Defense against Poisoning Attacks under Shuffle-DP

SIYI WANG[*†], Nanyang Technological University, Singapore
QIYAO LUO[*], OceanBase, Ant Group, China
YIHUA HU[*], Nanyang Technological University, Singapore
LIXU WANG, Nanyang Technological University, Singapore
QUANQING XU, OceanBase, Ant Group, China
CHUANHUI YANG, OceanBase, Ant Group, China
ZHAN QIN, Zhejiang University, China
KUI REN, Zhejiang University, China
WEI DONG[‡], Nanyang Technological University, Singapore

Differential Privacy (DP) has become the gold standard for protecting individual privacy in data analytics, and the shuffle-DP model has attracted significant attention from both academia and industry due to its favorable balance between privacy and utility. However, existing shuffle-DP protocols rely on a strong assumption: all users behave honestly. In real-world scenarios, adversarial users can exploit this vulnerability through poisoning attacks, compromising both privacy guarantees and the utility of analytical results. While defending against poisoning attacks in the shuffle-DP model has recently gained interest, existing solutions are limited to frequency estimation tasks. To address this issue, we propose the first general defense framework for all union-preserving queries, capable of transforming any shuffle-DP protocol into a version resilient to poisoning attacks. Beyond robust defense against poisoning attacks, our framework achieves high utility of analytical results. Compared to the original shuffle-DP protocol, it retains asymptotically equivalent error in attack-free settings and incurs only a polylogarithmic increase in error when a constant number of attackers are present. We demonstrate the generality of our framework on several common queries, including summation, frequency estimation, and range counting. Experimental results confirm that our approach effectively defends against poisoning attacks while maintaining strong utility and communication efficiency.

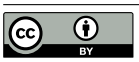
CCS Concepts: • **Security and privacy → Database and storage security**; • **Information systems → Data management systems**.

Additional Key Words and Phrases: Differential privacy, shuffle model, poisoning attacks

ACM Reference Format:
Siyi Wang, Qiyao Luo, Yihua Hu, Lixu Wang, Quanqing Xu, Chuanhui Yang, Zhan Qin, Kui Ren, and Wei Dong. 2026. Defense against Poisoning Attacks under Shuffle-DP. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 24 (February 2026), 27 pages. https://doi.org/10.1145/3786638

[*]These authors contributed equally to this research.
[†]Work partially completed during internship at Ant Group.
[‡]Wei Dong is the corresponding author.

Authors' Contact Information: Siyi Wang, Nanyang Technological University, Singapore, Singapore, wang.siyi@ntu.edu.sg; Qiyao Luo, OceanBase, Ant Group, Shanghai, China, luoqiyao.lqy@antgroup.com; Yihua Hu, Nanyang Technological University, Singapore, Singapore, yihua001@e.ntu.edu.sg; Lixu Wang, Nanyang Technological University, Singapore, Singapore, wanglixu4334@gmail.com; Quanqing Xu, OceanBase, Ant Group, Hangzhou, China, xuquanqing.xqq@oceanbase.com; Chuanhui Yang, OceanBase, Ant Group, Hangzhou, China, rizhao.ych@oceanbase.com; Zhan Qin, Zhejiang University, Hangzhou, China, qinzhan@zju.edu.cn; Kui Ren, Zhejiang University, Hangzhou, China, kuiren@zju.edu.cn; Wei Dong, Nanyang Technological University, Singapore, Singapore, wei_dong@ntu.edu.sg.



ACM 2836-6573/2026/2-ART24
https://doi.org/10.1145/3786638

1 Introduction

A key challenge in data analytics is to obtain meaningful insights while preserving privacy. *Differential privacy* (DP) [23], the gold standard for private data analysis, achieves this by adding noise to query results, preventing inference of individual data. Traditional DP model assumes a trusted curator who collects data and privatizes results before release, known as *central-DP*. However, real-world scenarios often lack a universally trusted curator, leading to the adoption of *local-DP* [40], where users privatize data locally before sharing it. Although local-DP offers stronger privacy guarantees, it incurs higher errors. For instance, in bit counting, central-DP achieves an error of $O(1/\varepsilon)$ by adding Laplace noise of scale $1/\varepsilon$ [23], where ε is a privacy budget to control the privacy loss (see Section 2 for more details). In contrast, the optimal error under local-DP has been proven to be $O(\sqrt{n}/\varepsilon)$ [5, 10].

To balance privacy and utility, shuffle-DP [2, 8, 15, 25] was introduced, leveraging a trusted shuffler to anonymize user messages before analysis. The process consists of three steps: (1) Each user privatizes its own data using a randomizer $\mathcal{R}(\cdot)$. (2) A trusted shuffler $\mathcal{S}$ collects all users' responses, randomly permutes them, and passes them to a potentially untrusted analyzer $\mathcal{A}$. (3) The analyzer conducts further analysis. By introducing additional randomness through shuffling, shuffle-DP reduces the required noise per user, thus improving utility. In bit counting, each user adds only $O(1/n\varepsilon)$ extra noise [30], which leads to a total error of $O(1/\varepsilon)$ and matches the central-DP error. However, unlike central-DP which places all trust in the analyzer, shuffle-DP only requires a trusted shuffler, which can be easily implemented via anonymous communication channels (e.g., mix networks [12, 19], onion routing [20, 52]) or trusted nodes/hardware [8, 53]). Due to its strong privacy-utility trade-off, shuffle-DP has been widely studied in fundamental problems such as sum aggregation [3, 28, 30], frequency estimation [16, 27, 29, 47], distinct counting [13], as well as in advanced problems such as triangle counting [36].

1.1 Poisoning Attack

However, traditional shuffle-DP protocols rely on a strong, often implicit trust assumption: while the analyzer may be adversarial, all users are honest and faithfully follow the protocol. This setting does not always align with real-world scenarios. In practice, some users themselves might act maliciously, aiming to disrupt the shuffle-DP protocol. The adversarial behavior is known as *poisoning attack* [4, 14]. More precisely, a poisoning attacker corrupts some users in the protocol and acts maliciously with two primary objectives: (1) to break privacy, and (2) to destroy the utility of analytical results.

For the privacy breaking, since the final result aggregates contributions from all users, the attacker can compromise the privacy mechanism by circumventing the required noise. For example, if half of the users are corrupted and withhold noise, the effective privacy protection is reduced by half. Mitigating such privacy issues has been extensively studied, known as *robust shuffle-DP* [1]. Existing approaches [1] suggest that honest participants introduce additional noise to counteract the noise omitted by corrupted users. Specifically, in the above example, each honest user adjusts the noise scale by doubling it to account for the lack of noise contributed by the remaining half, the corrupted users.

For the utility destroying, since shuffle-DP allows each user to send additional noise, attackers can manipulate the aggregation results by sending an excessive number of messages. In the bit counting protocol [30], each attacker can inject up to $O(n)$ noisy bits without being detected by the analyzer in the worst case, significantly compromising the utility of the final result (see Section 2.4 for more details). Astute readers may find that such an attack is a unique challenge in shuffle-DP. Under local-DP, each message is linked to a real user identity, allowing the analyzer to verify

Problem	Protocol	Detection	Recovery	#Messages/user	#Bits/message	Error w/o attacker	Error w/ attacker
Bit counting	GKMPS [30]	×	×	$1+O(\log n/n)$	2	$O(1)$	∞
	CSUZZ [15]	✓	×	1	1	$O(\log n)$	∞
	Ours + GKMPS	✓	✓	$O(\log n)$	$O(\log n)$	$O(1)$	$O(\log^3 n)$
Summation	GKMPS [30]	×	×	$1+O(\log^3 n \cdot U/n)$	$O(\log n)$	$O(U)$	∞
	BBGN(IKOS) [3]	×	×	$O(1)$	$O(\log n)$	$O(U)$	∞
	BBGN(recursive) [3]	✓	×	$O(\log\log n)$	$O(\log n)$	$O(U\sqrt{\log n}\cdot\log\log n)$	∞
	Ours + GKMPS	✓	✓	$O(\log^2 n \cdot U)$	$O(\log n)$	$O(U)$	$O(U\log^3 n)$
	Ours + BBGN(IKOS)	✓	✓	$O(\log n\cdot\log\log n)$	$O(\log n)$	$O(U)$	$O(U\log^3 n)$
Frequency estimation	GKMPS [29, 30]	×	×	$1+O(\log n\cdot U/n)$	$O(\log n)$	$O(\log n)$	∞
	GGKPV [27]	×	×	$O(\log n)$	$O(\log^2 n)$	$O(\log n)$	∞
	LWY [47]	×	×	$O(1)$	$O(\log n)$	$O(\log n)$	∞
	CZ [16]	✓	✓	2	U	$O(\log n)$	$O(\log n)$
	Ours + LWY	✓	✓	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log^3 n)$

Table 1. Comparison between our results and prior works for bit counting, summation, and frequency estimation under shuffle-DP, assuming $\varepsilon=\beta=\Theta(1)$ and $\log n=\Theta(\log(1/\delta))=\Theta(\log U)$. We use ℓ_∞ error for frequency estimation.

whether messages are reasonable. However, under shuffle-DP, since messages are anonymous, attackers can impersonate honest users and send excessive noise messages, but the analyzer cannot distinguish these poisoned messages from honest ones.

Such an issue has gained significant attention recently. Some protocols [3, 16, 27] control the number of messages each user can send, enabling the detection of corrupted users who exceed the expected count. For example, [3] studies the sum aggregation problem and limits the number of messages from a single user to $c=O(\log\log n)$. This allows the analyzer to identify poisoning attacks by counting the total number of received messages (i.e., equal to $c\cdot n$) and verifying whether each message falls in a reasonable range. However, this approach only enables attack detection, not recovery: the analyzer can recognize the happening of poisoning attacks, but cannot extract any meaningful analytical results from the poisoned messages. Consequently, corrupted users still succeed in their objective of destroying utility. Balle et al. [16] propose a shuffle-DP protocol for frequency estimation that incorporates a blind signature scheme to restrict messages during communication. This approach enables the recovery of meaningful results by filtering out messages with invalid signatures. However, the protocol is limited to scenarios where each user sends a fixed number of messages, and each message is drawn from a well-bounded domain. Consequently, it cannot be generalized to most existing shuffle-DP protocols, as they do not adhere to these constraints.

At first glance, there are several straightforward solutions to these questions: (i) Trusted shuffler with authentication: one could use the trusted shuffler to perform standard authentication and message-integrity checks, thereby preventing corrupted users from injecting arbitrarily many malicious messages. (ii) Local-DP protocols followed by a shuffler: another idea is to have each user first apply a local-DP mechanism to their data and then send the perturbed result to the shuffler, thus benefiting from privacy amplification. (iii) Two-round strategy: one may identify corrupted users based on malicious behaviors in an initial run, and then re-execute the shuffle-DP protocol with only the remaining honest users. However, we can easily show all of these approaches to fail in practice: the first requires the shuffler to support authentication, which is unrealistic to implement and offers limited utility even if feasible. The second is only effective under a very strict privacy regime, namely when $\varepsilon=O(1/\sqrt{n})$, and cannot recover meaningful results once corrupted users inject large amounts of noise. The third fails against adaptive corrupted users who behave honestly in the first round and attack in the second, leading to the same error as the shuffle-DP protocol without any defense. A more detailed discussion of these limitations is provided in Section 2.4.3.

Problem Statement. Therefore, the question comes: *Does there exist a general framework to defend against poisoning attacks in shuffle-DP, ensuring that the defense mechanism not only detects the attacks but also recovers meaningful analytical results from the poisoned messages?*

1.2 Our Results

In this work, we answer the above question affirmatively. We present a general framework for arbitrary union-preserving queries. It can transform any shuffle-DP protocol vulnerable to poisoning attacks into one that can defend against such attacks. Our approach develops a hierarchical defense framework based on the given shuffle-DP protocol. Specifically, the framework organizes the n users into a binary tree structure, where each leaf node corresponds to a single user, and each non-leaf node represents the union of users from its child nodes. Users within each node independently and in parallel execute the shuffle-DP protocol once. To detect utility-destroying attacks, the analyzer compares the output of each node with the aggregated results of its child nodes. If the deviation exceeds a predefined threshold, the node's output is flagged as poisoned. The final query result is recovered by recursively replacing poisoned results with aggregated outputs from valid child nodes' results. To defend against privacy-breaking attacks, honest users adjust their noise scale based on the estimated fraction of corrupted users in the population, thereby compensating for the noise lost due to malicious behaviors. With this idea, we can extend existing shuffle-DP protocols to defend against poisoning attacks.

More precisely, our contributions are shown below:

- We first consider the single-attacker setting and propose a general framework that extends any shuffle-DP protocol for union-preserving queries to defend against poisoning attacks. In the non-adversarial case (i.e., when no poisoning attacker is present), the framework preserves asymptotically equivalent error to the original shuffle-DP protocol. When a single attacker is present, the error increases by at most a polylogarithmic factor. Regarding communication cost, for most queries, it also increases by at most a polylogarithmic factor. As a result, we propose the first shuffle-DP mechanisms that defend against poisoning attacks for a range of tasks, including bit counting and summation and our results for most fundamental queries are summarized in Table 1.
- We further extend our framework to support the multi-attacker setting. Compared to the single-attacker case, the error increases by an additional factor of k, where k is the number of attackers, and the communication cost remains unchanged. Additionally, we propose one strategy to further optimize communication efficiency.
- We conduct comprehensive empirical evaluations[1] across three fundamental query tasks: bit counting, summation, and frequency estimation, using both synthetic and real-world datasets. Our experiments compare the proposed framework with state-of-the-art (SOTA) protocols under both non-adversarial and adversarial settings, evaluating both utility and efficiency. The empirical results consistently align with our theoretical analysis.

1.3 Related Work

DP has made a significant impact in the data management community [21, 41, 61]. Early studies primarily focus on the central-DP model [23, 32, 33, 49, 51], where a trusted curator has access to the raw data. To address trust concerns in distributed environments, local-DP was introduced and widely explored [17, 34, 35, 43, 45, 46, 54, 60, 62]. However, local-DP often suffers from an $O(\sqrt{n})$ utility loss compared to central-DP [10]. To strike a balance between these two extremes,

[1]The code is available at https://github.com/Whelsea/DefenseShuffleDP

shuffle-DP was introduced, which works in a distributed setting and matches central-DP in queries like count and sum [48, 57].

The anonymous communication was first introduced in [37], which solves the secure summation problem in $O(\log n)$ messages. Combining this idea with differential privacy [8] has led to the development of the shuffle-DP model. Since it reaches the sweet spot between privacy and utility, shuffle-DP has been widely studied in statistical analysis. [3] improved the message complexity of shuffle-DP summation protocol to $O(1)$ and [30] further improved to $1 + o(1)$, both achieving central-DP error. [16, 27, 29, 47] developed shuffle-DP protocols for the frequency estimation problem that also achieve central-DP error. [27, 48] extended the frequency estimation protocol to range counting queries. All these fundamental problems in shuffle-DP can reach central-DP error except distinct counting, where [13] proved and reached the lower bounds of both shuffle-DP (i.e., $O(\sqrt{n})$) and local-DP (i.e., $O(n)$).

Poisoning attacks have also been widely studied in different areas. [9, 14, 44, 56, 59] studies the attacks under local-DP and [4, 7, 26, 55] studies them under machine learning. In shuffle-DP, despite the concept of robust shuffle-DP [1], [16] studies the same behaviors in the frequency estimation problem with strict constraints. These types of behaviors are also studied in *secure multi-party computation* (MPC), known as malicious security. Unlike semi-honest adversaries who follow the protocol honestly but try to learn as much information from the transcript, malicious adversaries aim to destroy the protocol in different aspects. There are several general frameworks to defend against them using cryptographic methods, including zero-knowledge proof [31] and authenticated secret-sharing/garbling [6, 18, 58]. However, these methods rely on end-to-end communication and require multiple rounds, which are not easily extended to the shuffle-DP model which has only anonymous communication channels and aims for single-round communication.

2 Preliminary

2.1 Notation

Let multiset $D = \{x_1, x_2, \ldots, x_n\}$ be the input dataset, where each $x_i \in \mathcal{X}$ can be either a single element or a vector. Given a vector x, we use $||x||_{\ell_p}$ to denote its ℓ_p norm and define its ℓ_p distance to a set of vectors S as:

$$\mathrm{dis}_{\ell_p}(x, S) = \min\{\|x - y\|_{\ell_p} : y \in S\}.$$

The query $Q : \mathcal{X}^n \to \mathcal{Y}$ is a *union-preserving query* iff for any $D_1 \in \mathcal{X}^{n_1}$ and $D_2 \in \mathcal{X}^{n_2}$, we have $Q(D_1 \uplus D_2) = Q(D_1) + Q(D_2)$. Some common union-preserving queries are listed below.

(1) Bit counting: $Q_{\mathrm{count}}(D) = \sum_{i=1}^{n} x_i$ where $\mathcal{X} = \{0, 1\}$;
(2) Summation: $Q_{\mathrm{sum}}(D) = \sum_{i=1}^{n} x_i$ where $\mathcal{X} = \{0, 1, \cdots, U\}$;
(3) Frequency estimation: $Q_{\mathrm{hist}}(D) = (\sum_{i=1}^{n} \mathbb{I}[x_i = j])_{j=0}^{U}$;
(4) Range counting: $Q_{\mathrm{range}}(D) = (\sum_{i=1}^{n} \mathbb{I}[l \le x_i \le r])_{0 \le l \le r < U}$.

Unless otherwise specified, all queries considered in this paper are assumed to be union-preserving.

Let $\mathrm{Range}(Q, n) = \{Q(D), D \in \mathcal{X}^n\}$ be the output set of a query Q with all possible inputs and $\gamma_{\ell_p}(Q, n)$ be the diameter of $\mathrm{Range}(Q, n)$ in ℓ_p metric:

$$\gamma_{\ell_p}(Q, n) = \max\left\{\|Q(D) - Q(D')\|_{\ell_p}, D, D' \in \mathcal{X}^n\right\}.$$

For clarity, key notations used throughout this paper are summarized in Table 2.

2.2 Differential Privacy

Definition 2.1 (Differential Privacy)**.** For some $\varepsilon > 0$ and $0 \le \delta < n^{-\Omega(1)}$, a randomized mechanism $\mathcal{M} : \mathcal{X}^n \to \mathcal{Y}$ is (ε, δ)-differentially private if for any two neighboring datasets $D \sim D'$ (i.e., D

Notation	Meaning
x_i	Data of user i
D	Input dataset
n	#Users (equivalently, data size)
k	True #corrupted users
$\hat{k}$	Public upper bound of k
U	Domain size
Q	Union-preserving query
$Q(D)$	True value of Q on D
$\text{Range}(Q, n)$	Output set of Q with n users
$\text{dis}_{\ell_p}(x, S)$	ℓ_p-distance from vector x to set S
$\gamma_{\ell_p}(Q, n)$	ℓ_p-diameter of $\text{Range}(Q, n)$
$\mathcal{R}, \mathcal{S}, \mathcal{A}$	Randomizer, shuffler, analyzer
$\mathcal{P}_Q = (\mathcal{R}, \mathcal{S}, \mathcal{A})$	Shuffle-DP protocol for Q
$\mathcal{P}_Q(D)$	Query result of $\mathcal{P}_Q$ on D
$Y_i^{(r)}$	Messages generated by user i at level r via $\mathcal{R}$
$\mathcal{S}_g^{(r)}$	Shuffler assigned to group g at level r
$Z_g^{(r)}$	Shuffled messages of group g at level r via $\mathcal{S}_g^{(r)}$
$\varepsilon^{(r)}, \delta^{(r)}$	Privacy parameters at level r
$\beta^{(r)}$	High probablity parameter at level r
$\tilde{Q}_g^{(r)}$	Query result of group g at level r
$\text{Error}_{\ell_p}(P_Q, \varepsilon, \delta, \beta)$	ℓ_p-error guarantee of $\mathcal{P}_Q$
$\text{Msg}(P_Q, \varepsilon, \delta, n)$	Expected #messages per user under $\mathcal{P}_Q$
$\text{Bit}(P_Q, \varepsilon, \delta, U, n)$	Expected #bits per message under $\mathcal{P}_Q$

Table 2. Notations used in the paper.

and D' differ by a single element), $\mathcal{M}(D)$ and $\mathcal{M}(D')$ are (ε, δ)-indistinguishable. That is, for any subset of outputs $Y \subseteq \mathcal{Y}$,

$$\Pr[\mathcal{M}(D) \in Y] \leq e^{\varepsilon} \cdot \Pr[\mathcal{M}(D') \in Y] + \delta.$$

In practice, ε is usually a constant between 0.1 and 10, and δ should be much smaller than $1/n$. Below are some commonly used DP properties.

Lemma 2.1 (Post Processing [24])**.** If $\mathcal{M} : \mathcal{X} \to \mathcal{Y}$ satisfies (ε, δ)-DP and $\mathcal{M}' : \mathcal{Y} \to \mathcal{Z}$ is any randomized mechanism, then $\mathcal{M}'(\mathcal{M}(D))$ satisfies (ε, δ)-DP.

Lemma 2.2 (Basic Composition [24])**.** If $\mathcal{M}$ is a (possibly adaptive) composition of differentially private mechanisms $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$, where each $\mathcal{M}_i$ satisfies (ε, δ)-DP, then $\mathcal{M}$ satisfies $(k\varepsilon, k\delta)$-DP.

Lemma 2.3 (Parallel Composition [49])**.** Let $\mathcal{X}_1, \ldots, \mathcal{X}_k$ be pairwise disjoint subdomains of $\mathcal{X}$, and each $\mathcal{M}_i : \mathcal{X}_i^n \to \mathcal{Y}$ be an (ε, δ)-DP mechanism. Then the mechanism $\mathcal{M}(D) := (\mathcal{M}_1(D \cap \mathcal{X}_1), \ldots, \mathcal{M}_k(D \cap \mathcal{X}_k))$ satisfies (ε, δ)-DP.

2.3 Shuffle-DP

Different DP models can be captured by Definition 2.1 by the formulation of $\mathcal{M}(D)$. In central-DP, $\mathcal{M}(D)$ represents the output of a trusted curator. In contrast, for local-DP and shuffle-DP, where the analyzer $\mathcal{A}$ is untrusted, $\mathcal{M}(D)$ is defined by the view of $\mathcal{A}$. Under local-DP, each

user i independently privatizes its own data x_i using a local randomizer $\mathcal{R}$ and then sends the randomized response $\mathcal{R}(x_i)$ to $\mathcal{A}$ hence $\mathcal{M}(D) = (\mathcal{R}(x_1), \mathcal{R}(x_2), \ldots, \mathcal{R}(x_n))$. Under shuffle-DP, a trusted shuffler $\mathcal{S}$ randomly permutes the responses before forwarding them to analyzer $\mathcal{A}$, resulting in

$$\mathcal{M}(D) = \mathcal{S} \circ \mathcal{R}(D) = \{\mathcal{R}(x_1), \mathcal{R}(x_2), \ldots, \mathcal{R}(x_n)\}.$$

Under shuffle-DP, a protocol which is used to answer query Q is defined as $\mathcal{P}_Q = (\mathcal{R}, \mathcal{S}, \mathcal{A})$, where $\mathcal{R}, \mathcal{S}, \mathcal{A}$ denote the randomizer, shuffler, and analyzer, respectively. The protocol output is given by

$$\mathcal{P}_Q(D) \leftarrow \mathcal{A} \circ \mathcal{S} \circ \mathcal{R}(D) = \mathcal{A}(\mathcal{S}(\mathcal{R}(x_1), \cdots, \mathcal{R}(x_n))).$$

$\mathcal{S} \circ \mathcal{R}(D)$ satisfies (ε, δ)-DP.

Multiple shuffler setting. Some shuffle-DP protocols consider the setting where there exist multiple shufflers $\mathcal{S}$ [3]. Here, we have a set of shufflers $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m\}$ and n users. Each user i can send messages to one or more shufflers $\mathcal{S}$.

Shuffle-DP protocol parameters. By convention, a shuffle-DP protocol has four parameters: the privacy budgets ε and δ, the error parameter β, and the data size n. The parameter β specifically controls the probability of exceeding the stated error bound. More precisely, we use $\mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon, \delta, \beta)$ to denote the ℓ_p-norm error of the protocol $\mathcal{P}_Q$. Specifically, for any input dataset D, with privacy budgets ε, δ, with probability at least $1 - \beta$, we have

$$\left\|\mathcal{P}_Q(D) - Q(D)\right\|_{\ell_p} \leq \mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon, \delta, \beta).$$

Notably, the error of existing shuffle-DP protocols for union-preserving queries does not depend on the data size n. The exception happens in non-union-preserving queries, such as distinct count [13], which are beyond the scope of this work. We quantify the communication cost in two parts. First, $\mathrm{Msg}(\mathcal{P}_Q, \varepsilon, \delta, n)$ denotes the expected number of messages each user sends in protocol $\mathcal{P}$ with a group of n users, i.e., $\mathbb{E}[|\mathcal{R}(x_i)|]$. Second, $\mathrm{Bit}(\mathcal{P}_Q, \varepsilon, \delta, U, n)$ denotes the number of bits required to encode each message. For all existing shuffle-DP protocols, it is observed that $\mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon, \delta, \beta)$, $\mathrm{Msg}(\mathcal{P}_Q, \varepsilon, \delta, n)$, and $\mathrm{Bit}(\mathcal{P}_Q, \varepsilon, \delta, U, n)$ are all polynomial functions of their respective parameters. This observation will simplify the analysis in this paper. For example, $\mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon/c, \delta/c, \beta/c) = O\big(\mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon, \delta, \beta)\big)$ for any constant c.

Example 2.1 (Bit Counting Problem Q_{count})**.** We take the bit counting problem (Q_{count}) as an example. $\mathrm{Range}(Q_{\text{count}}, n) = \{0, 1, \ldots, n\}$, and the diameter $\gamma_{\ell_1}(Q_{\text{count}}, n) = n$. The SOTA solution [30] lets each user i send a single "+1" bit if $x_i = 1$. In addition, to preserve DP, each user samples a **random** but bounded number of noisy "+1" and "−1" messages. The protocol [30] has the following properties:

Lemma 2.4 (Bit Counting Protocol $\mathcal{P}_{Q_{\text{count}}}$ [30])**.** Given $\varepsilon > 0, \delta > 0$ and n, [30] solves the bit counting problem under (ε, δ)-shuffle-DP with the following guarantees:

- Utility guarantee: $\mathrm{Error}_{\ell_1}(\mathcal{P}_{Q_{\text{count}}}, \varepsilon, \delta, \beta) = O\left(\frac{1}{\varepsilon} \log \frac{1}{\beta}\right)$;
- Communication cost: $\mathrm{Msg}(\mathcal{P}_{Q_{\text{count}}}, \varepsilon, \delta, n) = 1 + O\left(\frac{\log(1/\delta)}{\varepsilon n}\right)$ and $\mathrm{Bit}(\mathcal{P}_{Q_{\text{count}}}, \varepsilon, \delta, U, n) = 2$.

2.4 Poisoning Attacks

Under the shuffle-DP model, we identify two types of adversaries:

(1) *Semi-honest analyzer*: it follows the protocol but attempts to infer private information from shuffled messages.

(2) *Poisoning attackers*: they, a.k.a. corrupted users, actively manipulate their inputs to disrupt the protocol.

Any protocol satisfying shuffle-DP effectively defends against semi-honest analyzers. However, poisoning attackers pose additional challenges that require further robustness measures. Below, we detail their objectives and behaviors.

2.4.1 Capacity. The number of corrupted users, denoted by k, is bounded by a public parameter $\hat{k}$. While the exact value of k is unknown, we assume that $\hat{k}$ is known to all parties. In MPC, $\hat{k}$ is typically set to a large value such as $n/2 - 1$, referred to as the "honest-majority" model, where the adversary can corrupt strictly fewer than half of the parties. However, in the shuffle-DP setting, $\hat{k}$ must be significantly smaller. Consider the bit counting task: if k corrupted users alter their inputs from 0 to 1, a shift between valid inputs, the final result may shift by as much as k. Since these users follow the protocol correctly, it is impossible for the analyzer to detect their manipulation. As a result, if k is too large, any defense becomes ineffective. Therefore, to ensure meaningful results, we set $\hat{k}$ to be polylogarithmic in n.

2.4.2 Objectives and behaviors.

Privacy attack: Exempt from generating randomness. Recall that shuffle-DP relies on collective randomness generation across all n users. Therefore, a privacy-breaking attack occurs when poisoning attackers drop out in generating randomness. For example, in the case of bit counting, if k users drop out, the resulting privacy protection is degraded compared to the intended level. Robust shuffle-DP protocols [1] mitigate this by having honest users amplify their noise by $n/(n - \hat{k})$, effectively safeguarding privacy against malicious behavior.

Utility attack 1: Alter input data. As mentioned before, the most straightforward utility attack is to alter the input data. In general, a single corrupted user can introduce at most $\gamma_{\ell_p}(Q, 1)$ error to the final result through this attack. Consider the bit counting query as an example. As described in Example 2.1, existing shuffle-DP protocols such as [29, 30] only allow users to send bits with value "+1" or "−1". Hence, any message outside this domain can be easily detected and filtered by the analyzer. As a result, a fake input can affect the result by at most $O(1)$. Similarly, for sum estimation, the impact of a fake input is bounded by $O(U)$, the size of the domain. Such an attack cannot be fundamentally defended, as corrupted users still provide syntactically valid inputs that conform to the protocol. Fortunately, since $\hat{k}$ is at most a polylogarithmic function of n, the overall impact of such malicious actions on the final result remains limited. Above all, this subtle attack is not our focus.

Utility attack 2: Send excessive messages. A more overt and consequential attack is to send excessive messages, potentially leading to unbounded error. Still using the bit counting query as an example, in protocols [29, 30], the "+1" and "−1" messages sent by users consist of two components: (i) the actual messages from users holding bit "1", and (ii) the DP noise composed of both "+1" and "−1" messages added to ensure privacy. Due to the shuffler's anonymity, the analyzer cannot distinguish between these two components. As a result, a corrupted user can introduce $O(n)$ error by injecting $O(n)$ noisy "+1" messages. This attack is particularly undetectable when the number of true "+1" messages is small, as the plausible range of query results is already $O(n)$. Several studies [16] have identified and tackled this "flooding" attack by fixing both the number of messages each user can send and the output domain of each message. However, these methods exhibit a critical limitation: they are often tailored to specialized tasks, lacking a unified framework applicable to more general queries.

2.4.3 Limitation of naive baselines. In Section 1.1, we mentioned several straightforward defense strategies against poisoning attacks. Here we show why these solutions do not work in practice.

Trusted shuffler with authentication and message-integrity checks. One might consider performing authentication and integrity checks at the shuffler to prevent malicious users from injecting unlimited messages. However, this is incompatible with the standard shuffle-DP model, where the shuffler only permutes messages; granting it additional functions changes the trust model and brings the model closer to central-DP. In practice, shufflers are often implemented via anonymous communication channels (e.g., mix networks or onion routing), making such authentication unrealistic. More importantly, even if additional integrity checks were available, they would only be effective in limited scenarios—specifically, when each user sends a fixed number of messages and each message originates from a bounded domain. As discussed in the introduction, these assumptions do not always hold. For example in the BBGN(IKOS) protocol [3] where each message is drawn from a range much larger than nU, a corrupted user can submit extremely large but valid values that pass verification yet introduce $O(nU)$ error. Hence, authentication and integrity checks may work for a narrow class of protocols with fixed message counts and well-bounded domains, but they cannot provide a general defense framework.

Local-DP + shuffler. First, the local-DP model inherently resists poisoning attacks, as each user independently perturbs their data and integrity can be verified at the user side. However, it suffers from an $O(\sqrt{n})$ utility loss compared with central-DP and shuffle-DP [2]. Shuffle-DP alleviates this by introducing *privacy amplification by shuffling*, which reduces the effective privacy budget ε to $O(\varepsilon_0/\sqrt{n})$, where ε_0 is the original local privacy budget and ε is the resulting privacy guarantee after shuffling [2, 25]. Therefore, a common idea is to apply a local-DP mechanism followed by random shuffling. However, the privacy amplification from shuffling is significant only under a stringent privacy regime. To achieve $\varepsilon = O(1/\sqrt{n})$, the local mechanism must operate with a constant $\varepsilon_0 = \Theta(1)$. In contrast, for the typical setting of $\varepsilon = O(1)$ used in most DP works, the required local privacy budget scales as $\varepsilon_0 = \Theta(\log n)$, resulting in the same error as the original local-DP protocol. This fundamental limitation explains why shuffle-DP has emerged as an independent and active research area, rather than as a simple combination of local-DP and shuffling. Moreover, because the shuffler hides individual contributions, this approach can at best support detection but cannot recover meaningful results once corrupted users inject large noise, as any noise added before shuffling becomes indistinguishably mixed afterward.

Two-round strategy. The two-round strategy, in which we first detect and exclude corrupted users and then re-run the query, fails against adaptively corrupted users who behave honestly in the first round to avoid detection and launch their attack in the second round. For example, in a bit counting query, the corrupted users can follow the protocol in the first round by sending a reasonable number of "1" messages, thereby avoiding identification as malicious. However, in the second round, they can inject an excessive number of "1" messages to distort the result by $O(n)$, resulting in the same error as if no defense were applied.

3 A Straw-man Solution

We first propose a simple *Single-User Shuffle-DP* (SUSDP) protocol as a straw-man solution. Although it increases the error by a factor of n compared to the original shuffle-DP protocol, thereby resulting in poor utility, it serves as a useful starting point and offers key insights that inform the design of our final protocol.

Given a shuffle-DP protocol $\mathcal{P}_Q = (\mathcal{R}, \mathcal{S}, \mathcal{A})$, SUSDP requests each user independently to perturb their own data, thus ensuring it remains unaffected by attackers and avoiding the difficulties of

Algorithm 1: Randomizer of SUSDP

Public Parameters: ε, δ, n
Input: $x_i, \mathcal{P}_Q = (\mathcal{R}, \mathcal{S}, \mathcal{A})$
$Y_i \leftarrow \mathcal{R}(x_i; \varepsilon, \delta, 1)$;
Send Y_i to shuffler $\mathcal{S}_i$;

Algorithm 2: Analyzer of SUSDP

Public Parameters: $\varepsilon, \delta, \beta, n$
Input: $\{Z_i = Y_i\}_i, \mathcal{P}_Q = (\mathcal{R}, \mathcal{S}, \mathcal{A})$
for $i \leftarrow 1$ **to** n **do**
  $\tilde{Q}_i \leftarrow \mathcal{A}(Z_i; \varepsilon, \delta, \beta/n, 1)$;
  if $\mathrm{dis}_{\ell_p}(\tilde{Q}_i, \mathrm{Range}(Q, 1)) > \mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon, \delta, \beta/n)$ **then**
    $\tilde{Q}_i \leftarrow 0$;
return $\sum_i \tilde{Q}_i$

detection under anonymization: Each user i privates their data x_i using a local randomizer $\mathcal{R}$, and then sends the messages Y_i to its corresponding shuffler $\mathcal{S}_i$. Finally, for each user i, the analyzer $\mathcal{A}$ generates an estimated result $\tilde{Q}_i$ and evaluates its reasonability by checking whether

$$\mathrm{dis}_{\ell_p}(\tilde{Q}_i, \mathrm{Range}(Q, 1)) \leq \mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon, \delta, \beta/n).$$

If not, the user is flagged as a poisoning attacker.[2] The final result is obtained by aggregating all valid outputs. The details of the randomizer and analyzer are presented in Algorithm 1 and 2, respectively.

At first glance, this protocol appears to defend against both privacy and utility attacks, as each user employs a distinct shuffler and independently adds noise to their own data. However, as we use a multi-shuffler architecture, attackers may impersonate others and send messages to others' shufflers, referred to as *impersonation attacks*. A straightforward countermeasure involves the analyzer assigning a unique random ID to each shuffler, such that only authorized users (for that particular shuffler) possess this ID. More precisely,

(1) *Shuffler identification:* When $\mathcal{S}_j$ is instantiated, the analyzer generates an ID r_j (e.g., a random string) and shares it only with those users who are authorized to send messages to $\mathcal{S}_j$.
(2) *Message authorization:* Each message sent to $\mathcal{S}_j$ must carry the corresponding ID r_j. If a user without knowledge of r_j attempts to flood $\mathcal{S}_j$ with messages, the analyzer can immediately discard them as unauthorized.

This idea effectively addresses the issue of impersonation attacks, even in the presence of multiple attackers, as each attacker is restricted to sending messages only to the shuffler they are authorized to access. In the following sections, we apply this idea as a general solution for defending against impersonation attacks.

Theorem 3.1. Given any $\varepsilon > 0, \delta > 0, n \in \mathbb{Z}_+$ and $U \in \mathbb{Z}_+$, when there is only one corrupted user, for any union-preserving query Q, the SUSDP protocol achieves that:

- The messages received by the analyzer preserve (ε, δ)-DP;

[2]Note that our detection rule assumes all users are honest and treats outliers as honest as well. The confidence parameter β controls the success rate of this check, ensuring that with probability at least $1 - \beta$, an honest outlier will not be misclassified.

- With probability at least $1-\beta$, the total error is bounded by:

$$O\big(n \cdot \mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon, \delta, \beta/n)\big) \;+\; \gamma_{\ell_p}(Q, 1);$$

- In expectation, each user sends $\mathrm{Msg}(\mathcal{P}_Q, \varepsilon, \delta, 1)$ messages with each containing $\mathrm{Bit}(\mathcal{P}_Q, \varepsilon, \delta, U, 1)+O(\log n)$ bits.

Due to space limitations, we defer all proofs to Appendices A–E.[3] The communication and privacy analysis are straightforward. For utility, the intuition is as follows. If a corrupted user injects excessive noise and is detected, its result will be discarded, incurring an error of $\gamma_{\ell_p}(Q, 1)$. Otherwise, if the corrupted user manipulates its data and evades detection, the error is bounded by $O(\mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon, \delta, \beta/n)) + \gamma_{\ell_p}(Q, 1)$. In the worst case, the corrupted user thus contributes at most this amount of error, while the remaining $n-1$ benign users together introduce at most $O((n-1) \cdot \mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon, \delta, \beta/n))$ error.

Example 3.1 (SUSDP for Q_{count})**.** Following the bit counting example, we use $\mathcal{P}_{Q_{\text{count}}}$ [30] to initialize our SUSDP protocol. Since each user independently executes the $\mathcal{P}_{Q_{\text{count}}}$ procedure, the difference threshold is set to $O\big(\frac{1}{\varepsilon}\log\frac{n}{\beta}\big)$ according to Lemma 2.4. Under this setup, SUSDP protocol with $\mathcal{P}_{Q_{\text{count}}}$ achieves the error $O\big(\frac{\sqrt{n}}{\varepsilon}\log\frac{n}{\beta}\big)$[4], and in expectation, each user sends $O\big(\frac{\log(1/\delta)}{\varepsilon}\big)$ messages, with each message containing $O(\log n)$ bits.

4 Methodology

While the SUSDP protocol effectively mitigates the poisoning attacks, the resulting increase in error is suboptimal. This is because, in contrast to shuffle-DP which amortizes noise across all users for improved utility, each user in SUSDP independently adds noise. In this section, we present our general framework which can not only detect and recover poisoning attacks but also enjoy the error benefits in shuffle-DP.

We first introduce a *Block Shuffle-DP* (BSDP) protocol as a foundational defense mechanism, which has an error upgraded by a factor of $O(\sqrt{n})$ compared with the given shuffle-DP protocol, in Section 4.1. Then, we reduce the factor to $O(\log n)$ by *Hierarchical Shuffle-DP* (HSDP) protocol in Section 4.2. Finally, in Section 4.3, we propose optimizations to reduce the communication cost and then extend our solution to the multiple corrupted users setting.

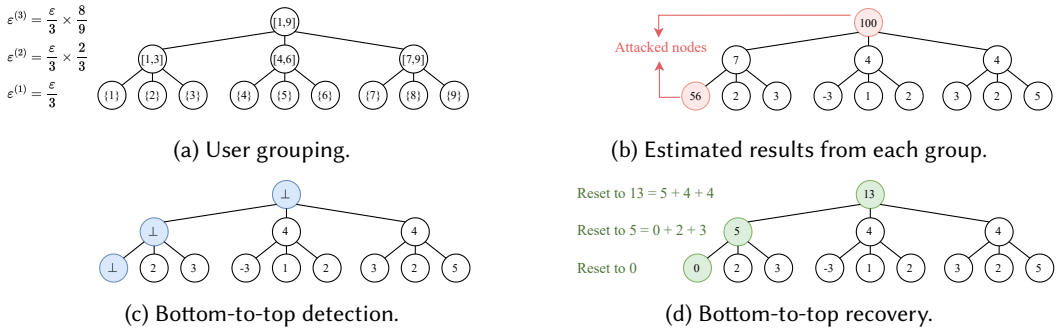
4.1 Block Shuffle-DP Protocol

The protocol operates on three levels. At the bottom level, we apply a shuffle-DP protocol for every single user (user-level). At the top level, we apply a shuffle-DP protocol across all users (output-level). Between this, we introduce an intermediate level, where users are partitioned into blocks of fixed size $\sqrt{n}$, and users in each block perform a shuffle-DP protocol (block-level).

To defend against poisoning attacks, detection is performed at all three levels. At the user-level, the detection evaluates the reasonability of each user's output. At the block-level, attacks can be identified by comparing each group's aggregated output with the sum of its members' user-level outputs and checking whether the discrepancy falls within a reasonable bound. Similarly, for the output-level, attacks are detected by comparing the aggregated output over all users with the sum of block-level outputs. The illustration of the block shuffle-DP protocol is shown in Fig. 1. For simplicity, we assume that n is a perfect square. The detailed protocol works as follows:

[3] https://drive.google.com/drive/folders/1oqBEyg0Uac1V6PMUYDEGV01GdlwRbSUv

[4] Directly applying Theorem 3.1 yields an error bound of $O\big(\frac{n}{\varepsilon}\log\frac{n}{\beta}\big)$. However, since the error of $\mathcal{P}_{Q_{\text{count}}}$ follows a Laplace distribution [30], we can leverage its concentration bound to obtain a tighter estimate.



(a) User grouping.

(b) Estimated results from each group.

(c) Bottom-to-top detection.

(d) Bottom-to-top recovery.

Fig. 1. Illustration of our block shuffle-DP protocol for bit counting with $n = 9$ users. A corrupted user (ID = 1) sends excessive messages at different levels.

Algorithm 3: Randomizer of BSDP

Public Parameters: ε, δ, n
Input: $x_i, \mathcal{P}_Q = (\mathcal{R}, \mathcal{S}, \mathcal{A})$
// User-Level Randomization
$Y_i^{(1)} \leftarrow \mathcal{R}\left(x_i; \frac{\varepsilon}{3}, \frac{\delta}{3}, 1\right)$;
Send $Y_i^{(1)}$ to shuffler $\mathcal{S}_i^{(1)}$;
// Block-Level Randomization
$Y_i^{(2)} \leftarrow \mathcal{R}\left(x_i; \frac{\varepsilon}{3} \cdot \frac{\sqrt{n}-1}{\sqrt{n}}, \frac{\delta}{3} \cdot \frac{\sqrt{n}-1}{\sqrt{n}}, \sqrt{n}\right)$;
Send $Y_i^{(2)}$ to shuffler $\mathcal{S}_b^{(2)}$.
// Output-Level Randomization
$Y_i^{(3)} \leftarrow \mathcal{R}\left(x_i; \frac{\varepsilon}{3} \cdot \frac{n-1}{n}, \frac{\delta}{3} \cdot \frac{n-1}{n}, n\right)$;
Send $Y_i^{(3)}$ to shuffler $\mathcal{S}^{(3)}$;

Randomizer. Each user i privates their data x_i using three local randomizers, each operating with a privacy budget of $\varepsilon/3$ and $\delta/3$. To ensure only a constant factor increase in total error over the original shuffle-DP protocol in the absence of attackers, we allocate $\beta/2$ to output-level query, and distribute the remaining equally among the $\sqrt{n} + n$ block-level and user-level queries. The first randomizer performs user-level randomization. The output of each user i is sent to their pre-assigned shuffler $\mathcal{S}_i^{(1)}$. The second randomizer performs block-level randomization. All users are partitioned into disjoint blocks of size $\sqrt{n}$, and each block jointly executes a shuffle-DP protocol. Specifically, the block b consists of users with ID in $\{(b-1)\sqrt{n}+1, \ldots, b\sqrt{n}\}$. Since the attacker may not contribute any noise, we need the remaining $\sqrt{n}-1$ honest users to collectively amortize the noise. Therefore, the privacy budget will be scaled by a factor of $(\sqrt{n}-1)/\sqrt{n}$. Each user sends their output to the block-level shuffler $\mathcal{S}_b^{(2)}$, which collects and shuffles messages from all users belonging to block g. The third randomizer performs output-level randomization, where all users jointly execute a shuffle-DP protocol. We also rescale the privacy budget by a factor of $(n-1)/n$. The output is sent to a central shuffler $\mathcal{S}^{(3)}$. The detailed procedure is shown in Algorithm 3.

Analyzer. After receiving the shuffled messages from the shufflers, denoted by $Z_i^{(1)}$, $Z_b^{(2)}$, and $Z^{(3)}$ for the user-, block-, and output-level outputs respectively, the analyzer invokes the given $\mathcal{A}$ on each message set to compute the estimated results, written as $\tilde{Q}_i^{(1)}$, $\tilde{Q}_b^{(2)}$, and $\tilde{Q}^{(3)}$, where i and

Algorithm 4: Analyzer of BSDP

Public Parameters: $\varepsilon, \delta, \beta, n$

Input: $\{Z_i^{(1)} = Y_i^{(1)}\}_i$, $\{Z_b^{(2)} = \cup_{i=(b-1)\cdot\sqrt{n}+1}^{b\cdot\sqrt{n}} Y_i^{(2)}\}_b$, $Z^{(3)} = \cup_i Y_i^{(3)}$, $\mathcal{P}_Q = (\mathcal{R}, \mathcal{S}, \mathcal{A})$

$\left(\varepsilon^{(1)}, \delta^{(1)}, \beta^{(1)}\right) \leftarrow \left(\frac{\varepsilon}{3}, \frac{\delta}{3}, \frac{\beta}{2(\sqrt{n}+n)}\right)$;

$\left(\varepsilon^{(2)}, \delta^{(2)}, \beta^{(2)}\right) \leftarrow \left(\frac{\varepsilon}{3} \cdot \frac{\sqrt{n}-1}{\sqrt{n}}, \frac{\delta}{3} \cdot \frac{\sqrt{n}-1}{\sqrt{n}}, \frac{\beta}{2(\sqrt{n}+n)}\right)$;

$\left(\varepsilon^{(3)}, \delta^{(3)}, \beta^{(3)}\right) \leftarrow \left(\frac{\varepsilon}{3} \cdot \frac{n-1}{n}, \frac{\delta}{3} \cdot \frac{n-1}{n}, \frac{\beta}{2}\right)$;

$\theta^{(r)} \leftarrow \mathrm{Error}_{\ell_p}\left(\mathcal{P}_Q, \varepsilon^{(r)}, \delta^{(r)}, \beta^{(r)}\right), \forall 1 \le r \le 3$;

// User-Level Detection

for $i \leftarrow 1$ **to** n **do**

  $\tilde{Q}_i^{(1)} \leftarrow \mathcal{A}(Z_i^{(1)}; \varepsilon^{(1)}, \delta^{(1)}, \beta^{(1)}, 1)$;

  if $\mathrm{dis}_{\ell_p}(\tilde{Q}_i^{(1)}, \mathrm{Range}(Q, 1)) > \theta^{(1)}$ **then**

    $\tilde{Q}_i^{(1)} \leftarrow \perp$;

// Block-Level Detection

for $b \leftarrow 1$ **to** $\sqrt{n}$ **do**

  $\tilde{Q}_b^{(2)} \leftarrow \mathcal{A}(Z_b^{(2)}; \varepsilon^{(2)}, \delta^{(2)}, \beta^{(2)}, \sqrt{n})$;

  if $\exists\ \tilde{Q}_i^{(1)} = \perp,\ i \in [(b-1)\cdot\sqrt{n}+1,\ b\cdot\sqrt{n}]$ **or** $\left|\tilde{Q}_b^{(2)} - \sum_{i=(b-1)\cdot\sqrt{n}+1}^{b\cdot\sqrt{n}} \tilde{Q}_i^{(1)}\right|_{\ell_p} > \sqrt{n}\cdot\theta^{(1)} + \theta^{(2)}$ **then**

    $\tilde{Q}_b^{(2)} \leftarrow \perp$;

// Output-Level Detection

$\tilde{Q}^{(3)} \leftarrow \mathcal{A}(Z^{(3)}; \varepsilon^{(3)}, \delta^{(3)}, \beta^{(3)}, n)$;

if $\exists\ \tilde{Q}_b^{(2)} = \perp,\ b \in [1, \sqrt{n}]$ **or** $\left|\tilde{Q}^{(3)} - \sum_b \tilde{Q}_b^{(2)}\right|_{\ell_p} > \sqrt{n}\cdot\theta^{(2)} + \theta^{(3)}$ **then**

  $\tilde{Q}^{(3)} \leftarrow \perp$;

// Recovery

for $i \leftarrow 1$ **to** n **do**

  if $\tilde{Q}_i^{(1)} = \perp$ **then**

    $\tilde{Q}_i^{(1)} \leftarrow 0$;

for $b \leftarrow 1$ **to** $\sqrt{n}$ **do**

  if $\tilde{Q}_b^{(2)} = \perp$ **then**

    $\tilde{Q}_b^{(2)} \leftarrow \sum_{i=(b-1)\cdot\sqrt{n}+1}^{b\cdot\sqrt{n}} \tilde{Q}_i^{(1)}$;

if $\tilde{Q}^{(3)} = \perp$ **then**

  $\tilde{Q}^{(3)} \leftarrow \sum_b \tilde{Q}_b^{(2)}$;

return $\tilde{Q}^{(3)}$

b index users and blocks, respectively. After collecting all results, the analyzer performs detection and recovery:

- *Detection.* The detection is performed in a bottom-to-top manner. At the user-level, analyzer checks whether each $\tilde{Q}_i^{(1)}$ falls within a reasonable range and flags abnormal outputs by setting $\tilde{Q}_i^{(1)} \leftarrow \perp$. At the block-level, analyzer invalidates $\tilde{Q}_b^{(2)}$ if any member in block b has

been flagged at user-level; otherwise, analyzer compares $\tilde{Q}_b^{(2)}$ with the sum of $\tilde{Q}_i^{(1)}$ in the block. Similarly, at the output-level, analyzer checks whether $\tilde{Q}^{(3)}$ is consistent with the sum of all valid block-level outputs.

- *Recovery.* The recovery proceeds in a bottom-to-top manner as well. For each invalid block-level output $\tilde{Q}_b^{(2)}$, the analyzer recovers it by summing all unflagged user-level outputs $\tilde{Q}_i^{(1)}$ of users within the block. After reconstructing all block-level results, the analyzer attempts to recover the output-level result. If the aggregated output-level result $\tilde{Q}^{(3)}$ is valid, it is returned as the final result. Otherwise, the analyzer aggregates all block-level outputs $\tilde{Q}_b^{(2)}$ (including those recovered from user-level results) to produce the final result.

The detailed procedure is shown in Algorithm 4.

Theorem 4.1. Given any $\varepsilon > 0, \delta > 0, n \in \mathbb{Z}_+$ and $U \in \mathbb{Z}_+$, when there is only one corrupted user, for any union-preserving query Q, the BSDP protocol achieves that:

- The messages received by the analyzer preserve (ε, δ)-DP;
- With probability at least $1 - \beta$, the total error is bounded by:

$$O\big(\sqrt{n} \cdot \mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \varepsilon, \delta, \beta/n)\big) + \gamma_{\ell_p}(Q, 1)$$

- In expectation, each user sends $O\big(\mathrm{Msg}(\mathcal{P}_Q, \varepsilon, \delta, 1)\big)$ messages, each containing $O\big(\mathrm{Bit}(\mathcal{P}_Q, \varepsilon, \delta, U, 1) + \log n\big)$ bits.

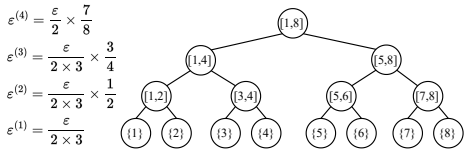
Example 4.1 (BSDP for Q_{count})**.** We initialize the BSDP protocol using $\mathcal{P}_{Q_{\text{count}}}$ from [30]. When there are no attackers, BSDP achieves an error of $O\big(\frac{1}{\varepsilon}\log\frac{1}{\beta}\big)$, which is asymptotically the same as the original $\mathcal{P}_{Q_{\text{count}}}$. In the presence of a single corrupted user, the error changes to $O\big(\frac{n^{1/4}}{\varepsilon}\log\frac{n}{\beta}\big)$[5]. Each user sends $O\big(\frac{\log(1/\delta)}{\varepsilon}\big)$ messages in expectation, each message containing $O(\log n)$ bits.

4.2 Hierarchical Shuffle-DP Protocol

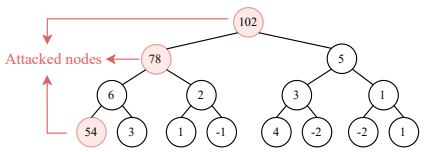
Recall that in the block shuffle-DP protocol, recovering the block-level or output-level result requires aggregating $O(\sqrt{n})$ results from the lower level, which contributes an error term $O(\sqrt{n})$. Our idea is to introduce additional levels, allowing us to iteratively combine and verify smaller blocks, thereby limiting the attacker's ability to disrupt the protocol. Building on this insight, we propose an advanced multi-level block protocol, referred to as the *Hierarchical Shuffle-DP* (HSDP) protocol, which is structured as a binary tree. This hierarchical design reduces the number of lower-level results required to reconstruct an upper-level output, thereby enabling more precise control over error accumulation across levels.

Specifically, at the bottom level, each user individually applies the single-user shuffle-DP protocol. Then, in each upper level, two groups from the previous level are merged to form a new group, continuing recursively until all users are consolidated into a single group at the top level. For simplicity, we assume that n is a power of 2. There are $\log n + 1$ levels, where at level $r \in \{1, 2, \ldots, \log n + 1\}$, each group has size 2^{r-1}. Each group performs a shuffle-DP protocol. On the analyzer's side, detection and recovery are carried out in a manner similar to BSDP, where we check the reasonability of each group's result and perform recovery using the results of its subgroups from the lower level if needed. This hierarchical process is illustrated in Fig. 2. Precisely, the protocol works as follows:

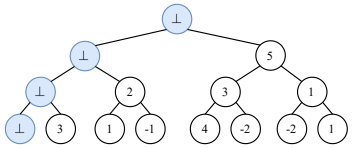
[5]We achieve this error bound by applying Theorem 4.1 and the concentration bound. Furthermore, we replace the terms $\sqrt{n}$ with $n^{1/4}$ in lines 11 and 14 of Algorithm 4.



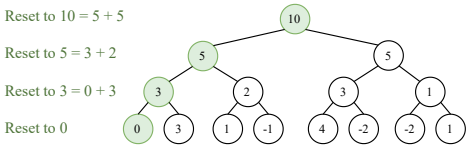
(a) Binary tree structure of user grouping.



(b) Estimated results from each group.



(c) Bottom-to-top detection.



(d) Bottom-to-top recovery.

Fig. 2. Illustration of our hierarchical shuffle-DP protocol for bit counting with $n = 8$ users. A corrupted user (ID = 1) sends excessive messages at different levels.

Algorithm 5: Randomizer of HSDP

Public Parameters: ε, δ, n
Input: $x_i, \mathcal{P}_Q = (\mathcal{R}, \mathcal{S}, \mathcal{A})$

$Y_i^{(1)} \leftarrow \mathcal{R}\big(x_i; \frac{\varepsilon}{2\log n}, \frac{\delta}{2\log n}, 1\big)$;
Send $Y_i^{(1)}$ to shuffler $\mathcal{S}_i^{(1)}$;
for $r \leftarrow 2$ **to** $\log n$ **do**
  $Y_i^{(r)} \leftarrow \mathcal{R}\big(x_i; \frac{\varepsilon}{2\log n} \cdot \frac{2^{r-1}-1}{2^{r-1}}, \frac{\delta}{2\log n} \cdot \frac{2^{r-1}-1}{2^{r-1}}, 2^{r-1}\big)$;
  Send $Y_i^{(r)}$ to shuffler $\mathcal{S}_g^{(r)}$, where $g = \lceil i/2^{r-1} \rceil$;
$Y_i^{(\log n+1)} \leftarrow \mathcal{R}\big(x_i; \frac{\varepsilon}{2} \cdot \frac{n-1}{n}, \frac{\delta}{2} \cdot \frac{n-1}{n}, n\big)$;
Send $Y_i^{(\log n+1)}$ to shuffler $\mathcal{S}_1^{(\log n+1)}$;

Randomizer. Each user i privatizes their data x_i using $\log n + 1$ local randomizers, each corresponding to one level of the hierarchical protocol. We allocate half of the privacy budget, i.e., $\varepsilon/2$ and $\delta/2$, to the randomizer at the top level, and distribute the remaining half equally among the lower $\log n$ levels. For β, we allocate $\beta/2$ to the query at the top level, and split the remaining half equally across the $2n - 2$ queries at lower levels. This allocation ensures that, in the absence of any attacker, the overall error increases by at most a constant factor compared to the given shuffle-DP protocol. The first randomizer is used to perform user-level randomization with its output sent to the shuffler $\mathcal{S}_i^{(1)}$ assigned to user i. The r-th randomizer with $r > 1$ performs a shuffle-DP over a group g, which consists of users with ID in $\{(g-1) \cdot 2^{r-1} + 1, \ldots, g \cdot 2^{r-1}\}$. The output is sent to shuffler $\mathcal{S}_g^{(r)}$. To defend against the case where the attacker contributes no noise, similar to the block approach, we rescale the privacy budget by a factor of $(2^{r-1} - 1)/2^{r-1}$ at each level $r > 1$. The detailed procedure is shown in Algorithm 5.

Analyzer. After receiving shuffled messages $Z_g^{(r)}$ from the shuffler $\mathcal{S}_g^{(r)}$, the analyzer invokes the given $\mathcal{A}$ to compute the estimated results $\tilde{Q}_g^{(r)}$. After collecting all results for every $r \in \{1, \ldots, \log n + 1\}$ and $g \in \{1, \ldots, n/2^{r-1}\}$, the analyzer performs detection and recovery:

- *Detection.* Similar to the block shuffle-DP protocol, the detection is split into two primary stages. At the user-level ($r = 1$), analyzer checks if each $\tilde{Q}_i^{(1)}$ falls within a reasonable range and flags abnormal outputs by setting $\tilde{Q}_i^{(1)} \leftarrow \perp$. At higher levels, the analyzer proceeds from bottom to top: for each group at level r, check its result with the sum of results from its two subgroups at level $r-1$. If any subgroup was previously flagged as invalid ($\tilde{Q}_{2g-1}^{(r-1)}$ or $\tilde{Q}_{2g}^{(r-1)} = \perp$), the current group $\tilde{Q}_g^{(r)}$ is also marked invalid. Otherwise, the analyzer compares $\tilde{Q}_g^{(r)}$ against the sum of its two subgroups $\tilde{Q}_{2g-1}^{(r-1)} + \tilde{Q}_{2g}^{(r-1)}$. The group $\tilde{Q}_g^{(r)}$ is marked invalid if the difference exceeds the cumulative error bound, that is, the sum of the error bound from the current group and its subgroups.
- *Recovery.* The recovery process proceeds in a bottom-up manner. At the user-level, any invalid result is set to 0. From the second level onward, if a result is marked as invalid (i.e., equal to $\perp$), it is recovered by aggregating the outputs of its two subgroups from the lower level: $\tilde{Q}_g^{(r)} \leftarrow \tilde{Q}_{2g-1}^{(r-1)} + \tilde{Q}_{2g}^{(r-1)}$. After completing the recovery, the final output is $\tilde{Q}_1^{(\log n+1)}$.

The detailed procedure is shown in Algorithm 6.

Theorem 4.2. Given any $\varepsilon > 0, \delta > 0, n \in \mathbb{Z}_+, U \in \mathbb{Z}_+$, when there is only one corrupted user, for any union-preserving query Q, the HSDP protocol achieves that:

- The messages received by the analyzer preserve (ε, δ)-DP;
- With probability at least $1-\beta$, the total error is bounded by:

$$O\Big(\log n \cdot \mathrm{Error}_{\ell_p}\big(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, \frac{\beta}{n}\big)\Big) + \gamma_{\ell_p}(Q, 1);$$

- In expectation, each user sends $O\left(\log n \cdot \mathrm{Msg}(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, 1)\right)$ messages, each containing $O\big(\mathrm{Bit}(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, U, 1) + \log n\big)$ bits.

Intuition of our enhancement over SUSDP and BSDP. The key is to strike a balance between utility and robustness. Shuffle-DP achieves high utility by amortizing noise across all users, but its strong anonymity makes recovery infeasible under poisoning attacks. SUSDP goes to the opposite extreme: each user uses a dedicated shuffler, allowing the analyzer to observe and filter individual outputs. However, this degenerates to local-DP, where aggregating n independently noised reports incurs $O(n)$ error. BSDP takes a step toward balance with a three-level structure, reducing noise accumulation to $O(\sqrt{n})$ by grouping users into blocks of size $\sqrt{n}$. HSDP further improves this trade-off through a hierarchical structure, offering two key benefits. (i) Utility: with only O($\log n$) levels, the final output aggregates a logarithmic number of independently noised outputs, leading to O($\log n$) noise accumulation. (ii) Robustness: detection begins at lower-level groups where adversarial impact is limited, and recursively validates higher-level outputs against their subgroups, thereby bounding the influence of corrupted users throughout the hierarchy.

Example 4.2 (HSDP for Q_{count})**.** We instantiate the HSDP protocol using $\mathcal{P}_{Q_{\text{count}}}$ [30]. When there are no attackers, HSDP achieves an error of $O\big(\frac{1}{\varepsilon}\log\frac{1}{\beta}\big)$, which is asymptotically the same as the original $\mathcal{P}_{Q_{\text{count}}}$. In the presence of a single corrupted user, the error increases to $O\big(\frac{\log^2 n}{\varepsilon}\log\frac{n}{\beta}\big)$. In expectation, each user sends $O\big(\frac{\log n \cdot \log(1/\delta)}{\varepsilon}\big)$ messages, each containing $O(\log n)$ bits.

4.3 Optimization and Extension

So far, the hierarchical shuffle-DP protocol has successfully reduced the error overhead of defending against poisoning attacks to a polylogarithmic level. In this section, we further propose a solution to reduce communication costs and extend the protocol to handle multiple corrupted users.

Algorithm 6: Analyzer of HSDP

Public Parameters: $\varepsilon, \delta, \beta, n$

Input: $\{Z_g^{(r)} = \cup_{i=(g-1)\cdot 2^{r-1}+1}^{g\cdot 2^{r-1}} Y_i^{(r)}\}_{r,g}, \mathcal{P}_Q = (\mathcal{R}, \mathcal{S}, \mathcal{A})$

$\left(\varepsilon^{(r)}, \delta^{(r)}, \beta^{(r)}\right) \leftarrow \begin{cases} \left(\frac{\varepsilon}{2\log n}, \frac{\delta}{2\log n}, \frac{\beta}{2(2n-2)}\right) & \text{if } r = 1 \\ \left(\frac{\varepsilon}{2\log n}\cdot\frac{2^{r-1}-1}{2^{r-1}}, \frac{\delta}{2\log n}\cdot\frac{2^{r-1}-1}{2^{r-1}}, \frac{\beta}{2(2n-2)}\right) & \text{if } 2 \le r \le \log n \\ \left(\frac{\varepsilon}{2}, \frac{\delta}{2}, \frac{\beta}{2}\right) & \text{if } r = \log n + 1 \end{cases}$

$\theta^{(r)} \leftarrow \mathrm{Error}_{\ell_p}\left(\mathcal{P}_Q, \varepsilon^{(r)}, \delta^{(r)}, \beta^{(r)}\right), \forall 1 \le r \le \log n + 1;$

// Detection in the Bottom Level

for $i \leftarrow 1$ **to** n **do**

  $\tilde{Q}_i^{(1)} \leftarrow \mathcal{A}\left(Z_i^{(1)};\ \varepsilon^{(1)},\ \delta^{(1)},\ \beta^{(1)},\ 1\right);$

  if $\mathrm{dis}_{\ell_p}\left(\tilde{Q}_i^{(1)}, \mathrm{Range}(Q,1)\right) > \theta^{(1)}$ **then**

    $\tilde{Q}_i^{(1)} \leftarrow \perp;$

// Detection in Upper Levels

for $r \leftarrow 2$ **to** $\log n + 1$ **do**

  for $g \leftarrow 1$ **to** $n/2^{r-1}$ **do**

    $\tilde{Q}_g^{(r)} \leftarrow \mathcal{A}\left(Z_g^{(r)};\ \varepsilon^{(r)}, \delta^{(r)}, \beta^{(r)}, 2^{r-1}\right);$

    if $\left(\tilde{Q}_{2g-1}^{(r-1)} = \perp\right) \vee \left(\tilde{Q}_{2g}^{(r-1)} = \perp\right)$ **or** $\left|\tilde{Q}_g^{(r)} - \left(\tilde{Q}_{2g-1}^{(r-1)} + \tilde{Q}_{2g}^{(r-1)}\right)\right|_{\ell_p} > 2\cdot\theta^{(r-1)} + \theta^{(r)}$ **then**

      $\tilde{Q}_g^{(r)} \leftarrow \perp;$

// Recovery

for $i \leftarrow 1$ **to** n **do**

  if $\tilde{Q}_i^{(1)} = \perp$ **then**

    $\tilde{Q}_i^{(1)} \leftarrow 0;$

for $r \leftarrow 2$ **to** $\log n + 1$ **do**

  for $g \leftarrow 1$ **to** $n/2^{r-1}$ **do**

    if $\tilde{Q}_g^{(r)} = \perp$ **then**

      $\tilde{Q}_g^{(r)} \leftarrow \tilde{Q}_{2g}^{(r-1)} + \tilde{Q}_{2g-1}^{(r-1)};$

return $\tilde{Q}_1^{(\log n+1)}$

4.3.1 Reducing communication cost. Using bit counting as an example: each user sends $O\left(\frac{\log n\cdot\log(1/\delta)}{\varepsilon}\right)$ messages. The $\log n$ factor arises because each user participates in $\log n$ levels of the shuffle-DP protocols. At lower levels, where group sizes are small, each user must contribute a larger number of noise messages to ensure sufficient privacy. Therefore, a natural idea is to increase the group size at the lower levels and reduce the number of levels.

Specifically, if we change the first-level group size from 1 to λ, then each user's overall communication cost changes to

$$\sum_{i=\log\lambda}^{\log n} 1 + O\left(\frac{\log n\cdot\log(1/\delta)}{\varepsilon\cdot 2^i}\right) = \log\frac{n}{\lambda} + O\left(\frac{\log n\cdot\log(1/\delta)}{\varepsilon\lambda}\right),$$

indicating a reduction in the total number of messages as λ grows.

However, increasing λ leads to a significant increase in error, because the corrupted user can destroy the result of an entire group of size λ. In the extreme case where $\lambda = n$, it will degrade into traditional shuffle-DP protocol. Therefore, the choice of λ involves a trade-off between communication cost and accuracy.

In our optimization, we set $\lambda = \log n \cdot \log(1/\delta)$. Similar to the hierarchical shuffle-DP protocol, we allocate half of the privacy budget to the top level, and distribute the remaining half equally among the lower $\log \frac{n}{\lambda}$ levels. The detection and recovery processes are the same, except that we treat every λ users as a group and need to check whether $\mathrm{dis}_{\ell_p}(\tilde{Q}_i^{(1)}, \mathrm{Range}(Q, \lambda)) > \theta^{(1)}$ at the first level.

Theorem 4.3. Given any $\varepsilon > 0,\ \delta > 0,\ n \in \mathbb{Z}_+,\ U \in \mathbb{Z}_+, \lambda = \log n \cdot \log(1/\delta)$, when there is only a corrupted user, for any union-preserving query Q, the *Optimized HSDP* (OHSDP) protocol achieves the following:

- The messages received by the analyzer preserve (ε, δ)-DP;
- With probability at least $1 - \beta$, the total error is bounded by:

$$O\Big(\log n \cdot \mathrm{Error}_{\ell_p}\big(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, \frac{\beta}{n}\big)\Big) + \gamma_{\ell_p}(Q, \log n \cdot \log(1/\delta));$$

- In expectation, each user sends

$$O\Big(\log n \cdot \mathrm{Msg}(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, \log n \cdot \log(1/\delta))\Big)$$

  messages, each containing $O\big(\mathrm{Bit}(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, U, \log n \cdot \log(1/\delta)) + \log n\big)$ bits.

Example 4.3 (OHSDP for Q_{count})**.** We instantiate the OHSDP protocol using $\mathcal{P}_{Q_{\text{count}}}$ [30]. When there are no attackers, the protocol achieves an error of $O\big(\frac{1}{\varepsilon}\log\frac{1}{\beta}\big)$, which is asymptotically the same as $\mathcal{P}_{Q_{\text{count}}}$. In the presence of a single corrupted user, the error increases to $O\big(\frac{\log^2 n}{\varepsilon}\log\frac{n}{\beta}\big)$. In expectation, each user sends $O\big(\log n\big)$ messages, with each message containing $O(\log n)$ bits.

4.3.2 Handling multiple corrupted users. Consider the case where the poisoning attacker can corrupt multiple users. To ensure privacy, since in one group there are at most $\hat{k}$ corrupted users, the privacy budget must be scaled by a factor of $(c - \hat{k})/c$, where c denotes the group size. However, if c is too small, the privacy budget will be split by at most k, thus significantly degrading utility.

To address this issue, we require each group to have strictly more honest members than attackers. Recall that $\hat{k}$ is polylogarithmic in n, we choose the first-level size $\lambda > 2\hat{k}$ to ensure that the lack of noise from attackers cannot compromise the privacy of honest users. For simplicity, we still set $\lambda = \log n \cdot \log(1/\delta)$ with the assumption that $\hat{k} < \log n \cdot \log(1/\delta)/2$. It is trivial to see that our detection and recovery mechanisms naturally support this scenario to address detection and recovery in the multiple corrupted users setting. The primary difference lies in the utility guarantee, since multiple corrupted users can collectively affect a larger number of results.

Theorem 4.4. Given any $\varepsilon > 0,\ \delta > 0,\ n \in \mathbb{Z}_+,\ U \in \mathbb{Z}_+, \lambda = \log n \cdot \log(1/\delta)$, when there are k corrupted users ($k \le \hat{k} < \log n \cdot \log(1/\delta)/2$), for any union-preserving query Q, the OHSDP protocol achieves the following:

- The messages received by the analyzer satisfy (ε, δ)-DP;
- With probability at least $1 - \beta$, the total error is bounded by

$$O\Big(k \log n \cdot \mathrm{Error}_{\ell_p}\big(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, \frac{\beta}{n}\big)\Big) + \gamma_{\ell_p}\big(Q, k \cdot \log n \cdot \log(1/\delta)\big);$$

- In expectation, each user sends

$$O\left(\log n \cdot \mathrm{Msg}\big(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, \log n \cdot \log(1/\delta)\big)\right)$$

messages, each containing $O\Big(\mathrm{Bit}\big(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, U, \log n \cdot \log(1/\delta)\big) + \log n\Big)$ bits.

Remark 1. Our protocol can be easily extended to the case where $\hat{k}$ exceeds $\log n \cdot \log(1/\delta)/2$ by setting $\lambda = 2\hat{k}$ feasibly. The error bound becomes $O\big(k \log n \cdot \mathrm{Error}_{\ell_p}(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, \frac{\beta}{n})\big) + \gamma_{\ell_p}(Q, 2k\hat{k})$ with probability at least $1-\beta$, and in expectation, each user sends $O\big(\log n \cdot \mathrm{Msg}(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, \hat{k})\big)$ messages, each containing $O\big(\mathrm{Bit}(\mathcal{P}_Q, \frac{\varepsilon}{\log n}, \frac{\delta}{\log n}, U, \hat{k}) + \log n\big)$ bits.

Example 4.4 (OHSDP for Q_{count} in Multiple Corrupted Users Setting)**.** We instantiate the OHSDP protocol using $\mathcal{P}_{Q_{\text{count}}}$ [30], which achieves an error of $O\big(\frac{k \cdot \log^2 n}{\varepsilon} \cdot \log \frac{n}{\beta} + k \cdot \log n \cdot \log(1/\delta)\big)$. In expectation, each user sends $O(\log n)$ messages, with each message containing $O(\log n)$ bits.

5 Application

In this section, we apply our OHSDP framework in three union-preserving queries: summation, frequency estimation, and range counting, as defined in Section 2.1, and further explore its extension to complex queries such as k-selection and OLAP.

Summation. For the summation problem, we introduce and integrate two SOTA solutions respectively. The first protocol, IKOS-based BBGN [3], improved from the IKOS exact summation protocol [37], achieves central-DP error of $O\big(\frac{U}{\varepsilon} \log \frac{1}{\beta}\big)$. Each of n users is expected to send $O\big(1 + \frac{\log U}{\log n}\big)$ messages of $O(\log U + \log n)$ bits. By integrating it into the OHSDP framework, we have:

- Without attackers, the error remains $O\big(\frac{U}{\varepsilon} \log \frac{1}{\beta}\big)$;
- With one corrupted user, the error becomes $O\big(\frac{U \log^2 n}{\varepsilon} \log \frac{n}{\beta}\big)$;
- The expected number of messages is $O(\log U \log \log n)$, each of $O(\log U + \log n)$ bits.

The second protocol, GKMPS [30], also achieves central-DP error $O\big(\frac{U}{\varepsilon} \log \frac{1}{\beta}\big)$, with each user sending $1 + O\big(\frac{U \log^2 U \log(U/\delta)}{\varepsilon n}\big)$ messages of $O(\log U)$ bits. By integrating it into OHSDP, we have:

- The error remains $O\big(\frac{U}{\varepsilon} \log \frac{1}{\beta}\big)$ without attackers;
- With one corrupted user, the error becomes $O\big(\frac{U \log^2 n}{\varepsilon} \log \frac{n}{\beta}\big)$;
- The expected number of messages is $O\big(\log n + \frac{U \log^2 U \log(U/\delta)}{\varepsilon \log(1/\delta)}\big)$, each of $O(\log U + \log n)$ bits.

Frequency estimation. For the frequency estimation (a.k.a. histogram) problem, the SOTA solution LWY [47] sends $O(1)$ messages of $O(\log U)$ bits and achieves an ℓ_∞-error of $O\big(\frac{1}{\varepsilon}\sqrt{\log \frac{U}{\beta} \log \frac{1}{\delta}}\big)$. By integrating it into the OHSDP framework, we have:

- Without attackers, the error remains $O\big(\frac{1}{\varepsilon}\sqrt{\log \frac{U}{\beta} \log \frac{1}{\delta}}\big)$;
- With one corrupted user, the error becomes $O\big(\frac{\log^2 n}{\varepsilon}\sqrt{\log \frac{U}{\beta} \log \frac{1}{\delta}}\big)$;
- The expected number of messages is $O(\log n)$, each of $O(\log U + \log n)$ bits.

Range counting. Range counting can be reduced to frequency estimation by building a binary tree over the domain $[0, U]$, where each node represents a range and each level forms a frequency estimation problem. The SOTA shuffle-DP protocol LYD+ [48] achieves an ℓ_∞-error of

$O\big(\frac{\log^{1.5} U}{\varepsilon}\sqrt{\log\frac{U}{\beta}\log\frac{1}{\delta}}\big)$, with $O(\log U)$ messages per user, each of $O(\log U)$ bits. By integrating it into the OHSDP framework, we have:

- Without attackers, the error remains $O\big(\frac{\log^{1.5} U}{\varepsilon}\sqrt{\log\frac{U}{\beta}\log\frac{1}{\delta}}\big)$;
- With one corrupted user, it becomes $O\big(\frac{\log^2 n\cdot\log^{1.5} U}{\varepsilon}\sqrt{\log\frac{U}{\beta}\log\frac{1}{\delta}}\big)$;
- The expected number of messages is $O(\log U\log n)$, each of $O(\log U+\log n)$ bits.

Furthermore, k-selection queries (which return the k-th largest item in the dataset) can be answered via range counting [22]. Our framework supports these queries under poisoning attacks with $\tilde{O}(1)$ rank error, but only of theoretical interest due to its large polylogarithmic terms.

OLAP queries. OLAP queries typically involve five core operations: join, select, project, aggregate, and group-by. Among these, join and projection (distinct count) are not union-preserving and are therefore outside the scope of our framework. Queries that combine selection with sum or count aggregation can be reduced to standard sum or count queries, which our framework directly supports. Group-by queries can be interpreted as answering multiple queries, which can be handled using DP basic composition theory (see Lemma 2.2).

6 Experiments

We conduct experiments on three union-preserving queries: bit counting (Q_{count}), summation (Q_{sum}), and frequency estimation (Q_{hist}), comparing our optimized hierarchical shuffle-DP protocols with the SOTA methods on both synthetic and real-world datasets. Specifically, for Q_{count}, we compare against CSUZZ [15], GKMPS [30], and BBGN [3][6] with our defense framework built on top of GKMPS and BBGN, referred to as Ours+GKMPS and Ours+BBGN. For Q_{sum}, we compare against GKMPS and BBGN with Ours+BBGN. For Q_{hist}, we compare against LWY [47] and CZ [16] with Ours+LWY.

6.1 Setup

Datasets. We incorporate a total of seven datasets in our experiments. The real-world datasets include two salary datasets from Kaggle, namely San Francisco Salary (SF-Sal) [38], and Brazil Salary (BR-Sal) [39]. The remaining two are the Adult dataset (Adult) [42] from the U.S. Census, and the AOL dataset (AOL) [50] containing real-world web search accesses. The synthetic datasets are generated using Uniform distribution (Unif), Zipfian distribution[7](Zipf) with $a = 1.5$, and Gaussian distribution[8](Gauss) with $\mu = \sigma = U/5$.

- Q_{count}: we use there real-world datasets: Adult with $n = 2^{15}$, SF-Sal with $n = 2^{17}$, and BR-Sal with $n = 2^{20}$. We count the number of females in the Adult dataset, and the number of individuals earning above the average salary in the salary datasets. Synthetic datasets are generated from three distributions with $n = 2^{24}$.
- Q_{sum}: we use the same there real-world datasets. On Adult, we sum the ages of individuals, with age values capped at 130. On the salary datasets, we compute the total salary, with salary values capped at $U = 2.5\times 10^5$. Synthetic datasets are generated from three distributions with $n = 2^{24}$ and $U = 10^5$.

[6]We use the IKOS-based solution only in our experiments.

[7]We sample n values from the Zipfian distribution (probability mass function $f(x)\propto x^{-a}$) and take modulo U.

[8]We sample n values from the Gaussian distribution (probability mass function $f(x)\propto\exp(-\frac{(x-\mu)^2}{2\sigma^2})$) and round to integers from 0 to U.

- Q_{hist}: three real-world datasets are used: AOL for website visit frequencies, SF-Sal and BR-Sal for salary frequencies. Synthetic datasets are generated from three distributions, with both real and synthetic datasets set to $n = U = 2^{17}$.

Poisoning attacks. For all queries, we randomly select k users from the user population as corrupted users. By default, we set $k = 1$. For Q_{count}, each corrupted user sends n messages with value 1 to the shufflers. For Q_{sum}, each corrupted user sends n messages with value U to the shufflers. For Q_{hist}, each corrupted user sends n messages with value 1 for every bin in the range $[0, U]$.

Experimental environment and parameters. All experiments are conducted on a single Linux server with an Intel Xeon CPU @ 2.50GHz and 178 GB memory, where we simulate the outputs of all users (with fresh randomness) and aggregate their results, following the convention of previous distributed DP settings [17, 34, 35, 43, 45, 46, 48, 54, 57, 60]. This single-node simulation exactly reproduces the algorithmic process of a real distributed setting in terms of both accuracy and communication, as both depend solely on the generated messages rather than the physical distribution of computation. Moreover, as achieving high utility requires a large n, implementing true n-node distributed environment would be prohibitively expensive. We set the privacy budgets as $\varepsilon = 1$, $\delta = n^{-2}$, and confidence level $\beta = 0.1$ by default. While we set $\varepsilon = 4$ for Q_{hist}. We choose the optimal λ for each setting in our protocols. We use ℓ_1-error for Q_{count} and Q_{sum}, and ℓ_∞-error for Q_{hist}. Each experiment is repeated 100 times, and we report the average after discarding the top 10% and bottom 10% of the results.

6.2 Experimental Results

Utility and communication. We evaluate the performances, including the relative error[9], the average number of messages per user, and the number of bits per message, under with and without attack settings to the three queries on both synthetic and real-world datasets, shown in Table 3 and Table 4, respectively.

The results show a clear superiority of our defense frameworks in terms of utility under poisoning attacks (see Relative Error(w/ atk) column). For Q_{count} and Q_{sum}, all SOTA methods fail to return a reasonable result, i.e., the relative error is larger than 100%. It shows the damage of one corrupted user conducting poisoning attacks. But luckily, our frameworks successfully detect and mitigate the attacks, recovering results with relative errors well below 1%, thereby preserving high utility. The only exception is CZ for Q_{hist}, which is the only related protocol considering poisoning attacks. However, it sends more than 16KB[10] per message, making it only practical for small domain settings. When there is no attack, our frameworks consistently achieve only 2× larger error than the base protocols. Comparing the error with and without attack for our protocols, the error with attack is no more than 225× larger than that without attack, which is roughly a $(\log n)^2\times$ increase. The gap is much smaller (only 10×) for Q_{hist} because the users in the drop out group split their contribution to different counters and reduce the error. The distribution of dataset does not affect the error without attack. But when there is an attacker, our defense frameworks will drop out the entire group of size λ containing corrupted user, and this leads to different errors in different distributions.

In terms of communication cost, our defense frameworks achieve approximately $(\log n)\times$ larger number of messages than the base protocols BBGN and LWY, which send $O(1)$ messages. Since GKMPS has a poor performance when n is not larger than U too much, the communication costs are very large for Ours+GKMPS in Q_{count} and GKMPS in Q_{sum}, which matches our analysis in

[9]For Q_{hist}, we define the relative error with respect to the data size n, i.e., ℓ_∞ error divided by n.
[10]We do not include the extra cost of blind signature [11]. The 16KB message cannot defend against poisoning attacks.

Query	Protocol	Unif			Zipf			Gauss			#Bits
		Relative Error (%)		#Msgs	Relative Error (%)		#Msgs	Relative Error (%)		#Msgs	
		(w/o atk)	(w/atk)		(w/o atk)	(w/atk)		(w/o atk)	(w/atk)		
Q_{count}	CSUZZ	3.80×10^{-5}	200.05	1	3.01×10^{-5}	154.66	1	3.00×10^{-4}	1495.45	1	1
	GKMPS	1.12×10^{-5}	200.05	0.50	1.09×10^{-5}	154.67	0.65	8.80×10^{-5}	1495.45	0.07	2
	BBGN	1.04×10^{-5}	200.05	6	6.91×10^{-6}	154.67	6	6.69×10^{-5}	1495.45	6	28
	Ours+GKMPS	2.82×10^{-5}	3.25×10^{-3}	2704.97	2.27×10^{-5}	2.88×10^{-3}	2707.23	2.17×10^{-4}	1.53×10^{-2}	2697.92	18
	Ours+BBGN	1.95×10^{-5}	3.38×10^{-3}	140	1.41×10^{-5}	3.03×10^{-3}	140	1.28×10^{-4}	1.13×10^{-2}	140	45
Q_{sum}	GKMPS	1.49×10^{-5}	199.98	2583.63	2.31×10^{-3}	2.84×10^{4}	2587.28	2.97×10^{-5}	461.51	2586.98	15
	BBGN	9.70×10^{-6}	199.98	7	1.56×10^{-3}	2.84×10^{4}	7	2.08×10^{-5}	461.51	7	44
	Ours+BBGN	2.26×10^{-5}	3.16×10^{-3}	159	2.89×10^{-3}	0.23	159	4.00×10^{-5}	4.25×10^{-3}	159	61
Q_{hist}	CZ	5.84×10^{-2}	5.75×10^{-2}	3	0.12	0.11	3	5.50×10^{-2}	5.83×10^{-2}	3	2^{17^*}
	LWY	2.53×10^{-2}	100.05	66.88	2.58×10^{-2}	100.23	66.88	2.59×10^{-2}	99.87	66.88	54
	Ours+LWY	5.02×10^{-2}	0.57	514.83	5.03×10^{-2}	0.64	514.83	5.02×10^{-2}	0.58	514.83	69

Table 3. Comparison of protocols on synthetic datasets.

		Relative Error (%)		#Msgs	#Bits	Relative Error (%)		#Msgs	#Bits	Relative Error (%)		#Msgs	#Bits
		(w/o atk)	(w/ atk)			(w/o atk)	(w/ atk)			(w/o atk)	(w/ atk)		
Q_{count}	Dataset	Adult				SF_Sal				BR_Sal			
	CSUZZ	2.31×10^{-2}	302.55	1	1	3.71×10^{-3}	206.99	1	1	7.70×10^{-4}	279.83	1	1
	GKMPS	1.13×10^{-2}	302.29	0.74	2	1.38×10^{-3}	206.96	0.60	2	2.96×10^{-4}	279.82	0.36	2
	BBGN	7.61×10^{-3}	302.34	9	19	1.07×10^{-3}	206.96	8	21	2.05×10^{-4}	291.99	7	24
	Ours+GKMPS	2.61×10^{-2}	0.84	1638.72	10	4.09×10^{-3}	0.11	1051.23	11	7.21×10^{-4}	0.04	1679.21	14
	Ours+BBGN	1.36×10^{-2}	0.55	109	29	1.91×10^{-3}	0.15	146	34	4.91×10^{-4}	0.03	147	39
Q_{sum}	Dataset	Adult				SF_Sal				BR_Sal			
	GKMPS	1.27×10^{-2}	336.93	3909.41	9	3.36×10^{-3}	352.21	11367.11	11	8.22×10^{-3}	7369.82	6053.09	13
	BBGN	9.36×10^{-3}	336.94	9	26	2.08×10^{-3}	352.21	9	39	6.55×10^{-3}	7369.82	8	42
	Ours+BBGN	1.80×10^{-2}	0.51	115	36	4.48×10^{-3}	0.13	125	50	1.32×10^{-2}	0.41	101	53
Q_{hist}	Dataset	AOL				SF_Sal				BR_Sal			
	CZ	7.12×10^{-2}	7.10×10^{-2}	3	2^{17^*}	5.81×10^{-2}	5.90×10^{-2}	3	2^{17^*}	5.85×10^{-2}	5.80×10^{-2}	3	2^{17^*}
	LWY	2.57×10^{-2}	100.03	66.88	54	2.57×10^{-2}	100.02	66.88	54	2.57×10^{-2}	100.00	66.88	54
	Ours+LWY	5.09×10^{-2}	0.58	514.83	69	5.08×10^{-2}	0.57	514.83	69	4.95×10^{-2}	0.58	514.83	69

Table 4. Comparison of protocols on real-world datasets.

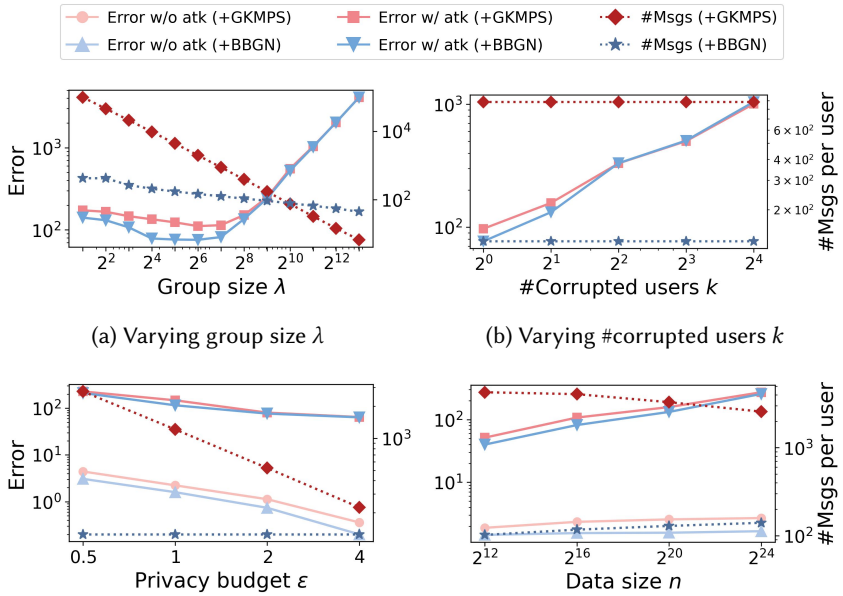
Table 1. Our framework sends $O(\log n)$ more bits in each message. The distribution of dataset does not affect the performance.

We further evaluate the performance of our defense frameworks Ours+GKMPS and Ours+BBGN for the three queries varying parameters, including the group size λ, the number of corrupted users k, the privacy budget ε, and the data size n. The results of Q_{count} are shown in Fig. 3, and the results of Q_{sum} and Q_{hist} are presented in Appendix F.[11]

Group size λ. Fig. 3a shows the error with attack and messages per user. We do not include the error without attack because it uses separate privacy budgets and keeps the same. The results match our expectations: The error with attack first decreases and then increases with λ growth. This is because the error consists of two parts: the accumulated error from the $O(\log n/\lambda)$ counters and the ignored group of size λ. The first part decreases with λ growth because the number of counters decreases and the privacy budgets per counter increase. The second part increases with λ growth. The communication cost decreases for both protocols as λ increases. However, Ours+GKMPS drops faster, since its cost is proportional to $1/\lambda$, whereas Ours+BBGN scales with $1/\log \lambda$.

The results demonstrate that an optimal λ will help to improve both the utility and efficiency, i.e., the lowest point of the line of error. And further increasing the λ leads to a trade-off between utility and efficiency.

[11] https://drive.google.com/drive/folders/1oqBEyg0Uac1V6PMUYDEGV01GdlwRbSUv?usp=drive_link



(a) Varying group size λ

(b) Varying #corrupted users k

(c) Varying privacy budget ε

(d) Varying data size n

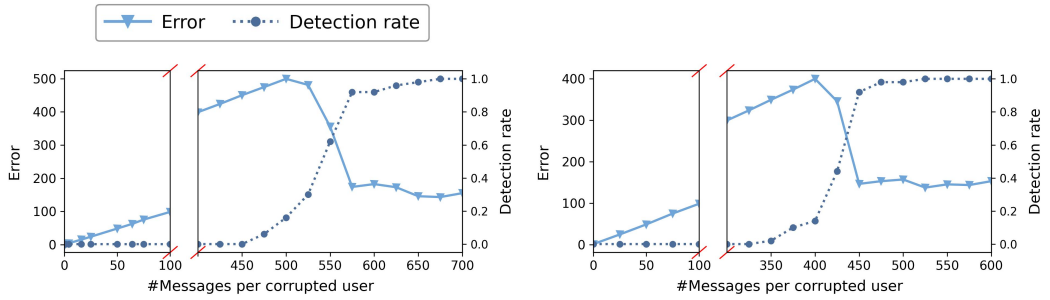
Fig. 3. Comparison of Ours+GKMPS and Ours+BBGN for Q_{count} on uniform distribution varying different parameters.

Number of corrupted users k. Fig. 3b shows that the error with attack inevitably grows linearly with the number of corrupted users k. However, the number of messages per user remains constant. This observation aligns with our theoretical analysis of multiple corrupted users (Theorem 4.4).

privacy budget ε. As shown in Fig. 3c, both protocols have lower error, whether with or without attack, when ε grows. The communication cost of Ours+GKMPS also decreases, but Ours+BBGN keeps the same. Because the latter one has a fixed number of messages, whatever ε is.

Data size n. The results are in Fig. 3d. Both protocols have larger errors with n growth, as it is proportional to polylogarithmic of n. The communication cost changes differently and shows an interesting phenomenon of different protocols: Ours+BBGN increases as larger n leads to more levels in the structure, while the number of messages in each level is stable. Its communication cost is dominated by the number of levels. Ours+GKMPS chooses a larger λ for larger n, which reduces the number of messages in the bottom level, which dominates the total number of messages.

Effect of the number of malicious messages sent by corrupted users. To empirically validate our theoretical analysis that the worst-case error occurs when corrupted users evade detection, we conduct an experiment by varying the number of malicious messages each corrupted user sends. The experiment is performed on the bit counting query Q_{count} over Unif dataset, using Ours+GKMPS and Ours+BBGN as representative protocols. We fix the total number of users to $n = 2^{20}$ and set the number of corrupted users to $k = 1$. The corrupted user sends a fixed number of messages per level, and we vary this number across experiments. The bottom-group size is set to $\lambda = 256$, the fixed parameters are $\varepsilon = 1, \delta = 1/n^2, \beta = 0.1$, the corresponding bottom-level error thresholds are $\theta^{(0)} = 288.65$ for Ours+BBGN and $\theta^{(0)} = 412.14$ for Ours+GKMPS. For each configuration, we run



(a) Ours + GKMPS (b) Ours + BBGN

Fig. 4. Comparison of Ours+GKMPS and Ours+BBGN for Q_{count} on uniform distribution varying #messages the corrupted user send.

50 independent trials and report (i) the absolute error and (ii) the detection rate (i.e., the fraction of runs in which the corrupted user is successfully detected). As shown in Fig. 4, the detection rate remains nearly zero when the number of malicious messages is small, indicating that the corrupted user successfully evades detection. In this regime, the overall error grows rapidly, approximately linearly, with the number of malicious messages and reaches its maximum near the detection boundary (around 400 messages for Ours+BBGN and 500 messages for Ours+GKMPS), which is roughly $(\lambda + \theta_0) - \lambda/2$ as our theoretical expectation. Once the number of malicious messages exceeds the detection threshold, the corrupted user is effectively identified and filtered out, leading to a sharp decrease in the overall error.

7 Conclusion

In this paper, we studied the poisoning attacks under the shuffle-DP model, including breaking the privacy and destroying the utility. We have proposed a general defense framework for all union-preserving queries that can convert any shuffle-DP protocol to a version defensible against such attacks with a limited communication cost, and have demonstrated its versatility through a number of common query workloads. There are some interesting open questions for future research: (i) Can our defense framework support more general queries beyond the union-preserving class, such as join and projection queries? (ii) The communication overhead is still large compared to its base protocol. [48] reduces the communication costs of hierarchical structure queries. Can we adopt the same idea to improve our communication efficiency?

Acknowledgments

This work has been supported by the NTU–NAP Startup Grant (024584-00001), the Singapore Ministry of Education Tier 1 Grant (RG19/25), the Ant Group Research Intern Program, NSFC (62441238), the National Research Foundation, Singapore, and the Cyber Security Agency of Singapore under the National Cybersecurity R&D Programme and the CyberSG R&D Programme Office (Award CRPO-GC3-NTU-001). Any opinions, findings, conclusions, or recommendations expressed in these materials are those of the author(s) and do not reflect the views of the National Research Foundation, Singapore, the Cyber Security Agency of Singapore, or the CyberSG R&D Programme Office. We also thank the anonymous reviewers for valuable suggestions on improving the paper.

References

[1] Victor Balcer, Albert Cheu, Matthew Joseph, and Jieming Mao. 2021. Connecting Robust Shuffle Privacy and Pan-Privacy. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. 2384–2403.

[2] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. 2019. The Privacy Blanket of the Shuffle Model. In *CRYPTO*.

[3] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. 2020. Private Summation in the Multi-Message Shuffle Model. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 657–676.

[4] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. 2006. Can machine learning be secure?. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*. 16–25.

[5] Amos Beimel, Kobbi Nissim, and Eran Omri. 2008. Distributed private data analysis: Simultaneously solving how and what. In *Advances in Cryptology–CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28*. Springer, 451–468.

[6] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic Encryption and Multiparty Computation. In *Advances in Cryptology – EUROCRYPT 2011*. 169–188.

[7] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*. 1467–1474.

[8] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. 2017. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th symposium on operating systems principles*. 441–459.

[9] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. Data Poisoning Attacks to Local Differential Privacy Protocols. In *30th USENIX Security Symposium (USENIX Security 21)*. 947–964.

[10] TH Hubert Chan, Elaine Shi, and Dawn Song. 2012. Optimal lower bound for differentially private multi-party aggregation. In *European Symposium on Algorithms*. Springer, 277–288.

[11] David Chaum. 1983. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto 82*. Springer, 199–203.

[12] David L. Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 24 (1981), 84–90.

[13] Lijie Chen, Badih Ghazi, Ravi Kumar, and Pasin Manurangsi. 2021. On Distributed Differential Privacy and Counting Distinct Elements. In *ITCS*. 56:1–56:18.

[14] Albert Cheu, Adam Smith, and Jonathan Ullman. 2021. Manipulation Attacks in Local Differential Privacy. In *2021 IEEE Symposium on Security and Privacy (SP)*. 883–900. doi:10.1109/SP40001.2021.00001

[15] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. 2019. Distributed differential privacy via shuffling. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*. Springer, 375–403.

[16] Albert Cheu and Maxim Zhilyaev. 2022. Differentially private histograms in the shuffle model from fake users. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 440–457.

[17] Graham Cormode, Tejas Kulkarni, and Divesh Srivastava. 2018. Marginal release under local differential privacy. In *Proceedings of the 2018 International Conference on Management of Data*. 131–146.

[18] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology – CRYPTO 2012*. 643–662.

[19] G. Danezis, R. Dingledine, and N. Mathewson. 2003. Mixminion: design of a type III anonymous remailer protocol. In *Symposium on Security and Privacy*. 2–15.

[20] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium (USENIX Security 04)*.

[21] Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, and Ashwin Machanavajjhala. 2022. R2T: Instance-optimal Truncation for Differentially Private Query Evaluation with Foreign Keys. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 759–772. doi:10.1145/3514221.3517844

[22] Wei Dong, Qiyao Luo, and Ke Yi. 2023. Continual observation under user-level differential privacy. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2190–2207.

[23] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*. Springer, 265–284.

[24] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and trends® in theoretical computer science* 9, 3–4 (2014), 211–407.

[25] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. 2019. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the Thirtieth*

Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, 2468–2479.
[26] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. 2020. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *29th USENIX Security Symposium (USENIX Security 20)*. 1605–1622.
[27] Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. 2021. On the Power of Multiple Anonymous Messages: Frequency Estimation and Selection in the Shuffle Model of Differential Privacy. In *Advances in Cryptology – EUROCRYPT 2021*. 463–488.
[28] Badih Ghazi, Ravi Kumar, and Pasin Manurangsi. 2024. Pure-DP Aggregation in the Shuffle Model: Error-Optimal and Communication-Efficient. In *5th Conference on Information-Theoretic Cryptography, ITC 2024*, Vol. 304. 4:1–4:13.
[29] Badih Ghazi, Ravi Kumar, Pasin Manurangsi, and Rasmus Pagh. 2020. Private counting from anonymous messages: Near-optimal accuracy with vanishing communication overhead. In *International Conference on Machine Learning*. PMLR, 3505–3514.
[30] Badih Ghazi, Ravi Kumar, Pasin Manurangsi, Rasmus Pagh, and Amer Sinha. 2021. Differentially private aggregation in the shuffle model: Almost central accuracy in almost a single message. In *International Conference on Machine Learning*. PMLR, 3692–3701.
[31] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to play ANY mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. 218–229.
[32] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, and Dan Zhang. 2016. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*. 139–154.
[33] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.* 3, 1–2 (sep 2010), 1021–1032. doi:10.14778/1920841.1920970
[34] Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2024. Common Neighborhood Estimation over Bipartite Graphs under Local Differential Privacy. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–26.
[35] Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Wei Ni. 2025. Robust Privacy-Preserving Triangle Counting under Edge Local Differential Privacy. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–26.
[36] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2022. Differentially private triangle and 4-cycle counting in the shuffle model. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1505–1519.
[37] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. 2006. Cryptography from Anonymity. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 239–248. doi:10.1109/FOCS.2006.25
[38] Kaggle. 2014. San Francisco City Employee Salary Data. https://www.kaggle.com/datasets/kaggle/sf-salaries. Accessed: 2025-05-18.
[39] Kaggle. 2020. Monthly Salary of Public Worker in Brazil. https://www.kaggle.com/datasets/gustavomodelli/monthly-salary-of-public-worker-in-brazil. Accessed: 2025-05-18.
[40] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? *SIAM J. Comput.* 40, 3 (2011), 793–826.
[41] Daniel Kifer and Ashwin Machanavajjhala. 2011. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 193–204.
[42] Ron Kohavi et al. 1996. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid.. In *Kdd*, Vol. 96. 202–207.
[43] Tejas Kulkarni. 2019. Answering range queries under local differential privacy. In *Proceedings of the 2019 International Conference on Management of Data*. 1832–1834.
[44] Xiaoguang Li, Ninghui Li, Wenhai Sun, Neil Zhenqiang Gong, and Hui Li. 2023. Fine-grained Poisoning Attack to Local Differential Privacy Protocols for Mean and Variance Estimation. In *32nd USENIX Security Symposium (USENIX Security 23)*. 1739–1756.
[45] Xiaochen Li, Weiran Liu, Jian Lou, Yuan Hong, Lei Zhang, Zhan Qin, and Kui Ren. 2024. Local differentially private heavy hitter detection in data streams with bounded memory. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–27.
[46] Zitao Li, Tianhao Wang, Milan Lopuhaä-Zwakenberg, Ninghui Li, and Boris Škoric. 2020. Estimating numerical distributions under local differential privacy. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 621–635.
[47] Qiyao Luo, Yilei Wang, and Ke Yi. 2022. Frequency Estimation in the Shuffle Model with Almost a Single Message. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2219–2232.
[48] Qiyao Luo, Jianzhe Yu, Wei Dong, Quanqing Xu, Chuanhui Yang, and Ke Yi. 2025. RM2: Answer Counting Queries Efficiently under Shuffle Differential Privacy. *Proc. ACM Manag. Data* 3, 3, Article 210 (2025), 24 pages.
[49] Frank D McSherry. 2009. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 19–30.

[50] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A picture of search. In *Proceedings of the 1st international conference on Scalable information systems*. 1–es.
[51] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2013. Understanding hierarchical methods for differentially private histograms. *Proc. VLDB Endow.* 6, 14 (sep 2013), 1954–1965. doi:10.14778/2556549.2556576
[52] M.G. Reed, P.F. Syverson, and D.M. Goldschlag. 1998. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications* (1998), 482–494.
[53] Michael K. Reiter and Aviel D. Rubin. 1998. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security* (1998), 66–92.
[54] Xuebin Ren, Liang Shi, Weiren Yu, Shusen Yang, Cong Zhao, and Zongben Xu. 2022. LDP-IDS: Local differential privacy for infinite data streams. In *Proceedings of the 2022 international conference on management of data*. 1064–1077.
[55] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. 2020. Data poisoning attacks against federated learning systems. In *Computer security–ESORICs 2020: 25th European symposium on research in computer security, ESORICs 2020, guildford, UK, September 14–18, 2020, proceedings, part i 25*. Springer, 480–501.
[56] Wei Tong, Haoyu Chen, Jiacheng Niu, and Sheng Zhong. 2024. Data Poisoning Attacks to Locally Differentially Private Frequent Itemset Mining Protocols. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 3555–3569.
[57] Tianhao Wang, Bolin Ding, Min Xu, Zhicong Huang, Cheng Hong, Jingren Zhou, Ninghui Li, and Somesh Jha. 2020. Improving utility and security of the shuffler-based differential privacy. *Proc. VLDB Endow.* 13, 13 (Sept. 2020), 3545–3558. doi:10.14778/3424573.3424576
[58] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 21–37.
[59] Yongji Wu, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2022. Poisoning Attacks to Local Differential Privacy Protocols for Key-Value Data. In *31st USENIX Security Symposium (USENIX Security 22)*. 519–536.
[60] Liantong Yu, Qingqing Ye, and Rong Du. 2025. PrivRM: A Framework for Range Mean Estimation under Local Differential Privacy. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–26.
[61] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. 2017. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)* 42, 4 (2017), 1–41.
[62] Yuemin Zhang, Qingqing Ye, and Haibo Hu. 2025. Federated Heavy Hitter Analytics with Local Differential Privacy. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–27.

Received July 2025; revised October 2025; accepted November 2025