and the data at the release timestamp, the smoother calculates the minimum noisy result among the w data that should be greater than the releasing data and the maximum noisy result among the w data that should be smaller than the releasing data. If the data at the release timestamp is not between them, it takes the midpoint of the two values as the result of the release.

D. Privacy Budget Allocation

As a privacy-preserving data-releasing framework, it is imperative to ensure that none of the three modules violate the DP guarantee. Among these modules, the post-processing module exclusively processes noisy data and does not interact with the raw data, thereby avoiding any privacy disclosure. The privacy budget ϵ is divided into two parts: ϵ_g is allocated for capturing the characteristics of the data stream, and ϵ_p is reserved for adding noise to the data for release. If there is an update to the data sensitivity in the noise addition module, a portion of the privacy budget ϵ_s needs to be allocated from ϵ_p. Since this paper focuses on strategies design within each module, we utilize the standard sequential composition theorem [33]. As long as $\epsilon_g + \epsilon_s + \epsilon_p = \epsilon$, the entire framework satisfies ϵ-DP. Other composition theorems such as advanced composition [19] and zCDP [8] can also be applied in practical deployment to compose a smaller total privacy budget.

We summarized all approaches that can be instantiated under the framework in Table II.

IV. TECHNICAL DETAILS OF OPTIMIZATION STRATEGY

In this section, we dive into the details of the optimization strategies. We present group-based and order-based approaches, respectively.

A. Group-Based Methods

The core idea of the group-based strategy is to aggregate similar data into a group and then utilize the data similarity to reduce the noise amount or smooth the added noise. We introduce two group-based methods in this part: the continuous grouping method and the discontinuous grouping method. The continuous grouping method follows the group-based post-processing framework proposed in PeGaSuS, but involves some differences in the details since the data is processed in batches in our setting. The discontinuous grouping method is an optimized group-based method to further leverage the advantages introduced by a delay time.

Continuous Grouping. The continuous grouping method maintains an open group, with the algorithm continuously judging the similarity between the streaming data at the current timestamp and the data in the group to decide whether to involve it in the group. If the current data cannot be involved in the current group, it is treated as an independent group, and the current group is closed. Note that treating the current data as an independent group ensures the algorithm does not violate the DP guarantee. In the delay-allowed setting, this continuous grouping method does not undergo significant changes. We defer the details of the grouping process within each delay time (batch) to Algorithm 6 in Appendix A-B. The primary distinction is that, unlike the real-time setting where noise is added or post-processing is performed once a data point is involved in a group, these operations can take place after a group is closed in the delayed setting, which can provide benefits in optimizing noise addition and improving post-processing accuracy.

A noteworthy detail in the continuous grouping algorithm presented in this paper is that the current group is always closed at the end of each batch, and a new group is initiated at the beginning of the next batch. An alternative approach could retain the last group at the end of a batch. If the group is still open, it can continue to absorb new streaming data at upcoming timestamps in the next batch. This has the potential to enhance the effectiveness of smoothing for groups spanning two consecutive batches. However, it is necessary to first process the data currently involved in the group to avoid exceeding the allowed delay before allowing the processed data to continue being involved in the processing of subsequent data within the same group. Therefore, the accuracy improvement of the first processed data is still limited to the current data batch. Considering the introduced computational complexity, we do not incorporate this alternative approach in the continuous grouping strategy.

Discontinuous Grouping. In terms of enhancing post-processing effectiveness, processing the batched data with continuous grouping method can only partially addresses the *challenge 2* mentioned in Section II-C, that is ensuring that data within the same group obtains the same benefits from the post processing. The problem of insufficient accuracy improvement due to short group length persists. Therefore, we propose a discontinuous grouping strategy.

Differing from the continuous grouping method, discontinuous grouping allows for the simultaneous maintenance of multiple open groups. For each streaming data, the algorithm traverses through all current groups until a suitable group is found for inclusion. If none of the current groups can involve the current data, then the algorithm creates a new group and adds the current data to it. In this way, the length of non-continuous grouping is no longer affected by data fluctuations. The larger the delay time, the greater the advantage of discontinuous grouping strategy. The algorithm for discontinuous grouping in a batch is presented in Algorithm 2.

Algorithm 2 Discontinuous Grouping in A Data Batch.

```
Input: A data batch $B_i = x_{i\cdot w}, ..., x_{(i+1)\cdot w-1}$, privacy budget $\epsilon_g$
    for grouping, threshold $\theta$.
Output: $\mathcal{G} = \{G_1, G_2, ...\}$
 1: $\mathcal{G} \leftarrow \emptyset, c \leftarrow 1, \tilde{\theta}_1 \leftarrow \theta + \mathsf{Lap}\,(2 \cdot (2w-1)\Delta_s/\epsilon_g)$
 2: for $x_i \in B_i$ do
 3:     Flag $\leftarrow 1$.
 4:     for $G_j \in \mathcal{G}$ do
 5:         if $dev\,(G_j \cup x_i) + \mathsf{Lap}\,(4 \cdot (2w-1)\Delta_s/\epsilon_g) < \tilde{\theta}_j$ then
 6:             Involve $x_i$ into $G_j$.
 7:             Flag $\leftarrow 0$.
 8:             Break.
 9:         end if
10:     end for
11:     if Flag $== 1$ or $i == 1$ then
12:         Create $G_c \leftarrow \emptyset$, involve $x_i$ in $G_c$, add $G_c$ into $\mathcal{G}$
13:         $\tilde{\theta}_c \leftarrow \theta + \mathsf{Lap}\,(2 \cdot (2w-1)\Delta_s/\epsilon_g)$, $c \leftarrow c+1$.
14:     end if
15: end for
16: return $\mathcal{G} = \{G_1, G_2, ...\}$
```

For discontinuous grouping, it is not advisable to keep all groups from the previous batch open when moving to the next batch. Since discontinuous grouping methods lack conditions for closing the groups, maintaining excessively long lengths of groups can result in an explosion of privacy budget.

B. Order-based Methods

Our designed group-based method can effectively address the challenges faced by the existing approaches in real-time scenarios when delay time is sufficient. However, a short delay time naturally limits the accuracy enhancement of group-based methods. To address this limitation, we design two strategies from a completely new perspective. Considering that order consistency is also a crucial aspect of the utility of released results, we design two non-strict order-preserving approaches.

CompOrder. The core idea of CompOrder is to continuously record pairwise comparisons between the data at the release timestamp and its subsequent w data within the delay time. At the same time, it can follow the comparison results recorded at the preceding w timestamps to adjust the noisy result at the release timestamp. All comparison processes need to adhere to DP protection. In this way, CompOrder can release the noisy stream with a non-strict order consistency compared with the raw data stream.

The detailed process of CompOrder is shown in Algorithm 3. In the setting where the delay time is allowed to be w timestamps, we have visibility into the data for the subsequent w timestamps when processing and releasing streaming data at i^{th} timestamp. The algorithm utilizes SVT to compare the data at i^{th} timestamp with the data at the next w timestamps, recording the order relationships in a one-dimensional array OrderList of length w. Specifically, for all $j \in [i+1,\ i+w]$,

Algorithm 3 CompOrder.

```
Input: Stream $\{x_1, x_2, ...\}$, privacy budget $\epsilon_g$ for recording orders.
Output: OrderList
 1: OrderList $\leftarrow \emptyset$, $\rho \leftarrow \mathsf{Lap}(4\Delta_s/\epsilon_g)$
 2: for $i \leftarrow 1, 2, \ldots$ do
 3:     for $j \in [i+1, i+w]$ do
 4:         if $x_i - x_j + \mathsf{Lap}(8w\Delta_s/\epsilon_g) > \rho$ then
 5:             Add 0 into OrderList[$i$].
 6:         else
 7:             Add 1 into OrderList[$i$].
 8:         end if
 9:     end for
10: end for
11: return OrderList
```

Algorithm 4 Post-processing for CompOrder.

```
Input: Noisy stream $\{\hat{x}_1, \hat{x}_2, ...\}$, OrderList.
Output: $\{\tilde{x}_1, ..., \tilde{x}_t\}$
 1: for $i \leftarrow 1, 2, \ldots$ do
 2:     for $j \in [i-w, i-1]$ do
 3:         Finds $\tilde{x}_l$ with OrderList[$j$][$i-j-1$] $= 0$.
 4:         Finds $\tilde{x}_r$ with OrderList[$j$][$i-j-1$] $= 1$.
 5:         $\tilde{x}_l, \tilde{x}_r \in \{\tilde{x}_{i-w}, ..., \tilde{x}_{i-1}\}$
 6:         if $\hat{x}_i < \tilde{x}_l$ or $\hat{x}_i > \tilde{x}_r$ then
 7:             $\tilde{x}_i \leftarrow (\tilde{x}_l + \tilde{x}_r)/2$.
 8:         else
 9:             $\tilde{x}_i \leftarrow \hat{x}_i$.
10:         end if
11:     end for
12: end for
13: return $\{\tilde{x}_1, ..., \tilde{x}_t\}$
```

if $x_i > x_j$, then 0 is recorded in OrderList; otherwise, 1 is recorded. Therefore, except for the initial w timestamps in the data stream, each timestamp's preceding w timestamps have recorded the order relationship with the data at that timestamp.

The post-processing method we designed for CompOrder is shown in Algorithm 4. The algorithm finds the minimum value $\tilde{x}_r$ among the noisy released data at the preceding w timestamps that should be greater than the current timestamp and the maximum value $\tilde{x}_l$ among w noisy released data that should be less than the current timestamp. Then we check whether the current noisy data falls between $\tilde{x}_l$ and $\tilde{x}_r$. If it does not satisfy this condition, we adjust the current noisy data with $(\tilde{x}_r + \tilde{x}_l)/2$ to release.

BucOrder. CompOrder acquires order information by continuously comparing the order between two streaming data, consuming a large amount of privacy budget. We propose a new order-based approach BucOrder that obtains the order information by partitioning the data domain and randomizing the partition in which the data is located. BucOrder processes the data in each delay time (batch) separately and only ensures the orders of the data within one batch.

The detailed algorithm is shown in Algorithm 5. First, the data domain is divided into $n_b \leftarrow \lceil |D|/m \rceil$ buckets of length m. Each data is then mapped to the corresponding bucket based on the range to which it belongs. Then, the algorithm randomizes the bucket mapped by each data using the GRR mechanism [41], that is each data remains in its original bucket with a probability $p = e^{\epsilon_g}/(e^{\epsilon_g} + n - 1)$, and randomly maps

Algorithm 5 BucOrder in A Data Batch.

```
Input: A data batch $B_i = x_{i\cdot w}, ..., x_{(i+1)\cdot w-1}$, $m$ is the size of
       data domain covered by each bucket, privacy budget $\epsilon_g$ and $\epsilon_p$
       for sorting and adding noise, respectively.
Output: $\{Buc_1, ..., Buc_{n_b}\}$
 1: Compute the number of the buckets $n_b \leftarrow \lceil |D|/m \rceil$
 2: Divide the data domain into $n_b$ buckets: $\{Buc_1, ..., Buc_{n_b}\}$.
 3: Map $B_i$ into $n$ buckets, $p \leftarrow e^{\epsilon_g}/(e^{\epsilon_g} + n_b - 1)$
 4: for $x_i \in B_i$ do
 5:     Map $x_i$ into the $\lceil x_i/m \rceil^{th}$ bucket.
 6:     $b \leftarrow \mathsf{Ber}(p)$
 7:     if $b = 1$ then
 8:         $x_i$ stays in the current bucket.
 9:     else
10:         Put $x_i$ into an Uniformly and Randomly selected bucket.
11:     end if
12: end for
13: return $\{Buc_1, ..., Buc_{n_b}\}$
```

into other buckets with a probability of $1 - p$.

The noise addition and post-processing for the data in each bucket are the same as those in group-based approaches. The bias introduced during this process is mainly controlled by the size of the bucket m, since the data mapped to the same bucket differs by at most m with the probability p. Besides, we can add an additional post-processing step to further improve the accuracy after post-processing. As the ranges of buckets are ordered, if the mean of the noisy data in a bucket is less than the lower boundary of the bucket, it is processed to be the value lower boundary. For example, if the mean of noisy data in the j^{th} bucket is smaller than its low boundary $j \times m$, then the summation of noisy data in this bucket is set to $j \cdot m \cdot Buc_j$.

V. Theoretical Analysis for Our Approaches

In this section, we analyze the privacy and accuracy of the all approaches that apply different optimization strategies, as shown in Table II.

A. Privacy Analysis.

Firstly, we analyze that all four approaches satisfy ϵ-DP privacy guarantee. We focus on analyzing the privacy guarantee in the optimization strategy module since the noise addition module provides $(\epsilon_s + \epsilon_p)$-DP privacy guarantee through noise sampling and the post-processing module does not consume any privacy budget.

Privacy of Continuous Grouping. In the continuous grouping method, each streaming data only participates in one SVT process when grouping, which provides a strict ϵ_g-DP privacy guarantee [32, Theorem 2].

Privacy of Discontinuous Grouping. The calculation of privacy budget consumption for the discontinuous grouping method is more complex than for continuous grouping. According to the privacy guarantee provided by SVT [32, Theorem 2], every time a data point does not meet the condition for joining a group, it consumes the privacy budget. The continuous grouping method concludes the current group when encountering non-satisfying data, so each timestamp incurs privacy budget consumption at most once. However, in the discontinuous grouping, each data participates in multiple SVT judgments, leading to multiple compositions of privacy budget consumption. We analyze the privacy guarantee provided by Algorithm 2 in Theorem 1.

Theorem 1 *Discontinuous grouping strategy satisfies ϵ_g-DP.*

Proof *(Sketch) We assume that the stream D and D' differ by the data at the i^{th} timestamp. $\mathcal{G}_i$ represents the grouping status at i^{th} timestamp. The probability distribution of group states $\mathcal{G}_i$ at the $i-1$ timestamps preceding the i^{th} timestamp are all identical. From the i^{th} timestamp onwards, the probability distribution of group states at each timestamp may have difference between D and D'. According to the privacy guarantee provided by SVT, we have*

$$\begin{aligned}&\frac{\Pr[M(\mathcal{G}'_{w-1}, x_w) = \mathcal{G}]}{\Pr[M(\mathcal{G}_{w-1}, x_w) = \mathcal{G}]}\\ =&\frac{\Pr[M(\mathcal{G}_{i-1}, x'_i) = \mathcal{G}'_i] \cdot \prod_{j=i+1}^{w} \Pr[M(\mathcal{G}'_{j-1}, x_j) = \mathcal{G}'_j]}{\Pr[M(\mathcal{G}_{i-1}, x_i) = \mathcal{G}_i] \cdot \prod_{j=i+1}^{w} \Pr[M(\mathcal{G}_{j-1}, x_j) = \mathcal{G}_j]}\\ \le& e^{\epsilon_g} \cdot \frac{\prod_{j=i+1}^{w} \Pr[M(\mathcal{G}'_{j-1}, x_j) = \mathcal{G}'_j]}{\prod_{j=i+1}^{w} \Pr[M(\mathcal{G}_{j-1}, x_j) = \mathcal{G}_j]} \le e^{(2w-1)\epsilon_g}\end{aligned}$$

Here, $M(\mathcal{G}_{j-1}, x_j)$ using the group set up to the previous timestamp and the current data as input, output the group set for the current timestamp. Therefore, when we provide $\epsilon_g/(2w-1)$ privacy guarantee for each SVT judgement, Algorithm 2 satisfies ϵ_g-DP.

The detailed proof is deferred to Appendix A-C. □

Privacy of CompOrder. Then, we analyze the privacy guarantee of CompOrder strategy in Theorem 2.

Theorem 2 *CompOrder strategy satisfies ϵ_g-DP.*

Proof *(Sketch) In Algorithm 3, recording the comparison results between the current data and the subsequent data within the delay time follows the SVT design [32]. To satisfy DP, we divide the privacy budget ϵ_g, which is used to record the order information, by 2 for the two SVT algorithms. Additionally, since the maximum possible number of positive answers for each SVT is w, ϵ_g needs to be divided by w. In this way, we ensure that CompOrder strategy satisfies ϵ_g-DP. The detailed proof is deferred to Appendix A-E.* □

Privacy of BucOrder. Finally, we prove the privacy of the BucOrder strategy in Theorem 3.

Theorem 3 *BucOrder strategy satisfies ϵ_g-DP.*

Proof *In Algorithm 5, using the GRR mechanism to randomly flip the bucket mapped by data satisfies $\frac{e^{\epsilon_g}/(e^{\epsilon_g}+n_b-1)}{1/(e^{\epsilon_g}+n_b-1)} = \epsilon_g$. Therefore, BucOrder strategy satisfies ϵ_g-DP.* □

B. Error Analysis

Next, we analyze the error of the noisy results released by all approaches, respectively.

Error of Continuous and Discontinuous Grouping. Firstly, we theoretically analyze the accuracy of the results released by the group-based approaches in Theorem 4.

Theorem 4 *The expected squared error of group-based methods within a delay time is bounded by $n_g \cdot 2\frac{|d|^2}{\epsilon_p^2} + (1-\beta)w\theta^2 + \beta w\left(\theta + \frac{4\log(2/(2-\beta))}{\epsilon_g}\right)^2$, where n_g is the number of groups, $|d|$ is the size of the data domain, and $\beta \in [0,1]$ is the probability of SVT incorrectly adding data above the threshold into a group.*

Proof *(Sketch) The error in a released data batch comes from Laplace noise, the bias introduced by approximating data within a group with their mean, and the error introduced by SVT for grouping. Denote n_g as the number of groups in a delay time, β is the failure probability of SVT, and $m_{>\theta}$ is the approximate bias in the data added to a group due to the misjudgment of SVT. We have*

$$\mathbb{E}\left[(\hat{B}_i - B_i)^2\right] = n_g \cdot 2\frac{|d|^2}{\epsilon_p^2} + (1-\beta)w\theta^2 + \beta w(m_{>\theta})^2$$

We first need to obtain the privacy guarantee of SVT. There are no existing conclusions can be directly applied to our current computation. We prove that the probability of $Dev(G_i \cup x_i) \ge \theta + \alpha$ is at most β when $\frac{\log(2/\beta)}{\epsilon_g} \le \alpha \le \frac{4\log(2/(2-\beta))}{\epsilon_g}$. Based on this conclusion, we have:

$$\begin{aligned}\mathbb{E}\left[(\hat{B}_i - B_i)^2\right] \le n_g \cdot 2\frac{|d|^2}{\epsilon_p^2} &+ (1-\beta)w\theta^2\\ &+ \beta w\left(\theta + \frac{4\log(2/(2-\beta))}{\epsilon_g}\right)^2\end{aligned}$$

The detailed proof is deferred to Appendix A-D. □

The conclusion in Theorem 4 applies to both continuous and discontinuous grouping approaches. When the same θ is set, n_g of the continuous grouping can be greater than discontinuous grouping. Besides, ϵ_g should be $\frac{\epsilon_g}{(2w-1)}$ in discontinuous grouping due to the multiple involvements of each data in SVT.

If we multiply the error bound in Theorem 4 by t/w, the error bound of the data stream released up to the t^{th} timestamp is approximately $O\left(\frac{t}{w} \cdot \frac{2|d|^2}{\epsilon_p^2} + t\theta^2\right)$. As w decreases, it approaches the naïve method. This indicates that the length of the delay time has a significant impact on the accuracy of the group-based methods.

Error of CompOrder. In CompOrder, the distribution of the data stream, such as the order and distance between streaming data, directly influences the accuracy of comparisons and the bias introduced by post-processing. Moreover, the accuracy of SVT indirectly affects the calculation of x_l and x_r, which in turn impacts the bias introduced by post-processing. All these factors combined increase the difficulty in deriving the error bound for the released results. As a compromise, we provide a lower bound of the probability that the noisy result of any $\hat{x}_i$ remains in the correct order position.

Theorem 5 *In CompOrder, the lower bound on the probability that the post-processed noisy data $\hat{x}_i$ maintains the same order as the noisy results corresponding to the two timestamps whose data's true orders are on either side of the x_i within preceding w timestamps is $(1-2e^{-\epsilon_g a/16})^2\left(1-\frac{3}{4}e^{-\frac{a\Delta_s}{\epsilon_p}}\right)^w$, where a represents the assumed minimum distance between adjacent ordered streaming data.*

Proof *The proof is deferred to Appendix A-F.* □

The impact of delay time's length on the accuracy of CompOrder is not linear. Specifically, for x_l and x_r among the preceding w timestamps of x_i, if extending w by k includes new streaming data $\{x_{i-w-k}\dots x_{i-w}\}$ that lie out of $[x_l, x_r]$, it has no impact on the accuracy of post-processed x_i. Besides, when w is large, the privacy budget allocated to each comparison decreases, which may even lead to a reduction in the overall accuracy of the released results. Therefore, CompOrder significantly reduces the impact of the length of delay time on the accuracy of the released results compared to the group-based approaches. Inevitably, post-processing the noisy data to follow the noisy comparison results may involve a large computational complexity and result in a continuously accumulating error backward.

Error of BucOrder. Finally, we provide the expected squared error of the BucOrder within a data batch in Theorem 6.

Theorem 6 *The expected squared error of BucOrder within a delay time is $n_b\cdot 2\frac{|d|^2}{\epsilon_p^2}+w\cdot\frac{m^2+(n_b-1)|d|^2/9}{e^{\epsilon_g}+n_b-1}$, where n_b is the number of the buckets, m is the size of domain covered by each bucket, $|d|$ is the size of the data domain.*

Proof *(Sketch) The error in a released data batch comes from the Laplace noise, the bias introduced by approximating data within a bucket with their mean, and the error introduced by GRR mechanism for mapping the data into the buckets. Denote n_b as the number of buckets in a delay time, $p=\frac{e^{\epsilon_g}}{(e^{\epsilon_g}+n_b-1)}$ is the probability that a data is in the right bucket, and m_{out} is the approximate bias introduced by data randomized to the wrong bucket by GRR.*

$$\mathbb{E}\left[(\hat{B}_i-B_i)^2\right]=n_b\cdot 2\frac{|d|^2}{\epsilon_p^2}+w\cdot\frac{m^2+(n_b-1)m_{out}^2}{e^{\epsilon_g}+n_b-1}$$

Since the GRR uniformly maps data to other buckets with a probability $\frac{1}{e^{\epsilon_g}+n_b-1}$, it's difficult to constrain m_{out}. We calculate the approximate expected value of m_{out} is $\frac{|d|}{3}$. Substituting the approximate expected m_{out} into the equation, we get

$$\mathbb{E}\left[(\hat{B}_i-B_i)^2\right]=n_b\cdot 2\frac{|d|^2}{\epsilon_p^2}+w\cdot\frac{m^2+(n_b-1)|d|^2/9}{e^{\epsilon_g}+n_b-1}$$

The detailed proof is deferred to the Appendix A-G. □

VI. Optimization with truncating data sensitivity

In this section, we delve into calculating an optimized data sensitivity, which can not only aids in adding noise to data streams with unknown data domains but also helps prevent the introduction of unnecessary noise due to the potential wide range of variations in streams.

Calculation of the Data Sensitivity. Firstly, we address how to find the optimal sensitivity. In the practical data streams, usually only a small number of data are distributed at the extremes of the data domain. We need to choose an optimal sensitivity based on the actual data distribution to truncate excessively large data, thereby reducing the amount of introduced noise. We design a quality function in Equation 3 to score each possible sensitivity τ_i in the data domain, where higher scores indicate sensitivities that can bring greater accuracy improvements to the released results.

$$\begin{aligned}q(D,\tau_i)&=\text{Benefit}(\mathsf{Lap}(\cdot))-\text{Loss}(\text{Bias}(D_{>\tau}))\\&=\left(\sqrt{2}m\cdot\frac{|d|-\tau_i}{\epsilon_p}-\sum_{v\in D_{>\tau}}(v-\tau_i)\right)\Bigg/(m\cdot|D|)\end{aligned}\tag{3}$$

Specifically, $\text{Benefit}(\mathsf{Lap}(\cdot))$ represents the decrease in error introduced by Laplace noise when reducing the sensitivity to τ_i, and $\text{Loss}(\text{Bias}(D_{>\tau}))$ represents the additional bias introduced by truncating streaming data greater than τ_i. To avoid the fluctuation range of scores being highly correlated with the actual data domain when varying τ, we divide the scoring results by $m\cdot|D|$ to normalize the sensitivity of the scoring function to 1.

If we directly choose the τ_i with the highest score as data sensitivity, it would leak the privacy of the raw streaming data. Therefore, we use the Report Noisy Max [18] to sample the optimal τ_i under DP protection. This algorithm adds independently generated Laplace noise $\mathsf{Lap}(1/\epsilon_s)$ to each score obtained from Equation 3 and return the score with the largest noisy value, where ϵ_s is the privacy budget used for updating the data sensitivity. The Report Noisy Max is proved to satisfies ϵ-DP guarantee in [18].

Data Sampling and Update Frequency. After addressing the issue of calculating the optimal data sensitivity, another important problem to discuss is when and with which data to calculate the data sensitivity. In addition to CompOrder, all other approaches proposed in this paper treat the streaming data within the delay time as a batch, and we can also update the data sensitivity in the batch. Although the strategy of CompOrder processes the data in the form of a sliding window, we can also update the data sensitivity based on batch. In a batch, the privacy budget for each streaming data has already been divided into ϵ_g for the optimization strategy and ϵ_p for adding noise. We can further allocate ϵ_s from ϵ_p for data sensitivity calculation. Considering that in reality, although streaming data fluctuates over time, it rarely undergoes significant shifts in the range of data fluctuations in a short period. Therefore, we can update data sensitivity at intervals of several batches to save the privacy budget.

For group-based approaches and BucOrder, there is an optimization strategy that can further save privacy budgets by utilizing the groups or buckets obtained from the optimization strategy module. Since most of the streaming data in the same group or in the same bucket fluctuates within the same range, we can also sample p percent representative data from each group or bucket for data sensitivity calculation. The sampled data and the remaining data used to calculate the released results can both consume the entire ϵ_p. Regardless of the noise addition and post-processing method, we can use the remaining data in the group or bucket to calculate the noisy released results. For example, for the summation noise addition method, the released results for all data in the group or bucket can be calculated as:

$$\frac{\left(\sum_{j \in G_r[i]} x_j\right) + \mathsf{Lap}\left(\Delta_s/\epsilon_p\right)}{|G_r|i||}$$

where $G_r[i]$ represents the group after sampling, and Δ_s is the current data sensitivity.

VII. Evaluation

In this section, we design experiments to evaluate our proposed approaches. The evaluation focuses on three aspects: (1) Allowing a delay time can bring about how much accuracy improvement in the streaming data releasing tasks; (2) How different designs of the three modules in the framework lead to differences in accuracy; (3) How key parameters affect the accuracy of the proposed approaches. Toward these goals, we conduct experiments on real-world data streams. Note that the source codes of PeGaSuS have not been formally released, we implemented it with our best effort and included it for comparison purposes.

A. Setup

Datasets. We run experiments on the following datasets:

- COVID19 DEATH [2]. It is collected by The National Center for Health Statistics (NCHS). It records the number of deaths per week occurring in the United States from 2019 to 2023. The streaming data ranges from 0 to $25,974$.
- Unemployment [1]. It records the Unemployment Level of 16-19 Yrs., Black or African American per month from 1972 to 2023. The streaming data ranges from 72 to 612.
- Outpatient [4]. It is collected through the U.S. Outpatient Influenza-like Illness Surveillance Network (ILINET). It contains patient information for each week from 1997 to 2023. We select the 'TOTAL PATIENTS' attribute for evaluation. The streaming data ranges from $27,263$ to $2,260,794$.
- FoodMart [3]. It contains customer transactions from a retail store. $18,319$ records are contained and the streaming data ranges from 1 to 1559.

Competitors. Three baselines are involved in our experiments.

- Naïve. Adding Laplace noise satisfying ϵ-DP to each streaming data before releasing.
- PeGaSus [9], PeGaSus_Delay. A widely used group-based post-processing framework and its variant. The variant benefits from the delay time without changing the original design of the framework, enables the post-processing of noisy data for an entire group.
- DPI [26]. A state-of-the-art method for responding to stream queries and publishing under the user-level privacy setting.
- Adapub [40]. An existing method for stream releasing under the w-event-level privacy setting.

To evaluate the different designs within each module, besides the methods under the proposed framework as shown in Table II, experiments also include a comparison with methods involving adding noise to the single data point. Three group-based approaches: Contin_reduce uses the continuous grouping strategy and adds noise to the summation of the entire group; Discontin_pp and Discontin_reduce use the discontinuous grouping strategy and add noise to the data independently and to the summation of the entire group, respectively. Two order-based approaches: CompOrder and BucOrder. BucOrder adds noise to the summation of the data in each bucket.

Metrics. In our experiments, we use Mean Absolute Error (MAE) as the accuracy metric, measuring the difference between released results and the actual data streams. In particular, for released stream prefix $\hat{D}_t$ at t^{th} timestamp we measure

$$MAE(\hat{D}_t) = \frac{1}{|D_t|} \sum_{x_i \in D_t} |\hat{x}_i - x_i|.$$

All results in experiments are averaged with 20 repeats.

Environment. All the approaches are implemented using Python 3.7.11 and numpy 1.24.3 libraries. Experiments are carried out on servers running Ubuntu 22.04.1, equipped with E5-2620 v4 2.10GHz processors and 128 GB of memory.

B. Analysis of Experimental Results

The Accuracy Improvement of the Delay-allowed Approaches. In Figure 2 and Figure 3, we respectively show the accuracy improvement of the delay-allowed approaches compared to the real-time releasing approaches when the length of the delay time $w = 10$ and $w = 100$. Firstly, we can observe that allowing a delay of only 10 timestamps can bring about a substantial increase in accuracy for the released streams. Especially for the Outpatient dataset, which contains 1366 timestamps, BucOrder can achieve more than 32-fold accuracy improvement compared to PeGaSuS and the naïve method when allowing a delay of 10 timestamps.

Secondly, the results in Figure 2 and Figure 3 both show that the performance of PeGaSuS is not improved compared to the naïve approach and is even lower than the naïve approach. Even when we apply a delay to PeGaSuS, conduct the post-processing of the results for the entire group together, or add noise to the summation to reduce the noise addition amount, it does not significantly improve the accuracy of the released results. This further confirms our theoretical analysis in Figure II-C. The discontinuous grouping strategy increases the length of each group, thus achieving more accuracy improvement. The advantage is not significant when the delay time is short, it can be seen that the accuracy of the discontinuous grouping strategy is much higher than PeGaSuS and approaches based on the same grouping strategy with PeGaSuS on all four datasets.

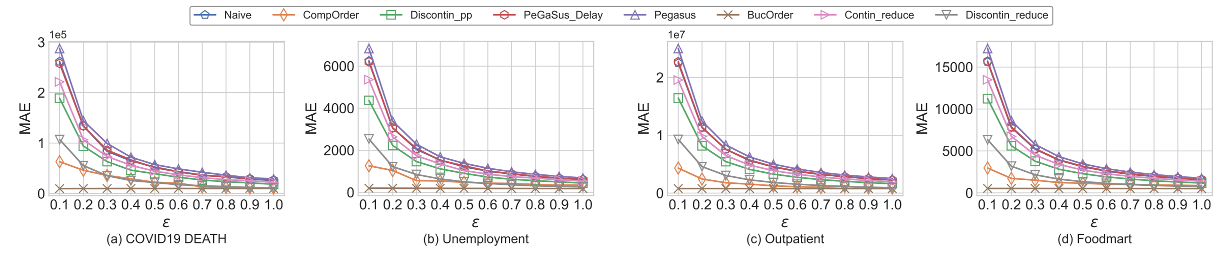
Fig. 2. Comparison of all methods on four dataset with varying ϵ, where delay time $w = 10$, and threshold $\theta = 3$. Each bucket size m of BucOrder in Outpatient dataset is $10,000$, and is 100 in other three datasets.

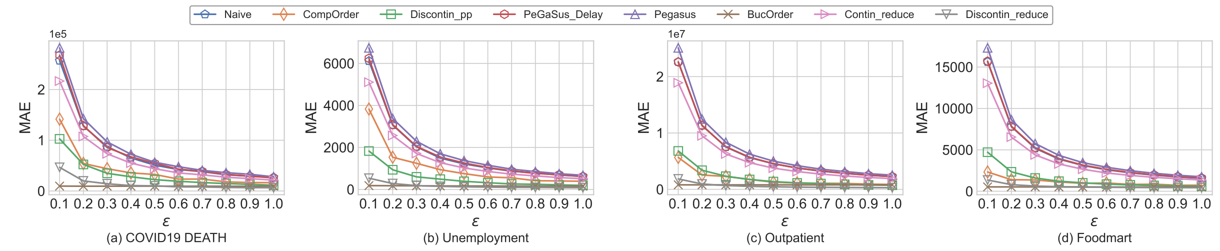
Fig. 3. Comparison of all methods on four dataset with varying ϵ, where delay time $w = 100$, and threshold $\theta = 3$. Each bucket size m of BucOrder in Outpatient dataset is $10,000$, and is 100 in other three datasets.

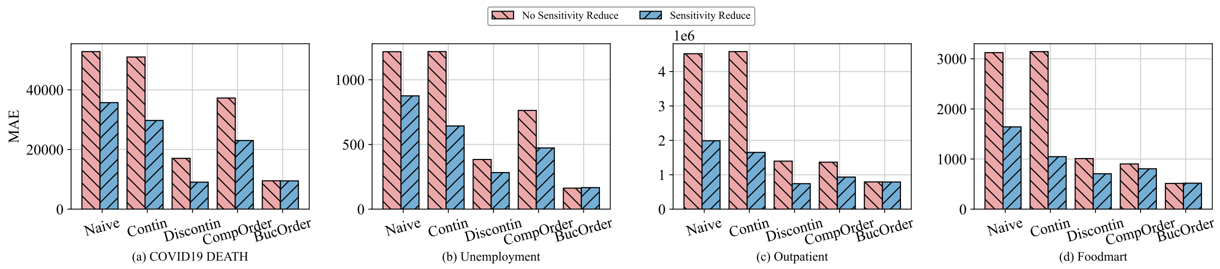
Fig. 4. The effectiveness of data sensitivity truncation algorithm for all methods on four dataset, where $\epsilon = 0.5$, $\theta = 3$, and $w = 100$. Each bucket size m of BucOrder in Outpatient dataset is $10,000$, and is 100 in the other three datasets. Both group-based approaches add noise to each data individually.

Thirdly, the advantage of the order-based approach is more pronounced when the delay time is short compared to the group-based approaches. BucOrder and CompOrder achieve the highest accuracy on all datasets when $w = 10$, and $\epsilon < 0.5$. However, as the delay time increases from 10 to 100, the accuracy of CompOrder decreases on the COVID19 DEATH and Unemployment. We think the reason is that a longer delay time helps improve the order consistency of the released streams, but at the same time, it means that each timestamp needs to be compared more times, reducing the privacy budget allocated to each comparison, resulting in low accuracy of the recorded comparison results. BucOrder effectively avoids this issue by providing privacy protection after sorting the results, rather than during sorting. At the same time, it reduces the dependence on the length of the delay time since it is based on mapping streaming data to buckets in the data domain. Additionally, note that we set the bucket size $w = 10,000$ on the Outpatient because the data domain of this stream is too large. A small bucket size would greatly reduce the computational efficiency of the BucOrder and the accuracy of the GRR mechanism used by BucOrder.

The Effectiveness of Data Sensitivity Truncation Algorithm. In Figure 4, we demonstrate the effect of adding data sensitivity truncation on the accuracy improvement of all approaches, including the naïve approach. To highlight the effect of data sensitivity truncation, we add noise independently for each data for both group-based and order-based approaches. We set the delay time $w = 100$ and update the data sensitivity every 5 delay times for all datasets. During the delay times when data sensitivity needs to be updated, we allocate half of privacy budget used for releasing the data $\epsilon_p/2$ to estimate the data sensitivity. To apply our data sensitivity truncation algorithm, we also introduce the same length of delay time in the naïve approach and update the data sensitivity as the same intervals as other approaches.

We can observe that the data sensitivity truncation algorithm has an improvement for all approaches (except BucOrder) on all datasets, especially on streams with large data domains such as Outpatient and Foodmart, i.e., the accuracy improvement of the continuous-based approach in the released results is approximately 3-fold on the Outpatient and Foodmart. Additionally, we find that the data sensitivity truncation algorithm

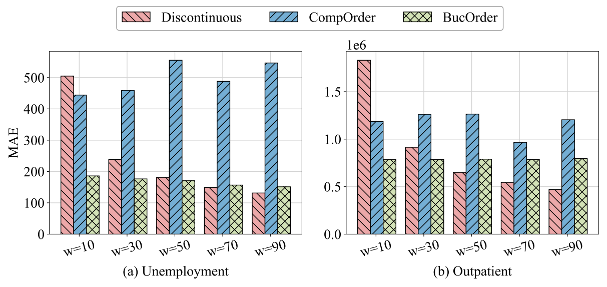
Fig. 5. Comparison of group-based method and order-based methods with varying the length of delay time w on Unemployment and Outpatient datasets, where $\epsilon = 0.5$, and $\theta = 3$. $m = 100$ and $10,000$ in BucOrder for Unemployment and Outpatient, respectively. The group-based approach adds noise to the summation of the data in the group.

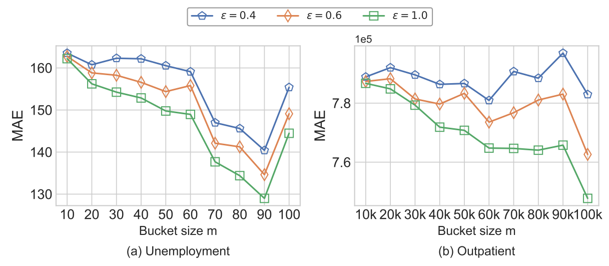
Fig. 6. Impact of each bucket size m on the accuracy of BucOrder under different privacy budget ϵ, where $w = 100$.

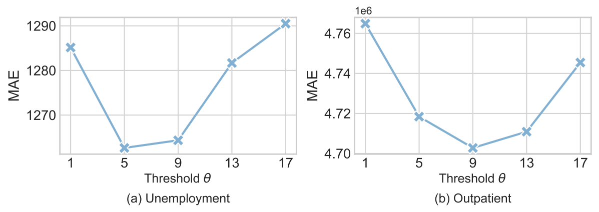
Fig. 7. Impact of threshold θ on the accuracy of the continuous group-based approach, where $\epsilon = 0.4$, and $w = 100$. The noise is added to the summation of the data in the group.

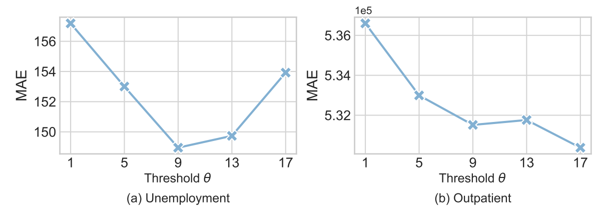
Fig. 8. Impact of threshold θ on the accuracy of the discontinuous group-based approach, where $\epsilon = 0.4$, and $w = 100$. The noise is added to the summation of the data in the group.

has almost no accuracy improvement for BucOrder. This is because BucOrder limits the bias within the same bucket by setting m much smaller than the domain size. The accuracy improvement brought by post-processing $\mathsf{Lap}\left(\frac{\Delta}{\epsilon_p}\right)$ noise is already close to the optimal accuracy improvement achievable by the data sensitivity truncation algorithm with adding $\mathsf{Lap}\left(\frac{2\Delta_s}{\epsilon_p}\right)$ noise.

Comparison of Group-based Approaches and Order-based Approaches. In Figure 5, we vary the length of the delay time to run one group-based approach and two order-based approaches. We aim to observe the impact of the delay time on these two different types of approaches and to make comparisons between them. We select the best-performing discontinuous grouping approach as a representative of group-based approach and apply noise addition to the summation of the entire group to reduce the noise.

Firstly, we can observe that the performance of the discontinuous approach improves as the length of the delay time increases. However, the order-based approaches do not show a clear pattern of accuracy variation with changes in the delay time. Additionally, we observe that when the delay time is short, order-based approaches have an advantage over group-based approaches. The same observation can also be obtained in Figure 2, and the advantage of CompOrder is more significant for data streams with faster fluctuations, such as COVID19 DEATH, especially when ϵ is small.

The reason is that the improvement of the group-based approaches depends on the length of each group, which is limited to the length of the delay time w. Therefore, when the delay time is very short, it limits the accuracy improvement of the group-based optimization strategy. For compOrder, increasing the length of the delay time can enhance the data consistency of the released results, thereby improving the accuracy. However, the number of times each data needs to be used for comparison increases, and the privacy budget for each comparison decreases, leading to inaccurate recorded comparison results at the same time. Additionally, when the similarity between streaming data within a certain length is higher, shorter delay time are more advantageous, while longer delay time are more favorable for CompOrder in the opposite case. These factors intertwined weaken the impact of delay time variations on CompOrder. For BucOrder, the accuracy improvement comes from enhancing the order consistency of the released results through bucketing and reducing the error introduced by noise. Buckets are partitioned based on the data domain, and for data streams where the data is relatively concentrated within a certain range, changing the delay time may not significantly alter the number of buckets the data falls into, thereby not noticeably reducing the noise error.

Impact of the key parameters. In Figure 6, we evaluate the impact of the bucket size on the accuracy of BucOrder under three different levels of privacy protection strength. We vary the bucket size m from 10 to 100 on the Unemployment and vary m from $100,000$ to $100,000$ on the Outpatient due to its excessively large data domain. From the results on the Unemployment, we can observe that when ϵ is small, i.e., $\epsilon = 0.4,\ 0.6$, too small bucket sizes can lead to a decrease in the accuracy of the released results. The reason is that a too-small bucket size implies a large number of buckets, this can increase the variance introduced by the GRR mechanism used by BucOrder. However, it does not imply that a larger m leads to higher accuracy, a larger m can introduce greater bias to the noisy results. The optimal m is highly correlated

TABLE III
COMPARISON OF PROPOSED METHODS WITH 10 TIMESTAMPS DELAY AND EXISTING METHODS OF OTHER PRIVACY SETTINGS UNDER $\epsilon = 0.1, 0.5, 1$. DPI IS THE STATE-OF-THE-ART METHOD UNDER USER-LEVEL PRIVACY SETTING, WHILE ADAPUB IS DESIGNED UNDER w-EVENT LEVEL PRIVACY SETTING. WE MODIFY THEM TO THE EVENT-LEVEL PRIVACY SETTING, AND COMPARE ALL METHODS AGAINST THE ACCURACY OF THE NAÏVE APPROACH AS A REFERENCE VALUE.

Methods	**Unemployment**			**Outpatient**		
	0.1	0.5	1.0	0.1	0.5	1.0
Naïve	1.0×	1.0×	1.0×	1.0×	1.0×	1.0×
DPI-event	0.08×	0.38×	0.76×	0.05×	0.25×	0.48×
Adapub-event	1.26×	1.24×	1.26×	1.23×	1.25×	1.23×
Discontin	0.41×	0.41×	0.41×	0.41×	0.41×	0.41×
CompOrder	0.19×	0.28×	0.40×	0.20×	0.39×	0.56×
BucOrder	0.03×	0.17×	0.35×	0.03×	0.15×	0.29×

with the distribution of the data stream. Although we cannot theoretically derive the optimal m, suboptimal m do not significantly decrease the accuracy, i.e., $m = 100$ is not optimal, but the accuracy of BucOrder remains superior to others in Figure 2 and Figure 3.

In Figure 7 and Figure 8, we evaluate the impact of the threshold θ used for grouping on the accuracy of the continuous grouping and the discontinuous grouping approaches. All approaches add noise to the summation of the group, and the evaluation results for adding noise independently to each data are shown in Figure 9 and Figure 10 in Section A-A. We observe that the accuracy of both continuous and discontinuous approaches does not monotonically change with the threshold θ. The reason is that increasing θ can increase the length of each group while reducing the similarity of data within the group. However, the specific grouping results obtained according to θ are related to the distribution of the data stream, making it challenging to determine the optimal θ theoretically.

C. Comparison with Other Competitors

To comprehensively evaluate the significance of delay time in improving the accuracy of the stream releasing, we also compared the proposed methods with a user-level privacy method DPI [26] and a w-event-level privacy method Adapub [40]. Although both methods are not optimized for the settings in this paper, we modified them to run under the event-level setting with the aim of comparing the effectiveness of our designs. Besides, DPI conducts the learning process of data in the form of timeslots. We divide 25% data for each timeslot and increase the corresponding privacy budget in the comparison to satisfy the event DP. The comparison results are shown in Table III. The comparison results indicate that even with just 10 timestamps delay, the proposed algorithm shows accuracy advantages over state-of-the-art solution.

VIII. RELATED WORK

While the standard definition of differential privacy (DP) [16] is proposed on the concept of neighboring datasets, its application is not limited to the processing of the dataset. With the development of the internet and sensor technologies, the applications of privacy-preserving data stream processing have become more widespread. The high variability, unpredictability, and correlations in streaming data make it challenging to directly apply DP algorithms designed for datasets to handle data streams.

The earliest studies in DP for streaming data collection originate from continuous observation of private data [21], [6], [20], [13], [17], [24]. These studies mainly consider the degradation of privacy guarantee due to the repeated appearance of streaming data when the user's state not changing for a period of time. Subsequently, some studies propose methods to predict and reduce the actual sensitivity of data streams to avoid the overestimation of sensitivity caused by outliers [35], [42]. Some studies focus on adding correlated noise on correlated streaming data to prevent privacy leakage [37], [7]. Considering the potential vast data domain of the large data volumes, recent work focuses on improving the memory efficiency of privacy-preserving collection of data streams [31]. In addition to the works satisfying the traditional definition of DP (event-level DP), there are also efforts focusing on more stringent DP definitions, such as w-event-level DP and user-level DP. w-event-level DP provides ϵ-DP protection for any neighboring w timestamps. Related research is dedicated to studying the sampling of streaming data for release and allocating appropriate privacy budgets for them within any w-length sliding window [29], [10], [30], [39], [38], [36]. User-level DP provides ϵ-DP protection for all streaming data contributed by one user. To avoid the privacy budget explosion, the existing works [14], [26], [22], [25], [15], [23] primarily employing strategies like adaptively allocate privacy budget from exponentially decreasing series or limiting user contributions.

To the best of our knowledge, all existing stream-releasing solutions are designed under real-time settings. In practice, not all stream requires absolute real-time and it is challenging to achieve due to various factors such as data processing and transmission delays. A slight relaxation of timeliness may provide opportunities for all aforementioned studies to significantly enhance the utility.

IX. CONCLUSION

In this paper, we explore improving the accuracy of streaming data release results by allowing a delay time. We propose a new delay-allowed data stream releasing framework and design two kinds of optimization approaches based on grouping strategy and order-preserving strategy. Furthermore, we effectively reduce the amount of added noise by adding noise to the summation result of the data and truncating the data sensitivity. The experimental results demonstrated the effectiveness of the proposed methods. For future work, we believe that leveraging delay time can also break accuracy limitations for streaming data releasing tasks under other privacy settings, such as w-event level and user-level. Furthermore, delay-allowed strategies can be extended to other tasks related to the data stream, such as range queries, heavy hitter detection.

Acknowledgment

This research is supported by the National Key Research and Development Program of China under Grant 2021YFB3100300 and the National Natural Science Foundation of China under Grants 62072395 and U20A20178. Yuan Hong is partially supported by the National Science Foundation (NSF) under Grants CNS-2302689, CNS-2308730, CNS-2319277, and CMMI-2326341. Additionally, we would like to express our sincere gratitude to the anonymous reviewers for their valuable time and insightful comments, which greatly contributed to the quality and rigor of this paper.

References

[1] "Unemployment." https://fred.stlouisfed.org/series/LNU03000018, 2023.

[2] "Covid19 death." https://gis.cdc.gov/grasp/fluview/mortality.html, 2024.

[3] "Foodmart." https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php, 2024.

[4] "Outpatient." https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html, 2024.

[5] M. Almashor, A. Fadiansyah, C. Pathmabandu, M. Amos, and M. A. P. Chamikara, "Mitigating privacy leakage in anomalous building data streams," in *Proceedings of the 10th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, BuildSys 2023, Istanbul, Turkey, November 15-16, 2023*. ACM, 2023, pp. 333–339.

[6] N. Bansal, D. Coppersmith, and M. Sviridenko, "Improved approximation algorithms for broadcast scheduling," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*. ACM Press, 2006, pp. 344–353.

[7] E. Bao, Y. Yang, X. Xiao, and B. Ding, "Cgm: an enhanced mechanism for streaming data collection with local differential privacy," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 2258–2270, 2021.

[8] M. Bun and T. Steinke, "Concentrated differential privacy: Simplifications, extensions, and lower bounds," in *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, ser. Lecture Notes in Computer Science, M. Hirt and A. D. Smith, Eds., vol. 9985, 2016, pp. 635–658.

[9] Y. Chen, A. Machanavajjhala, M. Hay, and G. Miklau, "Pegasus: Data-adaptive differentially private stream processing," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 1375–1388.

[10] M. Cheng, Y. Sun, B. Zhao, and J. Su, "An event grouping approach for infinite stream with differential privacy," in *Advances in Services Computing - 10th Asia-Pacific Services Computing Conference, APSCC 2016, Zhangjiajie, China, November 16-18, 2016, Proceedings*, ser. Lecture Notes in Computer Science, G. Wang, Y. Han, and G. M. Pérez, Eds., vol. 10065, 2016, pp. 106–116.

[11] J.-S. Chou and N.-T. Ngo, "Smart grid data analytics framework for increasing energy savings in residential buildings," *Automation in construction*, vol. 72, pp. 247–257, 2016.

[12] D. Dias and J. Paulo Silva Cunha, "Wearable health devices—vital sign monitoring, systems and technologies," *Sensors*, vol. 18, no. 8, p. 2414, 2018.

[13] B. Ding, J. Kulkarni, and S. Yekhanin, "Collecting telemetry data privately," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 3571–3580.

[14] W. Dong, Q. Luo, and K. Yi, "Continual observation under user-level differential privacy," in *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 2190–2207.

[15] C. Dwork, "Differential privacy in new settings," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, M. Charikar, Ed. SIAM, 2010, pp. 174–183.

[16] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, "Calibrating noise to sensitivity in private data analysis," *J. Priv. Confidentiality*, vol. 7, no. 3, pp. 17–51, 2016.

[17] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum, "Differential privacy under continual observation," in *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, L. J. Schulman, Ed. ACM, 2010, pp. 715–724.

[18] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3-4, pp. 211–407, 2014.

[19] C. Dwork, G. N. Rothblum, and S. P. Vadhan, "Boosting and differential privacy," in *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*. IEEE Computer Society, 2010, pp. 51–60.

[20] Ú. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, G. Ahn, M. Yung, and N. Li, Eds. ACM, 2014, pp. 1054–1067.

[21] A. V. Evfimievski, J. Gehrke, and R. Srikant, "Limiting privacy breaches in privacy preserving data mining," in *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, F. Neven, C. Beeri, and T. Milo, Eds. ACM, 2003, pp. 211–222.

[22] L. Fan and L. Xiong, "Real-time aggregate monitoring with differential privacy," in *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, X. Chen, G. Lebanon, H. Wang, and M. J. Zaki, Eds. ACM, 2012, pp. 2169–2173.

[23] ——, "An adaptive approach to real-time aggregate monitoring with differential privacy," *IEEE Transactions on knowledge and data engineering*, vol. 26, no. 9, pp. 2094–2106, 2013.

[24] L. Fan, L. Xiong, and V. Sunderam, "Differentially private multi-dimensional time series release for traffic monitoring," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2013, pp. 33–48.

[25] F. Farokhi, "Temporally discounted differential privacy for evolving datasets on an infinite horizon," in *11th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2020, Sydney, Australia, April 21-25, 2020*. IEEE, 2020, pp. 1–8.

[26] S. Feng, M. Mohammady, H. Wang, X. Li, Z. Qin, and Y. Hong, "Dpi: Ensuring strict differential privacy for infinite data streaming," *arXiv preprint arXiv:2312.04738*, 2023.

[27] S. Ghayyur, Y. Chen, R. Yus, A. Machanavajjhala, M. Hay, G. Miklau, and S. Mehrotra, "Iot-detective: Analyzing iot data under differential privacy," in *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, G. Das, C. M. Jermaine, and P. A. Bernstein, Eds. ACM, 2018, pp. 1725–1728.

[28] M. Hay, V. Rastogi, G. Miklau, and D. Suciu, "Boosting the accuracy of differentially private histograms through consistency," *Proc. VLDB Endow.*, vol. 3, no. 1, pp. 1021–1032, 2010.

[29] G. Kellaris, S. Papadopoulos, X. Xiao, and D. Papadias, "Differentially private event sequences over infinite streams," *Proc. VLDB Endow.*, vol. 7, no. 12, pp. 1155–1166, 2014.

[30] H. Li, L. Xiong, X. Jiang, and J. Liu, "Differentially private histogram publication for dynamic datasets: an adaptive sampling approach," in *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, J. Bailey, A. Moffat, C. C. Aggarwal, M. de Rijke, R. Kumar, V. Murdock, T. K. Sellis, and J. X. Yu, Eds. ACM, 2015, pp. 1001–1010.

[31] X. Li, W. Liu, J. Lou, Y. Hong, L. Zhang, Z. Qin, and K. Ren, "Local differentially private heavy hitter detection in data streams with bounded memory," *CoRR*, vol. abs/2311.16062, 2023.

[32] M. Lyu, D. Su, and N. Li, "Understanding the sparse vector technique for differential privacy," *Proc. VLDB Endow.*, vol. 10, no. 6, pp. 637–648, 2017.

[33] F. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," in *Proceedings of the ACM SIGMOD*

International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009, U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, Eds. ACM, 2009, pp. 19–30.

[34] Y. Nie, L. Huang, Z. Li, S. Wang, Z. Zhao, W. Yang, and X. Lu, "Geospatial streams publish with differential privacy," in *Collaborate Computing: Networking, Applications and Worksharing - 12th International Conference, CollaborateCom 2016, Beijing, China, November 10-11, 2016, Proceedings*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, S. Wang and A. Zhou, Eds., vol. 201. Springer, 2016, pp. 152–164.

[35] V. Perrier, H. J. Asghar, and D. Kaafar, "Private continual release of real-valued data streams," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[36] X. Ren, L. Shi, W. Yu, S. Yang, C. Zhao, and Z. Xu, "LDP-IDS: local differential privacy for infinite data streams," in *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Z. G. Ives, A. Bonifati, and A. E. Abbadi, Eds. ACM, 2022, pp. 1064–1077.

[37] H. Wang and Z. Xu, "CTS-DP: publishing correlated time-series data via differential privacy," *Knowl. Based Syst.*, vol. 122, pp. 167–179, 2017.

[38] Q. Wang, X. Lu, Y. Zhang, Z. Wang, Z. Qin, and K. Ren, "Secweb: Privacy-preserving web browsing monitoring with w-event differential privacy," in *Security and Privacy in Communication Networks - 12th International Conference, SecureComm 2016, Guangzhou, China, October 10-12, 2016, Proceedings*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, R. H. Deng, J. Weng, K. Ren, and V. Yegneswaran, Eds., vol. 198. Springer, 2016, pp. 454–474.

[39] Q. Wang, Y. Zhang, X. Lu, Z. Wang, Z. Qin, and K. Ren, "Rescuedp: Real-time spatio-temporal crowd-sourced data publishing with differential privacy," in *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 2016, pp. 1–9.

[40] T. Wang, X. Yang, X. Ren, J. Zhao, and K. Lam, "Adaptive differentially private data stream publishing in spatio-temporal monitoring of iot," in *38th IEEE International Performance Computing and Communications Conference, IPCCC 2019, London, United Kingdom, October 29-31, 2019*. IEEE, 2019, pp. 1–8.

[41] T. Wang, J. Blocki, N. Li, and S. Jha, "Locally differentially private protocols for frequency estimation," in *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, E. Kirda and T. Ristenpart, Eds. USENIX Association, 2017, pp. 729–745.

[42] T. Wang, J. Q. Chen, Z. Zhang, D. Su, Y. Cheng, Z. Li, N. Li, and S. Jha, "Continuous release of data streams under both centralized and local differential privacy," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 1237–1253.

Appendix A

A. Supplementary Experimental Results

The impact of threshold θ on the group-based approaches with adding noise to the individual data is shown in Figure 9 and Figure 10, respectively.

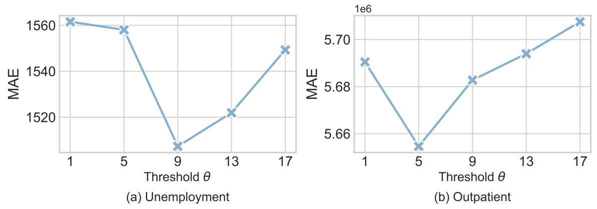

Fig. 9. Impact of threshold θ on the accuracy of the continuous group-based approach, where $\epsilon = 0.4$, and $w = 100$. The noise is added individually to the data.

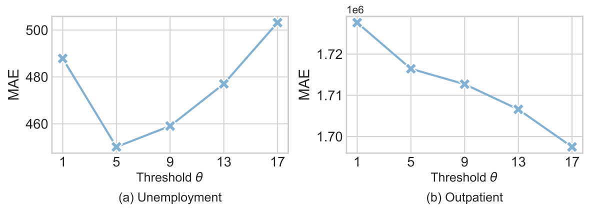

Fig. 10. Impact of threshold θ on the accuracy of the discontinuous group-based approach, where $\epsilon = 0.4$, and $w = 100$. The noise is added individually to the data.

B. Algorithm of Continuous Grouping in Batch

The algorithm of continuous grouping in a data batch is shown in Algorithm 6.

Algorithm 6 Continuous Grouping in A Data Batch.

```
Input: Streaming data $B_i = x_{i\cdot w}, ..., x_{(i+1)\cdot w-1}$, privacy budget $\epsilon_g$
    for grouping, threshold $\theta$.
Output: $\mathcal{G} = \{G_1, G_2, ...\}$
 1: $\mathcal{G} \leftarrow \emptyset$, $c \leftarrow 1$
 2: for $x_i \in B_i$ do
 3:     if $i$ equals 1 or $G_c$ is closed then
 4:         $c \leftarrow c+1$, $G_c \leftarrow \emptyset$, $\tilde{\theta} \leftarrow \theta + \mathsf{Lap}\,(2\Delta_s/\epsilon_g)$
 5:     end if
 6:     if $dev(G_c \cup x_i) + \mathsf{Lap}\,(4\Delta_s/\epsilon_g) < \tilde{\theta}$ then
 7:         Involve $x_i$ into $G_c$ and $G_c$ still open.
 8:     else
 9:         Close $G_c$, Add $G_c$ to $\mathcal{G}$, $c \leftarrow c+1$
10:         Involve $x_i$ into a new group $G_c$ and then close $G_c$, Add
            $G_c$ to $\mathcal{G}$.
11:     end if
12: end for
13: if $G_c$ is open then
14:     Add $G_c$ to $\mathcal{G}$.
15: end if
16: return $\mathcal{G} = \{G_1, G_2, ...\}$
```

C. Proof of Theorem 1

Proof *We assume that the stream D and D' differ by the data at the i^{th} timestamp. $\mathcal{G}_i$ represents the grouping status at i^{th} timestamp. The probability distribution of group states $\mathcal{G}_i$ at the $i-1$ timestamps preceding the i^{th} timestamp are all identical. From the i^{th} timestamp onwards, the probability distribution of group states at each timestamp may have difference between D and D'. We have*

$$\mathsf{Pr}[M(\mathcal{G}_{w-1}, x_w) = \mathcal{G}] = \prod_{j=1}^{i-1} \mathsf{Pr}[M(\mathcal{G}_{j-1}, x_j) = \mathcal{G}_j]$$

$$\cdot\, \mathsf{Pr}[M(\mathcal{G}_{i-1}, x_i) = \mathcal{G}_i] \cdot \prod_{j=i+1}^{w} \mathsf{Pr}[M(\mathcal{G}_{j-1}, x_j) = \mathcal{G}_j]$$

Here, $M(\mathcal{G}_{j-1}, x_j)$ using the group set up to the previous timestamp and the current data as input, output the group *set for the current timestamp. Then according to the privacy guarantee provided by SVT [32], we have*

$$
\begin{aligned}
&\frac{\Pr[M(\mathcal{G}'_{w-1}, x_w) = \mathcal{G}]}{\Pr[M(\mathcal{G}_{w-1}, x_w) = \mathcal{G}]} \\
=&\frac{\Pr[M(\mathcal{G}_{i-1}, x'_i) = \mathcal{G}'_i] \cdot \prod_{j=i+1}^{w} \Pr[M(\mathcal{G}'_{j-1}, x_j) = \mathcal{G}'_j]}{\Pr[M(\mathcal{G}_{i-1}, x_i) = \mathcal{G}_i] \cdot \prod_{j=i+1}^{w} \Pr[M(\mathcal{G}_{j-1}, x_j) = \mathcal{G}_j]} \\
\le& e^{\epsilon_g} \cdot \frac{\prod_{j=i+1}^{w} \Pr[M(\mathcal{G}'_{j-1}, x_j) = \mathcal{G}'_j]}{\prod_{j=i+1}^{w} \Pr[M(\mathcal{G}_{j-1}, x_j) = \mathcal{G}_j]}
\end{aligned}
$$

When x_i and x'_i are assigned to the same group, we have

$$\frac{\Pr[M(D') = \mathcal{G}]}{\Pr[M(D) = \mathcal{G}]} \le e^{\epsilon_g} \cdot e^{(w-i)\epsilon_g} = e^{(w-i+1)\epsilon_g} \le e^{w\epsilon_g}$$

Otherwise, we have

$$\frac{\Pr[M(D') = \mathcal{G}]}{\Pr[M(D) = \mathcal{G}]} \le e^{\epsilon_g} \cdot e^{2(w-i)\epsilon_g} = e^{(2(w-i)+1)\epsilon_g} \le e^{(2w-1)\epsilon_g}$$

Therefore, when we provide $\epsilon_g/(2w-1)$ privacy guarantee for each SVT judgement, Algorithm 2 satisfies ϵ_g-DP. □

D. Proof of Theorem 4

Proof *The expected squared error comes from Laplace noise, the bias introduced by approximating data within a group with their mean, and the error introduced by SVT for grouping. Denote n_g as the number of groups in a delay time, β is the failure probability of SVT, and $m_{>\theta}$ is the approximate bias in the data added to a group due to the misjudgment of SVT. We have*

$$\mathbb{E}\left[(\hat{B}_i - B_i)^2\right] = n_g \cdot 2\frac{|d|^2}{\epsilon_p^2} + (1-\beta)w\theta^2 + \beta w\,(m_{>\theta})^2 \tag{4}$$

We first need to obtain the privacy guarantee of SVT. There are no existing conclusions can be directly applied to our current computation. Our proof is inspired by Theorem 3.24 in [18]. We want to approve that except with at most β, we have:

$$
\begin{aligned}
&Dev(G_i \cup x_i) + v_i \ge \hat{\theta} \ge \theta + |\hat{\theta} - \theta| \\
&Dev(G_i \cup x_i) \ge \theta + (|\hat{\theta} - \theta| - |v_i|) \ge \theta + \alpha
\end{aligned}
$$

Recall that if $Y \sim \mathrm{Lap}\,(b)$, then it has $\Pr[|Y| \ge t \cdot b] = e^{-t}$. Thus, we have:

$$\Pr[|\hat{\theta} - \theta| \ge 2\alpha] = e^{-\alpha\epsilon_g}$$

When $\alpha \ge \frac{\log(2/\beta)}{\epsilon_g}$, this quantity to be at most $\beta/2$.

$$\Pr[|v_i| \le \alpha] = e^{-\frac{\epsilon_g \alpha}{4}}$$

When $\alpha \le \frac{4\log(2/(2-\beta))}{\epsilon_g}$, this quantity to be at most $\beta/2$. Therefore, we can obtain the conclusion that the probability of $Dev(G_i \cup x_i) \ge \theta + \alpha$ is at most β when $\frac{\log(2/\beta)}{\epsilon_g} \le \alpha \le \frac{4\log(2/(2-\beta))}{\epsilon_g}$.

Based on this conclusion, the Equation 4 is:

$$
\begin{aligned}
\mathbb{E}\left[(\hat{B}_i - B_i)^2\right] \le\ & n_g \cdot 2\frac{|d|^2}{\epsilon_p^2} + (1-\beta)w\theta^2 \\
&+ \beta w \left(\theta + \frac{4\log(2/(2-\beta))}{\epsilon_g}\right)^2
\end{aligned}
$$

□

E. Proof of the Theorem 2

Proof *In Algorithm 3, recording the comparison results between the current data and the subsequent data within the delay time follows the SVT design. The standard version of SVT and the proof of its privacy guarantee can be found in [32]. As shown in Figure 11, essentially, the streaming data at each timestamp participates in two independent SVT algorithms. For the streaming data at the i^{th} timestamp, in the first SVT, each query checks whether $x_j - x_i$ is greater than 0, where x_j represents the data preceding x_i within the w timestamps. In the second SVT, each query checks whether $x_i - x_j$ is greater than 0, where x_j represents the data following x_i within the w timestamps.*

To satisfy DP, we divide the privacy budget ϵ_g, which is used to record the order information, by 2 for the two SVT algorithms. Additionally, since the maximum possible number of positive query answers for each SVT is w, ϵ_g needs to be divided by w. Therefore, the algorithm of CompOrder satisfies ϵ_p-DP. □

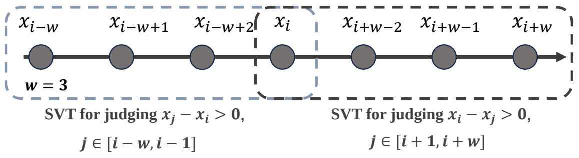
x_{i-w}
x_{i-w+1}
x_{i-w+2}
x_i
x_{i+w-2}
x_{i+w-1}
x_{i+w}
$w = 3$
SVT for judging $x_j - x_i > 0$, $j \in [i-w, i-1]$
SVT for judging $x_i - x_j > 0$, $j \in [i+1, i+w]$

Fig. 11. Taking a delay time $w = 3$ as an example, it can be seen that each streaming data x_i participates in two SVT.

F. Proof of Theorem 5

Proof *In CompOrder, the streaming data x_i at each timestamp is compared with w data within the delay time. Assuming the average distance between adjacent ordered streaming data is always larger than a, we calculate the probability of the maximum data $\hat{x}_l$ below x_i and the minimum data x_r above x_i remaining the maximum and minimum after adding noise.*

Firstly, we calculate the probability of x_l and m_r being surpassed by the values that are adjacent in the sorted order after adding noise. Since the probability calculations for x_l and m_r exhibit symmetry, let's first focus on calculating one side (Denote this case as H_r for convenience of expression).

$$
\begin{aligned}
&\Pr\left[H_r^1\right] \\
&= \frac{1}{2\lambda}\int_{-a}^{a} e^{-\frac{x}{\lambda}}dx + 2\int_{a}^{+\infty} \frac{1}{2\lambda}e^{-\frac{y}{\lambda}} \int_{y-a}^{+\infty} \frac{1}{2\lambda}e^{-\frac{x}{\lambda}}dxdy \\
&= 1 - \frac{3}{4}e^{-\frac{a}{\lambda}}
\end{aligned}
$$

Then, assuming there are s data on the left and $w-s$ data on the right, the probability that both x_l and x_r simultaneously maintain the maximum on the left and minimum on the right is

$$\begin{aligned}\Pr[H_lH_r] &= \left(1-\frac{3}{4}e^{-\frac{a}{\lambda}}\right)\left(1-\frac{3}{4}e^{-\frac{2a}{\lambda}}\right)\cdots\left(1-\frac{3}{4}e^{-\frac{sa}{\lambda}}\right)\\ &\cdot\left(1-\frac{3}{4}e^{-\frac{a}{\lambda}}\right)\left(1-\frac{3}{4}e^{-\frac{2a}{\lambda}}\right)\cdots\left(1-\frac{3}{4}e^{-\frac{(w-s)a}{\lambda}}\right)\\ &\geq \left(1-\frac{3}{4}e^{-\frac{a}{\lambda}}\right)^w\end{aligned}$$

If we simultaneously consider the errors introduced by SVT and want both H_l and H_r to hold, we only need to calculate the probability that x_l and x_r are both judged correctly. This is because the bound of $\Pr[H_lH_r]$ holds true regardless of whether other data, excluding these two, are misjudged by SVT and fall on the opposite side of x_i. We have

$$\Pr[H_lH_r] \geq \beta_1\beta_2\left(1-\frac{3}{4}e^{-\frac{a}{\lambda}}\right)^w$$

Here, β_1 and β_2 are the probabilities of SVT making accurate comparison. According to the error bound for SVT provided by Theorem 3.24 in [18], β_1 and β_2 satisfies $\beta_1, \beta_2 \geq 1-2e^{-\epsilon a/8}$. Therefore, we have

$$\Pr[H_lH_r] \geq \left(1-2e^{-\epsilon_g a/16}\right)^2\left(1-\frac{3}{4}e^{-\frac{a\Delta_s}{\epsilon_p}}\right)^w$$

□

G. Supplement Proof of Theorem 6

Proof *We analyze the expected squared error in a batch of the BucOrder. The error comes from Laplace noise, the bias introduced by approximating data within a bucket with their mean, and the error introduced by GRR mechanism for mapping the data into the buckets. Denote n_b as the number of buckets in a delay time, $p = \frac{e^{\epsilon_g}}{(e^{\epsilon_g}+n_b-1)}$ is the probability that a data is in the right bucket, and m_{out} is the approximate bias introduced by data randomized to the wrong bucket by GRR.*

$$\begin{aligned}\mathbb{E}\left[(\hat{B}_i-B_i)^2\right] &= n_b\cdot 2\frac{|d|^2}{\epsilon_p^2}+pw\cdot m^2+(1-p)w\cdot(m_{out})^2\\ &= n_b\cdot 2\frac{|d|^2}{\epsilon_p^2}+w\cdot\frac{m^2+(n_b-1)m_{out}^2}{e^{\epsilon_g}+n_b-1}\end{aligned}$$

Since the GRR uniformly maps data to other buckets with a probability $\frac{1}{e^{\epsilon_g}+n_b-1}$, it's difficult to constrain m_{out}. We calculate the approximate expected value of m_{out} is $\frac{|d|}{3}$. We assume that the mean of the data in the i^{th} bucket falls at a distance of η from the left boundary of the bucket. Since values outside the bucket are uniformly mapped into the current bucket with equal probability, we can calculate the expected m_{out} as

$$\begin{aligned}&\mathbb{E}[m_{out}]\\ &= \sum_{0\leq i\leq n_b}\frac{\sum_{(m-\eta)\leq j\leq m-\eta+|d|-(i+1)m} j+\sum_{\eta+1\leq j\leq \eta+im} j}{(|d|-m)n_b}\\ &\simeq \sum_{0\leq i\leq n_b}\frac{(|d|-(i+1)m)^2+(i\cdot m)^2}{2(|d|-m)n_b}(assume\ m\ll |d|)\\ &\simeq \sum_{0\leq i\leq n_b}\left(\frac{(|d|-m)-2im}{2n_b}+\frac{(im)^2}{(|d|-m)n_b}\right)\\ &\simeq \frac{(|d|-m)-2im}{2}+\frac{m(n_b+1)}{2}+\frac{m^2(n_b+1)(2n_b+1)}{6(|d|-m)}\\ &\simeq \frac{|d|}{3}\end{aligned}$$

Substituting the approximate expected m_{out} into the equation, we get

$$\mathbb{E}\left[(\hat{B}_i-B_i)^2\right] = n_b\cdot 2\frac{|d|^2}{\epsilon_p^2}+w\cdot\frac{m^2+(n_b-1)|d|^2/9}{e^{\epsilon_g}+n_b-1}$$

□

Appendix B
Artifact Appendix

A. Description & Requirements

1) How to access: We have stored the code repository on the Zenodo platform, and you can access it via the link https://doi.org/10.5281/zenodo.13643225.

2) Hardware dependencies: None.

3) Software dependencies: Python 3.10 and above. 'numpy', 'math', 'random', 'os', 'sys', 'matplotlib', and 'scikit-learn' libraries are required.

4) Benchmarks: All datasets used in the experiments are stored in the './data' folder.

B. Artifact Installation & Configuration

We have tested the source code in Python 3.10.8 and 3.11.4 environments, the code runs without issues in these environments. Additionally, please ensure that the 'numpy', 'math', 'random', 'os', 'sys' and 'matplotlib' libraries are installed. The code does not have high hardware requirements; a single-core CPU is sufficient.

C. Major Claims

- (C1): Allowing a delay of only 10 timestamps can bring about a substantial increase in accuracy for the released streams (Compare CompOrder, Discontin_pp, BucOrder, Contin_reduce, Discontin_reduce with Naive, PeGaSus, PeGaSus_delay).
- (C2): The performance of PeGaSus is not improved compared to the naïve approach and is even lower than the naïve approach.
- (C3): The advantage of the order-based approach (CompOrder, BucOrder) is more pronounced when the delay time is short compared to the group-based approaches

TABLE IV
THE RUNNING TIME OF EXPERIMENT (E1)

$Delay$	COVID	Unemploy	Outpatient	Foodmart
$w = 10$	5.9s	5.7s	1min33s	2min42s
$w = 100$	8.9s	22.5s	3min10s	13min12s

(Discontin_pp, Contin_reduce, Discontin_reduce, PeGaSus_delay).

- (C4): The data sensitivity truncation algorithm has an improvement for all methods (except BucOrder) on all datasets, especially on streams with large data domains such as Outpatient and Foodmart datasets.

D. Evaluation

In this section, we provide a detailed step-by-step verification of the experimental results included in the paper. (E1) and (E2) involve validating the effectiveness of the approaches proposed in this paper. (E3) and (E4) evaluate the impact of parameters on the proposed approaches, (E5) compares the proposed methods with state-of-the-art methods from other privacy settings.

1) Experiment (E1): [The Accuracy Improvement of the Delay-allowed Approaches.][About 21min]

This part of the experiment corresponds to Figures 2 and Figure 3 in the paper. The approximate running times for the four datasets under two different delay_time parameters are shown in Table IV.

[Execution] There are two steps for verifying Figure 2 and Figure 3.

Step 1. Set 'delay_time=10' and run 'python est_compall.py'. The terminal will sequentially display the error results of all methods on the four datasets, and comparison line charts for all methods will pop up. Please manually close the popped-up line charts for the code to continue running the next dataset.

Step 2. Set 'delay_time=100' and rerun 'python est_compall.py'. Other operations are the same as in Step 1.

[Results] In each step, the results displayed in the terminal and the line charts popped up by the code correspond to Figures 2/Figure 3 (a), (b), (c), and (d) in the paper. There may be some fluctuations, the trend of each method's error with changes in ϵ, and the relative magnitudes should be consistent with those in the paper. The observed results in this part verify C1, C2, and C3 in the Major Claims.

2) Experiment (E2): [The Effectiveness of Data Sensitivity Truncation Algorithm.][About 7min47s]

This part corresponds to Figure 4 in the paper and is used to evaluate the effectiveness of the sensitivity reduction method proposed in this paper.

[Execution] Run 'python est_sensitivity.py'.

[Results] The terminal will sequentially display the results of the Naive, Contin, Discontin, CompOrder, and BucOrder methods on the four datasets, both without sensitivity reduction and with sensitivity reduction. The data under "No Sensitivity Reduce" correspond to the red bars in Figure 4, while the results under "Sensitivity Reduce" correspond to the blue bars. Although there may be fluctuations due to randomness, the magnitude of the data should be consistent with those in Figure 4. For all datasets, except for the BucOrder method, the data under "Sensitivity Reduce" should show a decrease compared to "No Sensitivity Reduce". The observed results in this part verify C4 in the Major Claims.

3) Experiment (E3): [Comparison of Group-based Approaches and Order-based Approaches.][About 58s]

This part corresponds to Figure 5 in the paper, illustrating the impact of delay time length on group-based and order-based methods respectively.

[Execution] Run 'python est_delaylength.py'.

[Results] In the terminal, the results for the discontinuous, CompOrder, and BucOrder methods on two datasets will be sequentially displayed, showing the error in published results as the parameter w varies from 10 to 90. The results for each dataset will be between "Results" and "Results End". All data should correspond to the bars in Figure 5.

It should be observed from the results: a. The performance of the discontinuous approach improves as the length of the delay time increases. b. The order-based (CompOrder and BucOrder) approaches do not exhibit a clear pattern of accuracy variation with changes in the delay time. c. When the delay time is short, order-based (CompOrder and BucOrder) approaches demonstrate an advantage over group-based (discontinuous) approaches.

4) Experiment (E4): [Impact of the key parameters.][About 18s and 12s]

This part of the experiment corresponds to Figures 6, 7, 8, 9, and 10, primarily evaluating the impact of parameter variations on the performance of the proposed algorithms.

[Execution] There are two steps in this part. We have already set all the parameters.

Step 1. Run 'python est_order.py' and manually close the popped-up line charts to proceed with running the next dataset.

Step 2. Run 'python est_group.py'.

[Results] For Step 1, line charts similar to those in Figure 6 from the paper will pop up. It should be observed that too-small bucket size m can lead to a decrease in the accuracy of the released results.

For Step 2, the terminal will sequentially display the accuracy of the publication results for four group-based methods across different values of τ on two datasets. The results will show that the impact of θ on group-based methods is not monotonical. The magnitudes of the error results should correspond to the respective figures in the paper.

5) Experiment (E5): [Compare with existing works come from other privacy settings.][DPI: 31s, Adapub: 4s] This part of the experiment corresponds to Table III in the paper. The source code for DPI and Adapub comes from their corresponding open-source libraries. We only made some minor modifications to make them meet the event-level privacy and set the parameters for comparison in our experiment.

[Execution] There are two steps in this part.

Step 1. Run 'python est_other.py'.

Step 2. Run ‘python DPI_DEMO_script.py’.

[Results] In Step 1 of E1, it will show the error results of $\epsilon \in [0.1, 1]$ for Naïve, Discontin, CompOrder, and BucOrder on the Unemployment and Outpatient datasets. By dividing the error results (MAE) of all methods by the error results of naïve, the correctness of Table III can be verified. The proposed algorithm should show accuracy advantages over state-of-the-art solutions. Directly comparing the error results of these methods can also validate this.

E. Acknowledgement

We would like to express our sincere gratitude to the AE reviewers for their thorough evaluation of our work. Your valuable feedback and insightful comments have significantly contributed to improving both the quality of our code and the clarity of our paper.