Precio: Private Aggregate Measurement via Oblivious Shuffling

F. Betül Durak[1], Chenkai Weng[2], Erik Anderson[1], Kim Laine[1], and Melissa Chase[1]

[1]Microsoft
[2]Northwestern University

August 15, 2023

Abstract

We introduce Precio, a new secure aggregation method for computing layered histograms and sums over secret shared data in a client-server setting. Precio is motivated by ad conversion measurement scenarios, where online advertisers and ad networks want to measure the performance of ad campaigns without requiring privacy-invasive techniques, such as third-party cookies.

Precio has linear (communication) complexity in the number of data points and guarantees differentially private outputs. We formally analyze its security and privacy and present a thorough performance evaluation. The protocol supports much larger domains than Prio. It supports much more flexible aggregates than the DPF-based solution and in some settings has up to four orders of magnitude better performance.

1 Introduction

Privacy-Preserving Aggregation. Numerous applications and services collect statistics about their use, raising privacy and regulatory compliance concerns. The privacy-utility trade-off in collecting less or more information is an active area of research, including for smart meters [30, 15], private networks [29, 10, 19], and other measurements [22].

We focus on privacy-preserving aggregation, where a large number of *clients* each submit a data *report*. The goal is to compute aggregate statistics over the reports and deliver the results to a *Reporting Origin*, without compromising the clients' privacy. We follow the model in Prio [14], where clients secret share their reports and distribute the shares to a small number of servers. The servers compute aggregates over the reports in a way that guarantees differential privacy for each client report. Such systems have in the past been used for telemetry [20] and contact tracing [2]. We focus on the use-case of private ad conversion measurement, presenting a solution which is both more efficient and more flexible than prior solutions.

Web Privacy and Ad Conversion Measurement. People's activities across websites are constantly being tracked to derive insights about online marketing campaigns. This cross-site tracking is typically done via third-party cookies, where website A includes content getting and

setting a cookie tied to website F. When F is also loaded across other websites, it is able to track people's activities across these sites [5].

To mitigate privacy concerns, browser vendors have started to restrict when third-party cookies can be set. For example, Apple Safari blocks all third-party cookies by default and Google Chrome's Privacy Sandbox effort proposes to entirely phase out third-party cookies. Analytics relying on third-party cookies, however, are a vital part of the current ad-funded web that provides invaluable services free of charge to everyone.

Many measurement scenarios require reporting simple aggregates, *e.g.*, how often an ad placement on a publisher's site results in the visitor completing a purchase. Today, these aggregates are computed by first collecting information from individual web browser client, with no guarantee that the data is not used for other, potentially more privacy-invasive, purposes. If there was a way to ensure only aggregate results get revealed, most measurement scenarios could still be enabled, while the privacy risks and concerns would be minimized.

In this work, we propose a solution to this problem in the form of a privacy-preserving aggregate protocol based on secure multi-party computation and differential privacy.

1.1 Our Model

Parties and Trust Assumptions. We consider a system as in Figure 1, which follows along the same basic structure as the Verifiable Distributed Aggregation Functions currently being formalized by the IETF [4]. The system includes the following parties:

- *Clients*, that each submit one data report. As there is no way to limit how client software (*e.g.*, web browsers) behaves, we need security against malicious clients.
- A *Reporting Origin*, that collects reports from the clients, sends them to the helper servers (see below), and chooses which aggregates to compute. This could be an ad network measuring the effectiveness of an ad campaign. A malicious Reporting Origin should not learn anything about individual clients' reports.
- *Helper servers*, that together perform the aggregation.

Our protocol provides privacy against semi-honest helpers. We leave the stronger guarantee of correctness in the presence of malicious helper servers for future work.

Data Reports. Each report collected from a client is a bit string encoding one or more predefined categorical or numerical *attributes*. Each attribute is allocated a certain number of bits from the report, encoding the *attribute value*. Examples of categorical attributes include client information (age, gender, geographic region), device information (user-agent string, OS), or ad category. An example of a numerical attribute would be the dollar amount of ad conversion purchases.

The goal of our protocol is to produce layered histograms on the categorical attributes and, respectively, sums on the numerical attributes.

Repeated Partitioning and Filtering. In a non-private ad conversion measurement system [34], an analyst might start out by partitioning the reports according to an ad campaign identifier. Next, they might partition the reports by geographic region, filtering out regions with no relevant conversion activity, followed by partitioning by age, gender, or other attributes. Repeated partitioning and filtering is essential, as it allows the analyst to explore a sparse space of reports, without having to explore exponentially many combinations of attribute values.

No prior privacy-preserving ad conversion measurement solution allowed such on-the-fly partitioning and filtering. Specifically, prior solutions were based on *Distributed Point Functions* (DPFs) [23, 8, 9], *incremental DPFs* (iDPFs) [7], and Prio [14, 1]. iDPF is very similar to DPF; it adds support for longer reports and differential privacy.

In the (i)DPF approach, attribute values are encoded as bit strings into reports, but histograms can only be computed for prefixes of the entire report. For example, if the report encodes first a geographic location, then an age bracket, and then a device type, one can obtain counts for "east coast", "east coast, 65+", and "east coast, 65+, mobile", but not for "65+, mobile" except by separately querying with each possible location. This becomes infeasible when the number of attributes increases.

In Prio, data is encoded as vectors of values and the only statistics that can be computed are those that can be expressed as sums of these values. For example, to count the number of reports corresponding to a combination of location, age, and device type, the client would have to encode its report into a vector with an entry for every possible (location, age, device type) combination. This becomes infeasible when the number of attributes increases.

1.2 Our Results

Precio. We propose a privacy-preserving aggregation protocol, Precio, based on secure 3-party computation, with linear time and communication complexity in the number of reports. The Reporting Origin learns only differentially private aggregates and no per-client information. The high-level design of Precio is depicted in Figure 1.

Unlike any of the prior proposals, Precio supports privacy-preserving on-the-fly partitioning and filtering of reports. Attributes can be of any length and partitioning can be done on any set of attributes – in any order. Histograms (for categorical attributes) and sums (for numerical attributes) can be computed at any point, enabling remarkable flexibility for a data analyst, without compromising clients' privacy.

Cheap Sums. The Prio protocol [14] can compute sums on private numerical values in the presence of malicious clients. Since the clients' inputs are secret-shared in a group of large order, malicious clients can "poison" the computations with unrealistic inputs. To prevent this, the clients must provide expensive range proofs for their inputs. Prio+ [1] resolves this problem using limited size domains and Boolean secret sharing, but requires a costly conversion protocol to produce arithmetic shares for a sum computation.

In Precio, clients secret share their inputs in a cyclic group of limited (small) size. The shares are then converted to shares in a larger order group for the sum computation. This is done by combining *Quotient Transfer* [31] and *Oblivious Transfer* (OT) (Section 3.3), avoiding the expensive range proofs of Prio and with a cheaper share conversion than the one in Prio+.

Security Guarantees. We provide robustness against malicious clients: a malicious client cannot cause a report to be counted more than once (for categorical attributes) and its impact on the results of sums (for numerical attributes) is limited by the allowed input range.

We provide differential privacy (with a Gaussian mechanism) with parameter ϵ. This achieves (ϵ, δ)-DP (for no layering) with a very small δ for histograms. In other words, the reported histograms reveal only noisy aggregate counts.

Protocol	Time complexity	Server-to-server comm.	Client-to-server comm.
DPF [23, 8, 9]	$(2^\ell C)$ DPF.Eval calls	0	$\lambda\ell$
iDPF [7]	$(\ell C^2/t)$ DPF.Eval calls	$\ell \log_2(p)$	$\lambda\ell$
Prio [14]	$O(BC)$	$O(C \log_2(p))$	$B \log_2(p)$
Bucketization	$O(C + BM)$	$O(\ell(C + BM))$	ℓ

Table 1: Complexity comparisons of private histogram computation. ℓ is the report length; t is the pruning threshold ($t = 1$ for full histogram); $\log_2(p)$ is the size of the finite field; λ is the security parameter; C is the number of client reports; B is the number of buckets ($B = 2^\ell$ for full histogram, but in some other cases $B \ll 2^\ell$); $\bar{M}$ is the noise added on average to each bucket in Bucketization. DPF row does not account the DP protection (our scheme without DP would mean $\bar{M} = 0$). Note that the time complexity of DPF.Eval is exponential in ℓ.

We prove privacy against semi-honest non-colluding helper servers.[1]

Complexity. The time complexity of our histogram and sum protocol is $O(C + BM)$, where C is the number of client reports, B is the number of buckets, and M is the average noise added to each bucket.[2] The communication complexity between servers is $O(\ell(C + BM))$, where ℓ is the report length. The communication required from *each client* is only 2ℓ, with some encryption overhead. The detailed analysis in Appendix B.

When the report size ℓ increases, the number of buckets increases ($B = 2^\ell$ for full histograms). To resolve this, we introduce a layering technique, which we will describe in the next section. This layering technique will also allow us to do more flexible aggregation. In layering, both the domain size of each attribute and the distribution will affect the total complexity. Adding pruning to layering further reduces the complexity. For instance, we can limit the histogram with buckets occurring more than t times with complexity $O\big(\ell C + \frac{C}{t-M} M\big)$.

When we consider the communication complexity of the aggregate system as the total communication required for an end-to-end execution, we must include the communication required from clients to servers as well. In that sense, the linear complexity in the number of clients is inevitable, even in systems like those based on iDPF, where the server-to-server complexity is much lower. In our system both client-to-server and server-to-server communication are linear in the number of clients.

Table 1 compares our time and communication complexity to most relevant prior work.

Experiments. We run experiments to measure the performance of Precio on various sizes of reports for histograms. The results and analysis are shown in Section 5.

We explore five distinct test scenarios: 1) constructing a full histogram (without pruning) on a single attribute up to 22 bits; 2) constructing subset-histograms (with pruning) for different pairs of attribute sizes and data distributions; 3) constructing a subset-histogram (with pruning) for up to 10M reports with a large 32-bit attribute by breaking the attribute into two 16-bit chunks; 4) finding heavy-hitters in a set of Zipf-distributed very large 256-bit attribute values broken into 16 16-bit chunks (compared to the iDPF-based scheme in [7]); 5) a sum protocol on numerical

[1] As usual, semi-honest adversaries execute the protocol honestly but try to learn additional information from the protocol transcript.

[2] $M = O\left(\ln\left(1/\delta\right)\right)$ to achieve $(\Omega(1), \delta)$-DP.

attributes in 10M reports.

In each scenario where a comparison is meaningful, we significantly outperform DPF-based solutions. We omit a direct comparison to Prio, as it cannot perform such large and complex histograms at all. In particular, the experiments demonstrating a histogram on a large 32-bit attribute show the power of our layering technique. This would be prohibitively expensive with any other known approach.

1.3 Our Techniques

Base Protocol. We begin with the proposal of Gordon and Mazloom [33], where a histogram is built over a single categorical attribute. Clients securely send Boolean secret shares of their reports to two helper servers. To achieve differential privacy, for each possible attribute value a number is sampled from a (rounded, truncated, and shifted) Gaussian and as many *dummy reports* with that particular attribute value are added. Each server obtains secret shares of these addition reports. In Gordon and Mazloom's proposal, the sampling and generation of these dummy reports happens within a 2PC protocol.

Next, the servers run another 2PC protocol to shuffle the reports, hiding which ones are original and which are noise. This results in a new set of shares randomly permuted. The servers then reveal these new shares and construct a histogram on the values. For cases with an additional numerical attribute, an extra 2PC can be used to add the numerical values for each category.

Optimizing Building Blocks. The Gordon and Mazloom protocol contains three costly steps: 1) The two servers use costly generic 2PC to jointly generate dummy reports; 2) they shuffle the shares, requiring $O(C \log C)$ OTs, where C is the number of reports and $C\ell \log C$ communication, *i.e.*, $\log C$ times the size of the set of reports; 3) a costly 2PC is used to add numerical values, which cannot be done naively to prevent malicious clients from submitting arbitrarily large inputs.

In our protocol, we optimize each step. The two helpers independently sample noise and generates dummy reports, removing the need for generic 2PC. We introduce an additional helper party and use a 3-party permutation protocol corresponding roughly to two rounds of the protocol of [35]. The result is a protocol entirely based on symmetric operations where two of the three parties each send one message of size $C\ell$ (the size of the initial set of reports) and the third sends nothing. We combine the Quotient Transfer idea from [31] with a three-party OT for secure addition of numerical attributes. This results in a lightweight protocol for building a histogram on categorical attributes and subsequently computing sums of numerical attributes.

Layered queries. The base protocol above allows an analyst to pick on the fly which (categorical) attribute to partition and filter by. Unfortunately, re-running the protocol for subcategories does not work, as dummy reports at each iteration would accumulate and degrade both accuracy and efficiency.

For example, consider the second experiment in Section 5.3, in which we have 10 000 000 reports, each 256 bits. We want to find heavy hitter: bit strings that occur most frequently. While the domain is enormous, if we partition repeatedly for each 16-bit chunk, pruning partitions with very few reports, we can hope to efficiently build a histogram for all elements that occur sufficiently often.

Doing this naively with the base protocol would result in dummy reports being added for each of the 2^{16} values of an attribute at each of the 16 layers. These dummies means a factor of 16 increase in the noise of the final results. Moreover, they will significantly increase the cost of running the

protocol since we will have many more records to work with. As we will see later, for practical ϵ and dataset sizes, the dummies in a single level already form a non-trivial fraction of the total records (sometimes a vast majority). Finally, the additional noise will have a significant effect on the efficacy of pruning: either we will end up pruning more of the results that we'd like to keep, or we will prune many fewer buckets and end up with a significant loss in efficiency.

To solve this, dummy reports are added in a specific manner so they are automatically cleaned up in the next partitioning. This creates a layered histogram protocol where DP noise in intermediate layers only affects communication cost, while DP noise in the last layer impacts accuracy.

One more issue is that, since we are essentially computing 16 histograms (one for each level), we have to be very careful to optimize our use of the DP budget, to make sure that we can get both good utility and achieve DP with a reasonable total ϵ. This involves (1) careful DP budget management, so that we use less budget on parts of the protocol that don't directly affect the accuracy of the result (2) use of Gaussian noise, which is more complex to analyze but has better tail bounds and allows for better tradeoffs in our setting, and (3) a tight accounting of the budget being used when we combine the information revealed across all levels - we get this by using the privacy accountant of [27].

Why not Mixnets. An alternative to shuffling with helper parties might be to use mixnets. The client could encrypt the shares in a way that allows both servers to partially decrypt and shuffle them. This would create pairs of ciphertexts (encrypting shares) readable by the two servers, respectively, with neither knowing the full permutation.

While ensuring privacy, this method lacks the essential feature of cheap on-the-fly partitioning and filtering. Clients would have to know how many times each attribute must be shuffled to make sure that when it comes time to partition on that attribute its shares can be decrypted by the helpers, while the remaining attributes cannot.

Alternatively, the client could encrypt the shares with a re-randomizable encryption scheme. The servers could take turns shuffling and re-randomizing the ciphertexts, and then perform threshold decryptions. To provide privacy against corrupt servers, this encryption needs to be re-randomizable and RCCA secure. Such schemes are much more expensive than standard public key encryption [21].

1.4 Related Work

There exists a large volume of prior work in privacy-preserving aggregation, based on general purpose MPC [24, 41], Distributed Point Functions (DPF) [23, 8, 9, 7], or a variety of techniques [14, 1, 42, 43]. The closest in terms of functionality are the DPF-based solutions: regular DPF [23, 8, 9] and incremental DPF (iDPF) [7]. Both iDPF and Prio [14] are currently being proposed for standardization by IETF [40, 22, 38].

The DPF-based solutions introduce a high computational complexity linear in the size of the key space, while avoiding the linear communication complexity between servers. The iDPF-based protocol improves it to complexity quadratic to the number of clients, but only for subset histograms. We describe an existing (i)DPF-based private aggregation proposal in Appendix E and compare (i)DPF with our protocol in Section 5.

Prio provides a complete toolbox for secure and private aggregation. It requires clients creating proofs that their reports are well-formed. For histograms, Prio encodes each attribute in a one-hot vector and generates the histogram from the vectors using generic MPC protocols. However, it provides no (practical) formal differential privacy guarantees and does not support layered histograms. To imitate layered histograms, one would need to encode the report into a massive vector of the

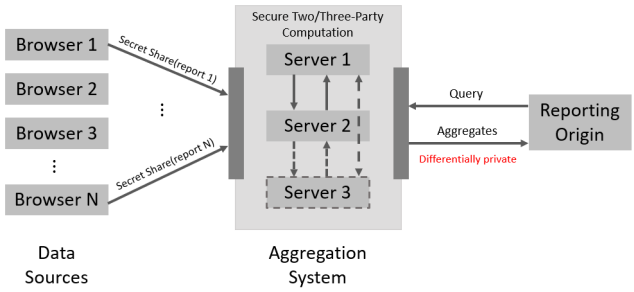

Figure 1: System Architecture.

size of the entire attribute space, which is completely impractical for almost all of our experiments. Due to these significant differences, we do not compare performance directly with Prio.

It is worth noting that Precio differs from the DPF-based protocols and Prio on the trust assumption. Namely, we assume an honest-majority among three servers, while the others require only two non-colluding servers.

Mixnet-based 2-server protocols with verifiable shuffling would incur prohibitive overhead for using public-key operations and zero-knowledge proofs [37, 12]. A private messaging scheme called Vuvuzela [39] employs mixnets and differential privacy, and can potentially be used for simple histogram queries. However, its differential privacy guarantees are undefined for secure aggregation.

Several works [13, 6] propose differential privacy by shuffling. However, they consider a different trust model, where each client changes their input with a small probability. These inputs are shuffled with a permutation that the adversary does not know, which – combined with the noise added by all the clients – provides differential privacy with significantly less noise per client than simple local DP would require. This approach requires many honest clients: if the adversary can omit the noise in most of the inputs, then the shuffling provides no additional privacy. This is partly a problem in our setting as well, since the Reporting Origin will decide which reports to submit for aggregation and the helper servers have no way to verify that the reports came from legitimate clients.

The histogram protocol from [13] requires that each client sends a single bit for each element in the domain. In our application the domain is the set of all ℓ-bit strings, so [13] requires 2^ℓ bits of communication per client. This is clearly infeasible for larger reports. A layering approach may still be possible, but would require significantly smaller bitstrings and more layers. Our layering protocol also allows us to remove dummy reports added in the previous layers each time we do a new bucketization. It is not clear how to correct errors in previous layers when using the approach from [13], so the errors would accumulate.

2 Preliminaries

2.1 Notation

We consider a protocol where multiple clients (*e.g.*, web browsers) each input a single data report. The protocol performs a secure 3-party computation using three helper servers $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, and outputs results to a Reporting Origin $\mathcal{R}$. We use $\mathsf{b} \in \{1, 2, 3\}$ to denote a server index. Unless

$d_1 = 00\ 010\ 25$
$d_2 = 01\ 001\ 32$
$d_3 = 00\ 000\ 19$
$d_4 = 00\ 010\ 64$
$d_5 = 10\ 010\ 53$

Figure 2: A visualization of our small example: the dataset D with $C = 5$ reports with $\mu = 3$ attributes: two categorical (2 and 3 bits) and one numerical (modulo 101).

explicitly stated otherwise, all indices start from 1.

D denotes the dataset of all reports d_i, one from each client; we denote its size by C. Each report consists of μ attributes. The value of the m-th attribute of d_i is $d_i[\mathsf{m}]$, where $\mathsf{m} \in [\mu]$. The attribute value is an element of a group G_m. We assume that G_m includes a "dummy value" $\perp$ and that $d_i[\mathsf{m}] \in G_\mathsf{m} \setminus \{\perp\}$. The $\perp$ value is reserved for internal use in our protocol.

There are several ways to implement the groups G_m, depending on the type of the attribute m. We consider two types of attributes: *categorical* and *numerical*. Our protocol partitions the reports and computes histograms over the categorical attributes, and subsequently computes sums over the numerical attributes. If a desired categorical attribute m requires ℓ_m bits, we set $G_\mathsf{m} = \mathbb{Z}_2^{\ell_\mathsf{m}}$ and $\perp = 1 \ldots 1$. To represent a numerical attribute m, we use $G_\mathsf{m} = \mathbb{Z}_{p_\mathsf{m}}$ for a large enough odd integer p_m which is set to a value larger than twice the maximum numerical attribute size which will be summed. Sums over numerical attributes are computed by first converting the values to a larger field $\mathbb{Z}_{p'_\mathsf{m}}$ (see Section 3.3), where an odd integer $p'_\mathsf{m} \gg p_\mathsf{m}$ is large enough to be able to represent (twice) the sum.

We let L_m be the order of G_m and let $G = G_1 \times G_2 \times \cdots \times G_\mu$. Hence, reports d_i, for $i \in [C]$, are elements of G.

An ℓ-bit report d_i represents μ different attributes, such that $\ell = \sum_{\mathsf{m}=1}^{\mu} \ell_\mathsf{m}$, meaning each attribute m requires ℓ_m bits. A categorical attribute m with ℓ_m bits can hold $L_\mathsf{m} = 2^{\ell_\mathsf{m}}$ values. For a selected categorical attribute m, our protocol partitions the set of reports into *buckets* $\mathcal{B}_j$, for $j \in G_\mathsf{m}$. We denote $\mathcal{B}_\perp$ a bucket reserved for the dummy value, which will be used internally by the protocol. We note that the number of non-empty buckets for a categorical attribute m may be much smaller than L_m, depending on how the values are distributed.

$D_i^{(\mathsf{b})}$ denotes server b's share of the i-th report. We denote server b's shares of the full dataset D by $D^{(\mathsf{b})}$.

Example. We will use a small running example to demonstrate how the protocol works. Suppose we have 5 client reports, where each report consists of two categorical and one numerical attribute: Gender (represented with 2 bits for "he/she/they/$\perp$"), Ads Category (represented with 3 bits), and Dollars Spent (represented within a range of $[0, 100]$). In this example, we take $G_1 = \mathbb{Z}_2^2$ and $G_2 = \mathbb{Z}_2^3$, with group operation bit-wise XOR, $G_3 = \mathbb{Z}_{201}$ with group operation modular addition. Since we have $C = 5$, D consists of 5 reports. Each report will have $\mu = 3$ attributes with 12 bits in total: $d_i[1] = \mathsf{x}_1\mathsf{x}_2$ and $d_i[2] = \mathsf{y}_1\mathsf{y}_2\mathsf{y}_3$ as well as a secret value $d_i[3]$ modulo 201 (8 bits). To build a histogram on attribute "Gender" ($\mathsf{m} = 1$), we will obtain $L_\mathsf{m} = 2^2$ buckets: $\{\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4\}$, where $\mathcal{B}_4$ is reserved for dummy reports with $d_i[1] = 11$. An example of a corresponding dataset D (without considering any secret sharing yet) is given in Figure 2. Notice that there are no reports with $d_i[1] = 11$ or $d_i[2] = 111$, as these buckets are reserved for the dummy reports.

$d_1^{(1)} = 10\ 011\ 78$
$d_2^{(1)} = 00\ 011\ 45$
$d_3^{(1)} = 01\ 111\ 10$
$d_4^{(1)} = 11\ 110\ 52$
$d_5^{(1)} = 01\ 001\ 11$

(a) Shares of $\mathcal{S}_1$: $D^{(1)}$

$d_1^{(2)} = 10\ 001\ 68$
$d_2^{(2)} = 01\ 010\ 88$
$d_3^{(2)} = 01\ 111\ 9$
$d_4^{(2)} = 11\ 100\ 12$
$d_5^{(2)} = 11\ 011\ 42$

(b) Shares of $\mathcal{S}_2$: $D^{(2)}$

Figure 3: Secret shares of D held by $\mathcal{S}_1$ and $\mathcal{S}_2$.

2.2 Secret Sharing

In the GMW protocol [24], a secret value is information-theoretically shared between multiple parties for secure multi-party computation. Let G denote a finite additive group. In the 2-party case, a client can share a secret value $k \in G$ to two servers by first uniformly sampling $r \leftarrow G$, sending r to one server, and $k - r$ to the other server. Neither share alone reveals any information about the value k. Such a scheme works for both Boolean circuits and arithmetic circuits; it requires no communication for the addition of two secret values, or addition or multiplication with public constant values.

Example. Continuing with our example, we secret share the reports in D for $\mathcal{S}_1$ and $\mathcal{S}_2$ as follows: $d_i[j] = d_i^{(1)}[j] \oplus d_i^{(2)}[j]$, for $i \in [5], j \in \{1, 2\}$ and $d_i[3] = d_i^{(1)}[3] + d_i^{(2)}[3] \mod 201$. We depict the shares in Figure 3.

2.3 Differential Privacy

Differential privacy [17, 18] protects the privacy of individual rows in a database, while still allowing meaningful aggregate queries to be made. For each (ordered) database $D \in \chi^C$ with $D(i) = d_i$, we define an (unordered) database $D \in \mathbb{N}^\chi$ by $D(j) = \#\{i : d_i = j\}$.

Let $\mathcal{M}$ be a randomized algorithm with domain $\mathbb{N}^\chi$ and let $D, D' \in \mathbb{N}^\chi$ be two neighboring databases that differ on only one row and have the same cardinality.[3]

We say that a mechanism $\mathcal{M}$ is (ϵ, δ)-differentially private ((ϵ, δ)-DP) for parameters $\epsilon \geq 0$ and $\delta \in [0, 1]$, if for any $S \subseteq \mathsf{Range}(\mathcal{M})$ and any neighboring D and D',

$$\mathsf{Pr}[\mathcal{M}(D) \in S] \leq e^\epsilon \mathsf{Pr}[\mathcal{M}(D') \in S] + \delta\,.$$

The property of differential privacy is maintained through *post-processing*. Informally, this means that once differential privacy is achieved for the output of a particular query, the data curator can make any computations with this output without violating the formal differential privacy guarantees.

To achieve (ϵ, δ)-DP for our protocol, we utilize the *Gaussian mechanism*, where noise is drawn from $\mathcal{N}(0, \sigma^2)$ with $\mathsf{PDF}_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}$ and added to the output of a statistical aggregate. This distribution has zero mean and standard deviation σ. In Section Section 4.4, we explain how we set the distribution parameters by using the techniques to compute privacy random variables [27, 26]. In our protocol, noise is added by both servers independently.

[3]Formally, in histogram D and D' are neighbours iff $\exists j$, $D_j \neq D'_j$ and $\forall i \neq j$, $D_i = D'_i$.

2.4 Oblivious Random Shuffling

We will make use of an oblivious shuffling protocol that runs between three servers. It inputs a dataset, initially secret shared between two servers, and outputs secret shares of a shuffled dataset. Obliviousness means that none of the servers learns the mapped positions before and after the shuffling for any element in the dataset.

Multiple oblivious shuffling protocols have been presented in prior work. Chase *et al.* [11] uses the idea of an oblivious permutation for 2-party oblivious random shuffling. However, for performance reasons, the 2-party approach is insufficient for our protocol. Instead, Mohassel *et al.* [35] proposed an oblivious permutation protocol in the honest majority 3-party setting, with linear computation and communication cost. We will instantiate a modified version of [35] in Section 3.2.

3 Subroutines

3.1 Differential Privacy with Constraints For Histograms

To achieve differential privacy for histograms, we add noise to the counts for each bucket. Ideally, we would like to use the Gaussian mechanism, which samples both positive and negative values for noise from a zero-centered Gaussian distribution. However, in our case positive noise values indicate how many dummy reports to add, but we cannot handle negative noise values as we cannot remove any reports. One solution would be to follow the proposal from Medina *et al.* [36] as a general framework for private optimization with non-negative constraints. Instead, we hope to still align with the standard Gaussian mechanism, as it provides a better balance between privacy and utility and it allows us to leverage the numerical composition in [27].

Rounded Shifted Truncated Gaussian Mechanism. As we stated before, our aim is to use the standard Gaussian distribution in a way that leads us to generate discrete positive or negative noise.

First, we define the truncation of the Gaussian. We sample a noise following $\mathcal{N}(0, \sigma^2)$ and we resample it until its value becomes larger than $-M - 1/2$. We call this noise X and we let n be its rounding to the nearest integer. We obtain the rounded truncated Gaussian noise with $\Pr[n] = \int_{n-1/2}^{n+1/2} \mathsf{PDF}_\sigma(x)/(1-\mathsf{p})$ for $n \geq -M$, and 0 otherwise, where p defined as the resampling probability, which we will define shortly. Finally, we shift the output by adding M to n, which ensures that $n + M$ is always non-negative.

In our application, we use $n + M$ as the number of dummy reports to be added to a particular bucket and later subtract the value M (a public parameter) from each count, leaving a Gaussian-like noise n added to the buckets.

Let X follow the truncated Gaussian distribution of point $-M - 1/2$. Given the PDF of the standard Gaussian, we define the resampling probability p as $\mathsf{p} = \int_{-\infty}^{-M-1/2} \mathsf{PDF}_\sigma(x)$. As an example, if we want $\mathsf{p} \approx 2^{-40}$, M becomes $M = \lceil 7.05\sigma - 0.5 \rceil$.

As described above, the noise added in each bucket consists of $n + M$ dummy reports. Let $\mathsf{PDF}_X(x) = \frac{\mathsf{PDF}_\sigma(x)}{1-\mathsf{p}}$. Since

$$\mathbb{E}[X] = \int_{-M-1/2}^{\infty} x \, \mathsf{PDF}_X(x) \, dx = \sigma^2 \mathsf{PDF}_X(-M - 1/2)$$

the expected value of $X + M$ becomes $\bar{M} = M + \sigma^2 \mathsf{PDF}_X(-M - 1/2)$, which is very close to M [4].

The protocol for noise generation is shown in Figure 4.

Protocol Π_{NoiseGen}

Parameters. An attribute index m. L_{m} buckets. There are four parameters: $\sigma_{\mathsf{buckets}}, M_{\mathsf{buckets}}$ and $\sigma_{\mathsf{flush}}, M_{\mathsf{flush}}$ which describe the noise distributions used for the attribute buckets and the dummy value bucket, respectively. We use discrete truncated Gaussian distribution with parameter σ which is either $\sigma_{\mathsf{buckets}}$ (for attribute buckets) or σ_{flush} (for the dummy value buckets) and M is either M_{buckets} or M_{flush}.

Input. Each server $\mathcal{S}_{\mathsf{b}}$ inputs $(\mathsf{dp}, D^{(\mathsf{b})}, \mathsf{m})$, where $D^{(\mathsf{b})}$ is the full dataset share of the server b and m is the attribute index.

Noise generation. For each $\mathsf{b} \in \{1, 2\}$, for each attribute bucket $j \in G_{\mathsf{m}} \setminus \perp$, $\mathcal{S}_{\mathsf{b}}$ randomly samples noise (until it is larger than $-M_{\mathsf{buckets}} - \frac{1}{2}$ by rejection sampling) from the distribution $\mathcal{N}(0, \sigma^2_{\mathsf{buckets}})$ and rounds it to the nearest integer. We call this rounded noise $n_j^{(\mathsf{b})}$. For the dummy value bucket $j = \perp$, each server does the same using M_{flush} and σ_{flush}. All the noise values are recorded as $\mathcal{N}^{(\mathsf{b})} = (n_j^{(\mathsf{b})} + M_{\mathsf{buckets}})_{j \in G_{\mathsf{m}} \setminus \perp} \,||\, (n_\perp^{(\mathsf{b})} + M_{\mathsf{flush}})$.

Generating dummy reports. For each $\mathsf{b} \in \{1, 2\}$ and each bucket $j \in G_{\mathsf{m}}$, $\mathcal{S}_{\mathsf{b}}$ creates $\mathcal{N}^{(\mathsf{b})}[m]$ dummy reports as follows: for noise index $i \in [n_j^{(\mathsf{b})} + M]$, set $d_{i,j}^{(\mathsf{b})}[\mathsf{m}] = j$ and $d_{i,j}^{(\mathsf{b})}[v] \leftarrow \perp (\in G_v)$ for $v \in [\mu] \setminus \{\mathsf{m}\}$ which form one dummy report $d_{i,j}^{(\mathsf{b})}$ (in the case of a numerical attribute for summing, $d_{i,j}^{(\mathsf{b})}[v] \leftarrow 0$ $(\in G_v)$ for v the attribute to be summed). $\mathcal{S}_{\mathsf{b}}$ forms all of $n^{(\mathsf{b})} = \big(\sum_{j \in G_{\mathsf{m}} \setminus \perp} n_j^{(\mathsf{b})} + M_{\mathsf{buckets}}\big) + (n_\perp^{(\mathsf{b})} + M_{\mathsf{flush}})$ dummy $d_{i,j}^{(\mathsf{b})}$ reports as $D_{\mathsf{dum}}^{(\mathsf{b})}$. Each server b shuffles the $D_{\mathsf{dum}}^{(\mathsf{b})}$.

Appending shares to dummy reports. $\mathcal{S}_1$ and $\mathcal{S}_2$ share the numbers $n^{(1)}$ and $n^{(2)}$ with each other. Set $D_{\mathsf{priv}}^{(1)}$ and $D_{\mathsf{priv}}^{(2)}$ as follows: $(D_{\mathsf{priv}}^{(1)})_i = D_i^{(1)}$ for $i < C$; $(D_{\mathsf{priv}}^{(1)})_i = (D_{\mathsf{dum}}^{(1)})_{i-C}$ for $C < i \le C + n^{(1)}$; and $(D_{\mathsf{priv}}^{(1)})_i = 0$ for $C + n^{(1)} < i \le C + n^{(1)} + n^{(2)}$. $\mathcal{S}_2$ computes similarly except that it puts all 0 reports before the dummy reports of $\mathcal{S}_1$.

Output. Each server $\mathcal{S}_{\mathsf{b}}$ outputs $D_{\mathsf{priv}}^{(\mathsf{b})}$.

Figure 4: The protocol of DP noise generation.

Example. We continue our example from previous Section 2.1. We want to build a histogram on the attribute "Gender" ($\mathsf{m} = 1$) with $L_{\mathsf{m}} = 4$ buckets $\{\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4\}$. In our Π_{NoiseGen} protocol, the *Noise generation* step will sample a number of dummy reports for each of these four buckets, with $\mathcal{B}_4$ consisting entirely of dummy reports.

Suppose the noise vector $\mathcal{N}^{(1)} = (2, 1, 0, 2)$ for $\mathcal{S}_1$.[5] This means that a total of $n^{(1)} = 5$ new

[4] The computation can be found in Section B.1.

[5] Note that sampling such a noise vector is in practice unrealistic. Instead, we would expect to get noise values centered around our chosen shift parameter M. We use this small vector in this example for the sake of simplicity.

$d_1^{(1)} = 10\ 011\ 78$
$d_2^{(1)} = 00\ 011\ 45$
$d_3^{(1)} = 01\ 111\ 10$
$d_4^{(1)} = 11\ 110\ 52$
$d_5^{(1)} = 01\ 001\ 11$
$d_6^{(1)} = 00\ 111\ 0$
$d_7^{(1)} = 00\ 111\ 0$
$d_8^{(1)} = 01\ 111\ 0$
$d_9^{(1)} = 11\ 111\ 0$
$d_{10}^{(1)} = 11\ 111\ 0$
$d_{11}^{(1)} = 00\ 000\ 0$
$d_{12}^{(1)} = 00\ 000\ 0$

(a) $D^{(1)}_{\mathsf{priv}}$

$d_1^{(2)} = 10\ 001\ 68$
$d_2^{(2)} = 01\ 010\ 88$
$d_3^{(2)} = 01\ 111\ 9$
$d_4^{(2)} = 11\ 100\ 12$
$d_5^{(2)} = 11\ 011\ 42$
$d_6^{(2)} = 00\ 000\ 0$
$d_7^{(2)} = 00\ 000\ 0$
$d_8^{(2)} = 00\ 000\ 0$
$d_9^{(2)} = 00\ 000\ 0$
$d_{10}^{(2)} = 00\ 000\ 0$
$d_{11}^{(2)} = 00\ 111\ 0$
$d_{12}^{(2)} = 11\ 111\ 0$

(b) $D^{(2)}_{\mathsf{priv}}$

Figure 5: Output of Π_{NoiseGen} on small example dataset D. Last 7 entries in D_{priv} are dummy reports; 5 of them were are added by $\mathcal{S}_1$ and 2 by $\mathcal{S}_2$.

reports will be added: two reports $d^{(1)}_{1,1}, d^{(1)}_{2,1}$ to $\mathcal{B}_1$, one report $d^{(1)}_{1,2}$ to $\mathcal{B}_2$, and two reports $d^{(1)}_{1,4}, d^{(1)}_{2,4}$ to $\mathcal{B}_4$.

In step *Generating dummy reports*, each dummy report (for $\mathcal{S}_1$) will have the form $d^{(1)}_{i,j} = [j|111|0_{201}]$, where j is represented in binary with 2 bits, the second (categorical) attribute is set to the reserved value 111, and the last 7-bit (numerical) attribute is set to $0 \in \mathbb{Z}_{201}$. The same steps are repeated for $\mathcal{S}_2$, with different total noise vector $\mathcal{N}^{(2)}$ and dummy report count $n^{(2)}$.

Finally, in the step *Appending shares to dummy reports*, $\mathcal{S}_1$ will append the $n^{(1)} = 5$ dummy reports, as well as $n^{(2)}$ fake reports with all bits filled with 0, to its true reports from 5 clients. Thus, the only communication needed between $\mathcal{S}_1$ and $\mathcal{S}_2$ is the exchange of the numbers $n^{(1)}$ and $n^{(2)}$.

Concretely, suppose the noise vector of $\mathcal{S}_2$ be $\mathcal{N}^{(2)} = (1, 0, 0, 1)$. We depict the noise addition following our example in Figure 5.

If the servers now reveal the buckets for the first attribute, they can then organize the reports in buckets accordingly:

$$\mathcal{B}_1 = \{d_1^{(\mathsf{b})}, d_3^{(\mathsf{b})}, d_4^{(\mathsf{b})}, d_6^{(\mathsf{b})}, d_7^{(\mathsf{b})}, d_{11}^{(\mathsf{b})}\}, \quad \mathcal{B}_2 = \{d_2^{(\mathsf{b})}, d_8^{(\mathsf{b})}\},$$

$$\mathcal{B}_3 = \{d_5^{(\mathsf{b})}\}, \quad \mathcal{B}_4 = \{d_9^{(\mathsf{b})}, d_{10}^{(\mathsf{b})}, d_{12}^{(\mathsf{b})}\}.$$

Finally, they reveal the requested histogram as the counts of the buckets $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, ignoring the dummy bucket $\mathcal{B}_4$.

3.2 Oblivious Random Shuffling

In our work, we take the Mohassel *et al.* [35] protocol and modify it into an efficient honest-majority 3-party oblivious random shuffling protocol with linear complexity. The protocol is formally described in Figure 6.

In *Initialize* step, we allow each pair of servers to jointly sample a permutation and a vector of masks. In practice, the permutation and mask can both be sampled from random seeds so the communication cost is trivial. Compared to 2-party protocols, the presence of the third party leads to linear instead of logarithmic computational and communication overhead. During the *Shuffling* phase, each party needs to send only one message, resulting in a constant number of rounds of communication. We do not need commitments or interaction because of the third server. More precisely, when one of the servers is corrupted, the randomness from the (third) honest server is enough to create a uniform distribution for the permutation.

Protocol Π_{RandShuf}

Notation. When the operators $\{+, -\}$ are applied to vectors, they mean element-wise addition and subtraction. Permutations π are on the index set $[C]$ and $\pi(D)$ is defined by $\pi(D)_{\pi(i)} = D_i$

Input. For $\mathsf{b} \in \{1, 2\}$, S_{b} inputs $(\mathsf{shuffle}, D^{(\mathsf{b})})$ where $D^{(\mathsf{b})} := (d_i^{(\mathsf{b})})_{i\in[C]}$, with $d_i^{(\mathsf{b})} \in G$ (G is defined as a product group: $G = G_1 \times \ldots \times G_\mu$).

Initialize. For $(\mathsf{b}_1, \mathsf{b}_2) \in \{(1, 2), (2, 3), (1, 3)\}$, S_{b_1} and S_{b_2} jointly sample (only one of the corresponding server samples and sends privately to the other server) a permutation $\pi_{b_1 b_2}$ and a random vector $R_{b_1 b_2} \in G^C$.

Shuffling.

1. $\mathcal{S}_2$ computes $A := \pi_{23}(\pi_{12}(D^{(2)}) + R_{12}) + R_{23}$ and sends A to $\mathcal{S}_1$.
2. $\mathcal{S}_1$ computes $B := \pi_{12}(D^{(1)}) - R_{12}$ and sends B to $\mathcal{S}_3$. It also computes $A' := \pi_{13}(A) - R_{13}$.
3. $\mathcal{S}_3$ computes $B' := \pi_{13}(\pi_{23}(B) - R_{23}) + R_{13}$.

Output. $\mathcal{S}_1$ outputs A' and S_3 outputs B'.

Figure 6: The protocol of oblivious random shuffling.

The formal privacy of the protocol in the semi-honest server model is proven in Theorem C.1 in Section C.1. Informally, what we prove is that the views of any one semi-honest party can be perfectly simulated. At the initialization phase, each pair of parties jointly samples a random permutation and a random mask vector. These can be simulated by uniformly sampling and sending to the adversaries random permutations and mask vectors. The simulation of shuffling phase is done as follows:

- *Corrupt $\mathcal{S}_1$:* The only message that $\mathcal{S}_1$ receives is $A := \pi_{23}(\pi_{12}(D^{(2)}) + R_{12}) + R_{23}$ from $\mathcal{S}_2$, where π_{23} and R_{23} are not known to S_1. Since the random vector R_{23} masks the permuted shares, A is indistinguishable from a random vector from S_1's view. The simulator can replace it with a random vector of same size.
- *Corrupt $\mathcal{S}_2$:* $\mathcal{S}_2$ receives no messages simulation is trivial.
- *Corrupt $\mathcal{S}_3$:* The only message that $\mathcal{S}_3$ receives is B from $\mathcal{S}_2$. As is the same situation to $\mathcal{S}_1$'s, B is indistinguishable from a random vector from $\mathcal{S}_3$'s view, so the simulator can replace it with

a random vector of the same size.

3.3 Secure Modulo Conversion for Sum

So far, we only worked with categorical attributes to build histograms, but our attributes can also be numerical. A core problem with summing numerical attribute values is how to prevent malicious clients from secret sharing unrealistic values with the helper servers, while still being able to compute the value of a possibly large sum in MPC?

If we want to do this with minimal client overhead, a natural approach is to have the client secret share its values over a small field, thus limiting the contribution of any one client, and then have the servers "lift" these shares into a larger field where we can efficiently perform additions. There are two ways to do this: (1) Boolean secret share with a fixed length and lift to a larger field, or (2) arithmetic secret share within a range (represented with small modulus) and lift to a larger field.

Let the client value be d_i between 0 and $(p-1)/2$. It is represented as ℓ-bit string boolean shares of d_i where $p < 2^\ell$ odd integer, determining the range for d_i.

Boolean-to-Arithmetic (B2A) Conversion. The Boolean-to-Arithmetic (B2A) conversion used in [1] is claimed to be more efficient than Arithmetic-to-Arithmetic (A2A) conversion. Clients secret-share values as $d_i = d_i^{(1)} \oplus d_i^{(2)}$, with Boolean shares limited to ℓ bits. This naturally restricts the size of the clients' inputs. A switch to high-precision arithmetic is needed to compute sums, since sum protocols over Boolean shares are too costly.

The current most efficient B2A technique in a 2PC semi-honest setting is due to ABY [16]. For example, to convert a pair of ℓ-bit Boolean shares, Prio+ [1] uses ℓ instances of OT, each communicating $\frac{\ell+1}{2}$ bits on average. Hence, the communication complexity for converting ℓ-bit Boolean shares to ℓ-bit arithmetic shares is $O(\ell^2)$. A sum scheme using B2A conversion is given in [1, pp. 14–18].

OTs in the ABY construction require pre-computations in an offline phase. One could alternatively use bit-wise multiplications, which also require pre-computations. The pre-computations can be entirely omitted with a third honest and non-colluding randomness generator server.

Arithmetic-to-Arithmetic (A2A) Conversion. Each client secret shares its value d modulo an integer p, as $d = d^{(1)} + d^{(2)} \bmod p$. Again, as long as p is appropriately chosen, malicious clients simply cannot share very large (larger than p) values. To compute the sum, the shares need to be lifted to a larger domain. We argue that this can be cheap using techniques from Quotient Transfer [31, 32].

Briefly, we will describe how to convert secret shares of a value d (modulo a small odd integer p) into shares of the same value modulo a large odd modulus p'. We start with the following observation: Assume that the client submits the shares of $d = d^{(1)} + d^{(2)} \bmod p$ where $d < p/2$. The servers can deduce shares $x^{(\mathsf{b})} = 2d^{(\mathsf{b})} \bmod p$ such that $2d = x^{(1)} + x^{(2)} \bmod p$. This implies

$$2d = x^{(1)} + x^{(2)} - q \cdot p\,, \tag{1}$$

where $q \in \{0, 1\}$. For sure $2d$ is even, thus q is equal to the XOR of the least significant bits (lsb) of $x^{(1)}$ and $x^{(2)}$:

$$\begin{aligned} q &= \mathrm{lsb}(x^{(1)}) \oplus \mathrm{lsb}(x^{(2)}) \\ &= \mathrm{lsb}(x^{(1)}) + \mathrm{lsb}(x^{(2)}) - 2 \cdot \mathrm{lsb}(x^{(1)}) \cdot \mathrm{lsb}(x^{(2)})\,. \end{aligned} \tag{2}$$

This suggests that if we know how to compute secret shares mod p' of q given the lsb of $x^{(\mathsf{b})}$, then we can compute the modulo conversion easily. In other words, sharing the secret bit q with shares in p' is enough to lift the shares of d from modulus p into large modulus p'. This technique is called Quotient Transfer (QT) and was first used in [31].

Let $q^{(1)}$ and $q^{(2)}$ be the shares of q modulo p'[6] We want to obtain two shares $(d^{(\mathsf{b})})' = \frac{x^{(\mathsf{b})} - q^{(1)}p}{2} \bmod p'$ such that $d = (d^{(1)})' + (d^{(2)})' \bmod p'$. If we share q between the servers using modulus p', then the conversion of d to the larger field can be done locally. For that, we need a 2-party protocol that multiplies the least-significant bits, *i.e.*, takes $a = \mathrm{lsb}(x^{(1)})$ from $\mathcal{S}_1$ and $b = \mathrm{lsb}(x^{(2)})$ from $\mathcal{S}_2$ and outputs $m^{(1)}$ to $\mathcal{S}_1$ and $m^{(2)}$ to $\mathcal{S}_2$, where $m^{(1)} + m^{(2)} \bmod p' = \mathrm{lsb}(x^{(1)}) \cdot \mathrm{lsb}(x^{(2)})$. Finally, each server b computes $q^{(\mathsf{b})} = \mathrm{lsb}(d^{(\mathsf{b})}) - 2 \cdot m^{(\mathsf{b})} \bmod p'$ and $(d^{(\mathsf{b})})' = \frac{x^{(\mathsf{b})} - q^{(\mathsf{b})}p}{2} \bmod p'$. The total cost of this protocol is a single OT per client input. It is a factor of ℓ faster than the Prio+ proposal, because it requires only one OT per record instead of ℓ OTs, where ℓ could be very large, for example, 64. We describe our 3-party OT protocol in Figure 11.

Protocol $\Pi_{\mathsf{A2A\ Convert}}$

Participants. Two helper servers $\mathcal{S}_1$ and $\mathcal{S}_2$. Note that this protocol utilizes an oblivious transfer (OT) protocol as a subroutine and our instantiations of such OT runs with **three** servers for better performance, even though it could in theory run with two servers.

Initialize. $\mathcal{S}_1$ and $\mathcal{S}_2$ receive shares of records $D^{(1)} := \{d_i^{(1)}\}_{i\in[n]}$ and $D^{(2)} := \{d_i^{(2)}\}_{i\in[n]}$, resp.

1. Each server b obtains $d_i^{(\mathsf{b})}$ in $\{0, \ldots, p-1\}$. Compute $x_i^{(\mathsf{b})} = 2d_i^{(\mathsf{b})} \bmod p$ so that it satisfies $2d_i = x_i^{(1)} + x_i^{(2)} \bmod p$.
2. Each server b computes $\mathrm{lsb}(x_i^{(\mathsf{b})})$. Then, they run an OT protocol (Figure 11) with their least significant bits to compute the shares mod p' of $\mathrm{lsb}(x_i^{(1)}) \cdot \mathrm{lsb}(x_i^{(2)})$. At the end, $\mathcal{S}_1$ obtains $m_i^{(1)}$ and $\mathcal{S}_2$ obtains $m_i^{(2)}$ such that $m_i^{(1)} + m_i^{(2)} \bmod p' = \mathrm{lsb}(x_i^{(1)}) \cdot \mathrm{lsb}(x_i^{(2)})$.
3. Each server b computes $q_i^{(\mathsf{b})} = \mathrm{lsb}(x_i^{(\mathsf{b})}) - 2m_i^{(\mathsf{b})} \bmod p'$.
4. Each server b computes $(d_i^{(\mathsf{b})})' = \frac{x_i^{(\mathsf{b})} - q_i^{(\mathsf{b})}p}{2} \bmod p'$.

Output. $\mathcal{S}_1$ and $\mathcal{S}_2$ output record shares $D^{(1)} := \{(d_i^{(1)})'\}_{i\in[n]}$ and $D^{(2)} := \{(d_i^{(2)})'\}_{i\in[n]}$, respectively, which are lifted shares of d_i modulo p'.

Figure 7: Secure modulo conversion with Quotient Transfer.

[6] In our protocol a malicious client can potentially submit an input in the range $\{-p/2, \ldots, 0, \ldots, p-1\}$. Thus, to make sure that we do not overflow, we require $p' \geq 2pC + 2M_{\mathsf{sum}}$ where M_{sum} is a bound on the noise added by the helper servers. This allows us to guarantee that the difference between the true sum with no malicious clients and the sum with malicious clients is in $\left[-N\frac{p}{2}, Np\right]$ where N is the # of malicious clients. We show how these bounds obtained in Section D.2.

4 Precio

The goal of the Precio protocol is to arrange a set of reports D collected from C clients into buckets according to a subset of encoded attributes in order to compute aggregate function, while preserving privacy of the individual reports. For attribute m of ℓ_{m} bits, there are $L_{\mathsf{m}} = 2^{\ell_{\mathsf{m}}}$ buckets in total. Our full protocol involves four parties: a Reporting Origin $\mathcal{R}$ and three helper servers $\{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$.

We assume that no two helper servers in the aggregation system collude. We build our system with three helper servers where two of them, say $\mathcal{S}_1$ and $\mathcal{S}_2$, receive the secret shared inputs, and other two, say $\mathcal{S}_1$ and $\mathcal{S}_3$, output the results of the histogram computations. Internally, two of the servers run DP noise generation and all three run the oblivious random shuffling protocol.

The input to our full protocol is a dataset of reports $D = (d_i)_{i\in[C]}$, initially secret shared between two non-colluding servers. To ensure that the servers get the same ordering of reports, each clients may attach an ephemeral ID along with the encryption of their shared report (under the public key of the servers), before passing them to the servers. This is equivalent to including a *Leader* server, which is a trusted (for correctness) entity whose only job is to maintain the order of the reports as in the Internet-Draft [22]. At the end of the protocol, two servers obtain a new secret shared vector $D_{\mathsf{priv}} = (d_i)_{i\in[C+n']}$, where all reports with the same attribute values ("buckets") are stored consecutively in D_{priv}. Note that the size of D_{priv} becomes $C + n'$, where n' represents appended dummy reports, ensuring that the outputs revealed to the $\mathcal{R}$ and helper servers are differentially private.

4.1 Private Histogram Protocol Description

We describe our Precio protocol as a full procedure $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ in Figure 8. Let $D = (d_i \in G)_{i\in[C]}$ be the dataset of reports collected from clients. $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ takes D and a query index m (to indicate on which attribute the histogram is built) as an input.

The procedure is triggered by $\mathcal{R}$, with helper servers $\mathcal{S}_1$ and $\mathcal{S}_2$ receiving the shares of the reports as $D^{(1)} = \left(d_i^{(1)}\right)_{i\in[C]}$ and $D^{(2)} = \left(d_i^{(2)}\right)_{i\in[C]}$, such that $d_i = d_i^{(1)} + d_i^{(2)}$ for all $i \in [C]$.

After secret sharing the reports in D, the first step is to achieve differential privacy by $\mathcal{S}_1$ and $\mathcal{S}_2$ independently adding dummy reports as noise, which is done by invoking the noise generation subroutine, as defined in Figure 4. This step inputs the shares of the dataset and the query attribute index m, and outputs a new dataset with appended dummy reports $D_{\mathsf{priv}} = (d_i)_{i\in[C+n']}$.

After generating D_{priv}, the servers $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ execute a 3-party oblivious random shuffling protocol, as defined in Figure 6. At the end of this step, the real reports and dummy reports will be mixed up by the random permutation so that none of the servers can trace back any report to the original report, nor can they distinguish between the authentic reports and dummy reports. The output will be a permuted dataset $D_{\mathsf{priv\_perm}} = (d'_i)_{i\in[C+n']}$, secret shared between the helper servers $\mathcal{S}_1$ and $\mathcal{S}_3$.

For each $i \in [C + n']$, the servers reveal the selected attribute $d_i[\mathsf{m}]$ and bucketize the reports into buckets $\{\mathcal{B}_j\}$ for $j \in G_{\mathsf{m}}$ according to the attribute value.

Finally, we support the pruning of some buckets for a given threshold parameter to our algorithm so that all the buckets with size less than a pruning threshold t, as well as the dummy bucket $\mathcal{B}_\perp$, are discarded. The meaning of this threshold is discussed in Section 4.4. After pruning, since the output shares remain with $\mathcal{S}_1$ and $\mathcal{S}_3$, we apply an organizing step to put the shares back in place, (*i.e.*, reshare them to $\mathcal{S}_1$ and $\mathcal{S}_2$, to run the protocol again if needed (see below). While this

resharing is not really necessary in implementations, we include it in the protocol description in order to be able to use $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ repeatedly.[7]

The protocol outputs the counts for each bucket, shifted by the public parameter $2M$ as described in Section 3.1.

For a pictorial representation of dummy reports addition and oblivious shuffling see Figure 14 (in Appendix F).

4.2 Private Sum Computation

We described how to lift a secret shared value from a small modulus to a larger one in Section 3.3. The helper servers run the modulus conversion for each record using modulus p', which is large enough so that the final sum with Gaussian noise added is almost always between $-p'/2$ and $p'/2$. We set $p' \geq 2pC + 2M_{\mathsf{sum}}$. Then, each server locally adds the shares of every value to be summed to obtain sum of all shares. Each server adds Gaussian noise modulo p' with parameter σ_{sum} (rounded to the nearest integer). The local sums are revealed to compute the noisy sum of the numerical attribute value modulo p'. We require that $p' \geq 2pC + 2M_{\mathsf{sum}}$ to make sure that the difference between the results and the sum from honest clients is in $\left[-Np/2, Np\right]$ where N is the number of malicious clients.

4.3 Layered Protocol

For reports with multiple attributes, our protocol supports aggregation on an arbitrary set of attributes by recursively invoking $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$. For example, our method enables running queries such as

$$\mathsf{SELECT}\ \ \mathsf{COUNT}(\text{Ads Category})\ \ \mathsf{FROM}\ \ \text{D}$$
$$\mathsf{WHERE}\ \ \text{Gender} = \text{“She”}\ \ \mathsf{GROUP\ BY}\ \ \text{Ads Category}$$

as long as there is enough data in the corresponding bucket. Moreover, such an approach enables a performance gain for attributes with large report sizes. For example, if an attribute has 32 bits, we can apply layered pruning to speed up the protocol. More importantly, if the domain of the attributes is sparse, say we have 2^{16} reports represented with 32 bits, splitting the attribute into three smaller attributes (e.g. 10, 10, and 12 bits) and pruning the buckets after each layer allows us to run the protocol more efficiently even for very large reports. A formal description of the layered algorithm is given in Algorithm 1. Its complexity and further optimizations are discussed in Appendix B.

When the first layer is done with m_1, only the corresponding bits are revealed. Then, the procedure bucketizes on the next attribute m_2 by generating noisy reports, creating buckets for each possible value of m_2, and setting the values for other attributes to dummy values.

Example. We continue with our toy example. The reports consist of 5 bits representing two attributes with 2 and 3 bits, respectively. We bucketize with respect to the first attribute, ‘Gender’ (2 bits), into 3 buckets $B = \{\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3\}$ after discarding the dummy bucket, prune the buckets which have number of reports below some threshold t, $B' = \{\mathcal{B}_i : |\mathcal{B}_i| \geq t\}$, and continue bucketization for

[7] This resharing step is not necessary for functionality or security, but it simplifies the presentation in the layered setting as we can directly repeat the protocol as written. Alternatively, for the next layer we could run the protocol but with the roles of S_2 and S_3 reversed.

Protocol $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$

Parameters. C as the total number of collected reports. Pruning threshold t. Parameters $\epsilon_{\mathsf{buckets}}, \epsilon_{\mathsf{flush}}$ and $M_{\mathsf{buckets}}, M_{\mathsf{flush}}$ for Π_{NoiseGen}. For a single layer and for the last layer of multiple attributes histogram, $\epsilon_{\mathsf{flush}}$ will be set to 0.

Input. A query index m to indicate which attribute to bucketize on; shares of $\mathcal{S}_1$ and $\mathcal{S}_2$ as $D^{(1)} = \left(d_i^{(1)}\right)_{i\in[C]}$ and $D^{(2)} = \left(d_i^{(2)}\right)_{i\in[C]}$, such that $d_i = d_i^{(1)} + d_i^{(2)}$; a threshold t.

Initialization. Each server receives and decrypts their share. They discard the shares if they are not in G.

Precio.

1. (**Differential privacy**) $\mathcal{S}_1$ and $\mathcal{S}_2$ invoke the protocol $\Pi_{\mathsf{NoiseGen}}[\epsilon_{\mathsf{buckets}}, \epsilon_{\mathsf{flush}}, M_{\mathsf{buckets}}, M_{\mathsf{flush}}]$ from Figure 4 on input $(\mathsf{dp}, D^{(1)}, \mathsf{m})$ and $(\mathsf{dp}, D^{(2)}, \mathsf{m})$ to get shares of the database with n' dummy reports added, $D^{(1)}_{\mathsf{priv}} = \left(d_i^{(1)}\right)_{i\in[C+n']}$ and $D^{(2)}_{\mathsf{priv}} = \left(d_i^{(2)}\right)_{i\in[C+n']}$ respectively.
2. (**Random shuffling**) $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ invoke the protocol Π_{RandShuf} from Figure 6 on inputs $(\mathsf{shuffle}, D^{(1)}_{\mathsf{priv}})$, $(\mathsf{shuffle}, D^{(2)}_{\mathsf{priv}})$. It outputs $D^{b}_{\mathsf{priv\_perm}} = \left(d_i'^{b}\right)_{i\in[C+n']}$ to $\mathcal{S}_{\mathsf{b}}$ for $b \in \{1, 3\}$.
3. (**Bucketizing**) For $i \in [C + n']$, $\mathcal{S}_1$ and $\mathcal{S}_3$ reveal their shares of $d'_i[\mathsf{m}]$. They allocate L_{m} empty buckets $\{\mathcal{B}_j\}$, one for each $j \in G_{\mathsf{m}}$. For $i \in [C + n']$, $\mathsf{b} \in \{1, 3\}$, $\mathcal{S}_b$ puts the report $d_i'^{b}$ into the corresponding bucket according to the value of $d'_i[\mathsf{m}]$.
4. (**Pruning and organizing**) For each bucket revealed in previous step, discard the bucket $\mathcal{B}_j$ if $|\mathcal{B}_j| < t$ and discard the dummy bucket (indexed as $j = \perp$) along with all the reports in them. For the shares of noisy database from $\mathcal{S}_1$ and $\mathcal{S}_3$, $\mathcal{S}_3$ generates two secret shares of $D^{3}_{\mathsf{priv\_perm}}$ and sends the shares to $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively; $\mathcal{S}_1$ combines the share he receives from $\mathcal{S}_3$ with $D^{(1)}_{\mathsf{priv\_perm}}$ so that $\mathcal{S}_1$ and $\mathcal{S}_2$ have the new shares of the $D_{\mathsf{priv\_perm}}$ (see footnote 7).

Output. Output the attribute value of index m for each bucket which remains after pruning. Servers keep a private output which shares datasets of reports for each bucket so that they can be used as input for further instances of the protocol.

Figure 8: Our Histogram Protocol from Oblivious Shuffling.

the next attribute for each remaining bucket in B'. We will detail the meaning and computation of the threshold shortly in Section 4.4.

When we open the first attribute (*i.e.*, only the first two bits), the bucket $\mathcal{B}_1$ contains

$$\{00\,010\,25,\ 00\,000\,19,\ 00\,010\,64,\ 00\,111\,0,\ 00\,111\,0,\ 00\,111\,0\},$$

yielding 6 reports where the second and third attributes are still secret shared.

Next, suppose the protocol goes to the second layer on $\mathcal{B}_1$, where bucketization is run on the

Algorithm 1 Layered $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$

Input: A list of attribute indices $\mathcal{M} = \{\mathsf{m}_1, \ldots, \mathsf{m}_\lambda\}$; a shared dataset $D^{(1)}$ and $D^{(2)}$ for $\mathcal{S}_1$ and $\mathcal{S}_2$. label and i are reserved for the recursion and set to $\perp$ by default.
Output: Histogram on attributes in $\mathcal{M}$.

1: **procedure** LAYERED($\mathcal{M}$, label , $i, D^{(1)}, D^{(2)}$)
2: **if** $(\mathsf{label}, i) = (\perp, \perp)$ **then**
3: Set $\mathsf{label} = \text{null}, i = 1$.
4: **end if**
5: **if** $i \leq \lambda$ **then**
6: Call $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ with inputs $(\mathsf{m}_i, D^{(1)}, D^{(2)})$.
7: **for each** produced bucket $\mathcal{B}_j$ **do**
8: Set v to the attribute value $\mathcal{B}_j$.
9: Set $D^{(1)}$ and $D^{(2)}$ to the shares of the bucket.
10: Call LAYERED($\mathcal{M}$, $\mathsf{label} \mid v, i+1, D^{(1)}, D^{(2)}$).
11: **end for**
12: **else**
13: Output label.
14: Subtract the parameter $2M$ (used in $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$) from the number of reports in $D^{(1)}$ and output the result.
15: **end if**
16: **end procedure**

second attribute. For this example, we focus on the bucket $\mathcal{B}'_3$, which counts the reports where $d_i[1] = 00$ and $d_i[2] = 010$. Suppose $\mathcal{S}_1$ sampled 2 and $\mathcal{S}_2$ sampled 1 to add in $\mathcal{B}'_3$, then the output of that bucket will have $3+2+1$ reports, where 3 comes from real reports (neither $\mathcal{S}_1$ nor $\mathcal{S}_2$ know these true report counts), and additional 2 and 1 come from the dummy reports. In this case a query that asks for counts, where first attribute is 00 and second attribute is 010, will output 6, instead of 3. Finally, in the second layer, more dummy reports are added with $d_i[2] = 111$ and the records are shuffled and the second attribute values are revealed. After revealing, the bucket corresponding to 111 and the buckets with less than t counts will be discarded.[8]

4.4 Differential Privacy and Pruning

The final protocol, detailed in Algorithm 1, conducts a depth-first exploration of buckets layer by layer. Layers are defined either by each logically separate attribute, or by splits of a large logical attribute into many smaller physical attributes (sub-strings). Accordingly, the algorithm outputs noisy histograms either per layer or when the algorithm ends. Depending on report distribution, at each layer an analyst may choose to prune buckets with a size below a threshold. Pruning is often crucial, as without it long-tailed attribute value distributions necessitate an extensive number of iterations at each layer.

Differential Privacy Parameters. Our noisy histogram protocol applies DP in two distinct ways: 1) dummy reports are added for each (non-dummy) bucket at each layer to provide

[8]Note that in practice we would need to subtract the shift parameter $2M$ from the bucket counts, but we omit it here for the sake of simplicity. For details, refer back to Section 3.1.

$(\sigma_{\mathsf{buckets}}, \delta_{\mathsf{buckets}})$-DP for bucket counts; 2) dummy reports are added for the dummy bucket to provide $(\sigma_{\mathsf{flush}}, \delta_{\mathsf{flush}})$-DP for the total number of dummies added. These are then *flushed* out with the other dummies when processing the subsequent layer. This separation into two sets of DP parameters improves both efficiency and privacy. For example, the flush noise has no impact on accuracy, so we can use *less* of our privacy budget for that. In the end, these two sets of DP parameters will determine the total DP parameters (ϵ, δ). For DP composition, we leverage the work of Gopi *et al.* from [27, 28]. We compute the parameters using their software from [26].

To show how we adjust our parameters, we will work through some examples with different number of layers. Our goal is to reach (ϵ, δ)-DP for the layered protocol. We start with a single attribute: we only run one layer. We do not generate any dummies for the dummy bucket $\mathcal{B}_\perp$, because it is a reserved bucket and there is no original report that falls in that bucket. In this case we only have to set our parameters with $(\sigma_{\mathsf{buckets}}, \delta_{\mathsf{buckets}})$. The neighboring notion of DP suggests that one report is replaced with another report with different attribute value. This implies that there are two non-dummy buckets that differ in counts and they both contribute to the ϵ equally. If we want to achieve $(\epsilon, \delta) = (2, 2^{-40})$, we use $(\sigma_{\mathsf{buckets}} = 4.75, \delta_{\mathsf{buckets}} = 2^{-40})$. Then, we set $M_{\mathsf{buckets}} = 35$.

For two layers, we will have two attributes. Again, our neighboring notion means that one report is replaced with another report. Suppose the reports differ in both attributes. At the first layer, we reveal the noisy counts of each bucket $\mathsf{buckets}_i$. Since the two reports differ in the first attribute, then two regular buckets are impacted at the first layer and the sensitivity will be 2.

In the second layer, we reveal the noisy counts of each attribute bucket and dummy bucket. For the attribute buckets, we have again sensitivity 2, since the differences in the two reports can affect at most two buckets. Analyzing the privacy impact of the dummy bucket is a bit less straightforward, since it is not directly affected by either of the records. Let m_1, m_2 be the attributes on which we partition in the first and second layer respectively. For each $i \in G_{m_1}$, let the corresponding bucket be $\mathcal{B}_i$. Suppose the noisy count for $\mathcal{B}_i$ is c_i, and let the exact value for that bucket be e_i. Then, we can represent the exact dummy count as $c_i - e_i$, which we can see as a function computed on the dataset, where the choice of the function depends on the previous result, c_i. In the second layer of our protocol, the noisy count for the dummy bucket within $\mathcal{B}_i$ can be seen as a DP computation of $c_i - e_i$. If we consider counts for all the dummy buckets in the second layer, the exact dummy count $c_i - e_i$ will differ for at most two of the buckets $\mathcal{B}_i$, so again the sensitivity of this function evaluated over all the layer 2 dummy buckets is 2.

So, we have two queries of sensitivity 2, which we compute using $(\sigma_{\mathsf{buckets}}, \delta_{\mathsf{buckets}})$-DP and one query of sensitivity 2, which we compute using $(\sigma_{\mathsf{flush}}, \delta_{\mathsf{flush}})$-DP. Thus, if we want to achieve $(\epsilon, \delta) = (2, 2^{-40})$, we use $\sigma_{\mathsf{buckets}} = 6.8$ and $\sigma_{\mathsf{flush}} = 20$ computed with private accountant tools [26]. Then, we set $M_{\mathsf{buckets}} = 50$ and $M_{\mathsf{flush}} = 250$.

Finally, if we have λ layers, we have (ϵ, δ)-DP coming composition of λ sensitivity 2 queries evaluated using $(\sigma_{\mathsf{buckets}}, \delta_{\mathsf{buckets}})$-DP and $(\lambda - 1)$ sensitivity 2 queries evaluated using $(\sigma_{\mathsf{flush}}, \delta_{\mathsf{flush}})$-DP. We give sample parameters in Table 2 and detailed analysis in Section C.2.

Pruning Parameters. We treat pruning as a utility factor that the analysts can set as they wish. Namely, we let the analyst decide a pruning threshold t_{true} for the true counts, with some error probability q (picked again by the analyst). This means they may lose true counts which are more than t_{true}, except with probability at most q (there could be false positive/negatives due to the noise). Then, the pruning threshold t is computed as follows. Let n be a noise generated from Gaussian noise, then $M+n$ many dummy reports are added to a bucket. We want to find a pruning threshold t so that the true count is at least t_{true} for a fixed probability q.

	$\lambda = 1$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 16$
$(\sigma_{\mathsf{buckets}}, M_{\mathsf{buckets}})$	$(4.75, 35)$	$(6.8, 50)$	$(8.75, 65)$	$(9.75, 72)$	$(19, 146)$
$(\sigma_{\mathsf{flush}}, M_{\mathsf{flush}})$	N/A	$(20, 250)$	$(20, 250)$	$(40, 300)$	$(150, 1125)$
t	0	1053	1069	1079	1150

Table 2: Parameters for λ layers and $(\epsilon, \delta) = (2, 2^{-40})$-DP, for $C = 10^7$, $t_{\mathsf{true}} = 1000$, and error threshold $q = 1\%$.

To this end, we look at the following equation for q:

$$q = \Pr\left[t_{\mathsf{true}} + 2M + n^{(1)} + n^{(2)} < t\right].$$

Let $s = t - t_{\mathsf{true}} - 2M$. We know that $n^{(1)} \sim \mathcal{N}(0, \sigma^2)$ and $n^{(2)} \sim \mathcal{N}(0, \sigma^2)$, so $n^{(1)} + n^{(2)} \sim \mathcal{N}(0, 2\sigma^2)$ and

$$q = \frac{\lambda C}{t_{\mathsf{true}}} \Pr[n^{(1)} + n^{(2)} < s] = \frac{\lambda C}{t_{\mathsf{true}}} \mathrm{CDF}_{\sigma\sqrt{2}}(s).$$

Thus, $t = \lfloor (t_{\mathsf{true}} + 2M + \sigma\sqrt{2}\,\mathrm{invCDF}\left(\frac{q t_{\mathsf{true}}}{\lambda C}\right)\rceil$.

For example, our algorithm can be used to query all the buckets with $C = 10^7, \lambda = 8$, and pruning threshold $t_{\mathsf{true}} = 1000$ for true counts with 1% error, *i.e.*, $q = 0.01$. For $t_{\mathsf{true}} = 1000$, $\sigma = 13.5$, $q = 0.01$, we obtain $M = 101$, which gives $t = 1104$. We use t to prune the noisy buckets (before correcting the result by $2M$). This implies that $s = -98$ and the probability that $n^{(1)} + n^{(2)} < -98$ is 0.01 with the Gaussian distribution.

5 Performance Evaluation

We implement layered Precio in Rust and show its performance with several experiments. First, we demonstrate the efficiency of our protocol by generating both full-domain and subset histograms and compare the results to prior works. Second, we run micro-benchmarks to demonstrate the bottlenecks of our implementation. The implementation is available at GitHub.com/Microsoft/Precio.

We run all three helper servers $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ on a single virtual machine. We use an Azure Standard E16ads v5 VM with 16 vCPUs (running at 2.45 GHz) and 128 GB RAM. Our experiments did not use this machine to its full extent. Namely, despite the many available vCPUs, *all of our helper servers run on a single thread.*

We do not benchmark the client's computation (secret sharing a single report) or the client-to-server communication (sending the shares to $\mathcal{S}_1$ and $\mathcal{S}_2$).

5.1 Existing Proposals

The existing proposals for web conversion measurements include DPF-based protocols [8, 7] and Prio [14, 1]. They are further discussed in Appendix E. We will compare our work with the DPF-based protocol for full-domain histograms and the iDPF-based protocol for subset histograms. Note that full-domain histograms get no benefit from iDPF over DPF.

5.2 Constructing a Full Histogram

We benchmark the performance of our protocols for generating differentially private full histograms for report size $\ell \in \{16, 18, 20, 22\}$ with $\lambda = 1$ parameters from Table 2. For each execution, $C = 10\,000\,000$ reports are generated from a Zipf-distribution with parameter 1.03. The helper servers output a full histogram consisting of 2^ℓ buckets.

We compare the performance of our protocol with the DPF implementation in [25] that outputs a full histogram. This is a non-interactive protocol and the computational cost for the two helper servers are equal. The benchmark of [25] runs the DPF algorithm for one server and records the average time for processing one report. Multiplying this average time with $C = 10\,000\,000$ reports results in the total running time.

The results are shown in Table 3. For $\ell \in \{16, 18, 20, 22\}$, our protocol is at least **four orders of magnitude more efficient** than the DPF-based protocol for generating full histograms from short reports.

ℓ	16	18	20	22
Precio time (seconds)	6.46	16.9	64.28	310.80
Precio comm. (MB)	233	453	1334	4857
DPF [8] time (days)	1.15	4.75	19	76

Table 3: Running times (single thread) and communication for Precio compared to the running times for DPF.

5.3 Constructing a Subset-Histogram via Pruning

We continue with a benchmark of subset histogram aggregation. In practice, generating a histogram of the whole domain is not always meaningful. The domain size can be large and the reports are usually not uniformly distributed. We tested our protocols for two layers with various different Zipf-distribution parameters from 1.0 to 1.5. Our results are shown in Figure 9 and Figure 10. As was expected, for layered histograms the shape of the attribute value distribution has a huge impact on performance. The distribution makes no difference for a simple single-layer histogram.

32-bit Attributes. Next, we follows the setting of [7] and sample 32-bit logical attributes from a specific distribution. For such a large attribute many buckets end up empty or only contain a small number of reports. This corresponds to the setting where the reporting origin is only interested in popular reports and wants to ignore rarely appeared outliers.

We demonstrate the performance of Precio in this scenario as follows. The parameters for (ϵ, δ)-DP are chosen according to the analysis in Section 4.4. We first synthesize a dataset of input 32-bit reports with various C and pruning thresholds 552 and 1053 (obtained from our analysis in Section 4.4 with $t_{\mathsf{true}} = \{500, 1000\}$), so the attributes are sparsely encoded in the reports. The reports follow a Zipf-distribution, with Zipf-parameter 1.03. We run Algorithm 1 with 2-layer Precio ($\lambda = 2$). Our results are shown in Table 4.

This demonstrates how even large 32-bit attributes can be explored by breaking them down into manageable chunks with layered Precio.

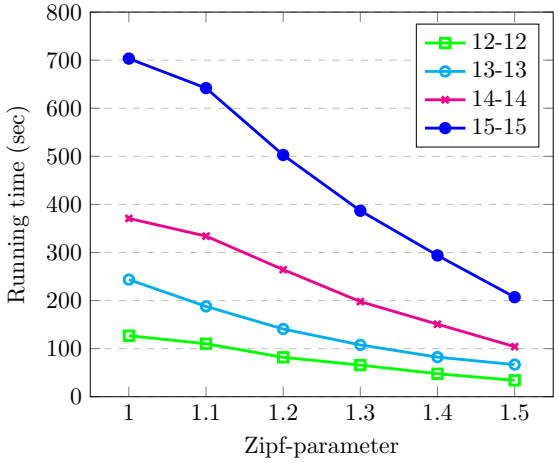

Figure 9: The running time and communication cost for 2-layer Precio for different Zipf parameters and attribute sizes for $C = 10\,000\,000$ and privacy parameters from Table 2 with $\lambda = 2$ and the threshold set to 1053.

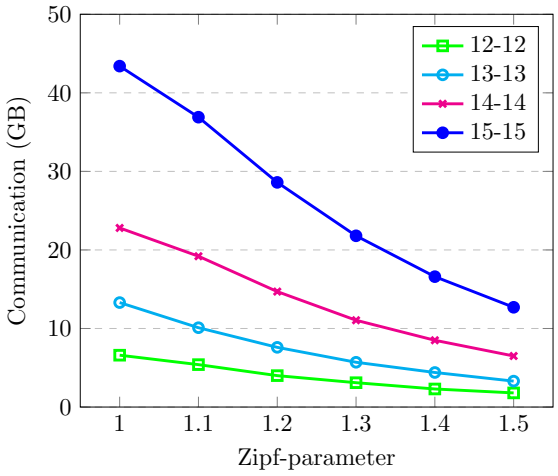

Figure 10: Total server-to-server communication cost for 2-layer Precio for different Zipf-parameters and attribute sizes for $C = 10\,000\,000$ and privacy parameters from Table 2 with $\lambda = 2$ and the threshold set to 1053.

	$C = 400\text{K}$		$C = 1\text{M}$		$C = 10\text{M}$	
t	Time	Comm.	Time	Comm.	Time	Comm.
1053	39.7	2.3	93.95	5.4	966	55
552	72.13	4.2	175.64	10	1912	111

Table 4: Performance of building histograms on 32-bit logical attributes by splitting into two 16-bit physical attributes ($\lambda = 2$) with different pruning thresholds. The times are in seconds and the communication in GB.

5.4 Heavy-Hitters

We also compare the performance of our protocol with the subset-histogram appearing in the end-to-end performance evaluation of iDPF [7]. In the experiment, $C = 400\,000$ input reports are sampled from a Zipf-distribution, with Zipf-parameter 1.03 and artificially limited support of 10 000. Their bit-length of input reports is 256 and their prune threshold is set to $t = \frac{C}{1000}$.

This experiment in [7] is done between two servers with 32 vCPUs each and utilizes a network with 61.9 ms round-trip latency. Their protocol takes around 53 minutes to generate a subset-histogram.

For Precio, we use $\lambda = 16$ with parameters from Table 2 and bucketize the reports on 16-bit attributes at each layer. It takes only 251 seconds to generate the subset-histogram with 12847 output buckets, when we keep the privacy parameters as $(\epsilon, \delta) = (2, 2^{-40})$.

Interestingly, because the 16-bit physical attributes are very large and the total number of noise reports added at each layer is proportional to 2^{16}, the number of noise reports completely overshadows a C of 400 000. In fact, even with $C = 10\,000\,000$ our performance remains the same, whereas the complexity of iDPF behaves quadratically in the number clients, increasing by a factor of 625. In their experiments, [7] mitigates this to reduce the increase to just linear by simultaneously increasing the pruning threshold, whereas we can retain the threshold at $t = 400$.

5.5 Sums

We benchmark sums for numerical attributes with different C and input modulus p, which defines the upper bound for the inputs. For a 14-bit modulus p and 10 000 000 reports, Precio takes 43 seconds (on a single thread) and requires 540 MB of server-to-server communication.

The running time increases linearly in the number of reports. In the same setting but 100 000 000 reports, Precio takes 440 seconds and requires 5.4 GB of communication.

6 Conclusions

We have presented an efficient 3-party protocol, Precio, for computing privacy-preserving histogram queries. The basic protocol Figure 8 can be used iteratively to compute histograms for large keys with sparse domains, as described in Algorithm 1.

In addition to other use-cases, we believe our approach presents a viable method for enabling web advertisers to obtain valuable information about their ad campaigns, while still preserving people's privacy with state-of-the-art cryptography and differential privacy. Our protocol is simpler

than prior work and it outperforms prior work in many practical scenarios, as demonstrated by our experiments.

Acknowledgements. We thank Sivakanth Gopi and Sergey Yekhanin for their very helpful and insightful discussions on differential privacy. We thank Wei Dai and Siddharth Sharma for their help with an early-stage prototype. We thank Esha Ghosh for participating in the early discussions on this work. We also thank the anonymous reviewers who reviewed earlier versions of this work and provided helpful comments.

References

[1] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy Preserving Aggregate Statistics via Boolean Shares. *Cryptology ePrint Archive*, 2021.

[2] Apple and Google. Exposure Notification Privacy-preserving Analytics (ENPA) White Paper, Apr 2021.

[3] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548, 2013.

[4] Richard L. Barnes, Christopher Patton, and Phillipp Schoppmann. Verifiable Distributed Aggregation Functions, Apr 2022.

[5] Muhammad Ahmad Bashir and Christo Wilson. Diffusion of User Tracking Data in the Online Advertising Ecosystem. *Proc. Priv. Enhancing Technol.*, 2018(4):85–103, 2018.

[6] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 441–459, New York, NY, USA, 2017. ACM.

[7] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight Techniques for Private Heavy Hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776. IEEE, 2021.

[8] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 337–367. Springer, 2015.

[9] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing: Improvements and Extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016.

[10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. Sepia: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10. USENIX Association, 2010.

[11] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-Shared Shuffle. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 342–372. Springer, 2020.

[12] David L Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[13] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed Differential Privacy via Shuffling. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 375–403. Springer, 2019.

[14] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 259–282. USENIX Association, 2017.

[15] George Danezis, Cédric Fournet, Markulf Kohlweiss, and Santiago Zanella-Béguelin. Smart Meter Aggregation via Secret-Sharing. In *Proceedings of the first ACM workshop on Smart energy grid security*, pages 75–80, 2013.

[16] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY-A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*, 2015.

[17] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[18] Cynthia Dwork, Aaron Roth, et al. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[19] Tariq Elahi, George Danezis, and Ian Goldberg. Privex: Private Collection of Traffic Statistics for Anonymous Communication Networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14. ACM, 2014.

[20] Steven Englehardt. Next Steps in Privacy-Preserving Telemetry with Prio, jun 2019.

[21] Antonio Faonio and Dario Fiore. Improving the Efficiency of Re-randomizable and Replayable CCA Secure Public Key Encryption. In *Applied Cryptography and Network Security*, pages 271–291. Springer International Publishing, 2020.

[22] Tim Geoghegan, Christopher Patton, Eric Rescorla, and Christopher Wood. Privacy Preserving Measurement, March 2022.

[23] Niv Gilboa and Yuval Ishai. Distributed Point Functions and Their Applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 640–658. Springer, 2014.

[24] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play Any Mental Game, or a Completeness Theorem for Protocols with Honest Majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 307–328, 2019.

[25] Google. An Implementation of Incremental Distributed Point Functions in C++, Feb 2022. commit 88c73a78cd61dacba6d8258f13d0f5dc5f1fb0d2.

[26] Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. Privacy Random Variable (PRV) Accountant. `https://github.com/microsoft/prv_accountant`. (accessed: July 10, 2023).

[27] Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. Numerical Composition of Differential Privacy. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 11631–11642, 2021.

[28] Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. Numerical composition of differential privacy, 2021.

[29] Rob Jansen and Aaron Johnson. Safely Measuring Tor. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16. Association for Computing Machinery, 2016.

[30] Marek Jawurek and Florian Kerschbaum. Fault-Tolerant Privacy-Preserving Statistics. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 221–238. Springer, 2012.

[31] Ryo Kikuchi, Dai Ikarashi, Takahiro Matsuda, Koki Hamada, and Koji Chida. Efficient Bit-Decomposition and Modulus-Conversion Protocols with an Honest Majority. In *Australasian Conference on Information Security and Privacy*, pages 64–82. Springer, 2018.

[32] Yi Lu, Keisuke Hara, Kazuma Ohara, Jacob Schuldt, and Keisuke Tanaka. Efficient Two-Party Exponentiation from Quotient Transfer. In *Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy, June 20–23, 2022, Proceedings*, page 643–662. Springer-Verlag, 2022.

[33] Sahar Mazloom and S Dov Gordon. Secure Computation with Differentially Private Access Patterns. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 490–507, 2018.

[34] Meta. Make smarter business decisions with actionable insights. `https://www.facebook.com/business/measurement`. (accessed: July 30, 2023).

[35] Payman Mohassel, Peter Rindal, and Mike Rosulek. Fast Database Joins and PSI for Secret Shared Data. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1271–1287, 2020.

[36] Andrés Muñoz Medina, Umar Syed, Sergei Vassilvtiskii, and Ellen Vitercik. Private Optimization without Constraint Violations. In *International Conference on Artificial Intelligence and Statistics*, pages 2557–2565. PMLR, 2021.

[37] C Andrew Neff. A Verifiable Secret Shuffle and its Application to E-Voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125, 2001.

[38] Web Platform Incubator Community Group. Attribution Reporting API with Aggregatable Reports, May 2022. commit fd75741c4e5c047de7536c02c20cbc903645aa17.

[39] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152, 2015.

[40] IETF working group on Privacy Preserving Measurement. Privacy Preserving Measurement Protocol, May 2022. commit 3aa0e86a4261cd749f5fa0b2569f5a44f482f042.

[41] Andrew Chi-Chih Yao. How to Generate and Exchange Secrets. In *27th Annual Symposium on Foundations of Computer Science*, FOCS'86, page 162–167. IEEE Computer Society, 1986.

[42] Ke Zhong, Yiping Ma, and Sebastian Angel. Ibex: Privacy-preserving ad conversion tracking and bidding. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 3223–3237, 2022.

[43] Ke Zhong, Yiping Ma, Yifeng Mao, and Sebastian Angel. Addax: A fast, private, and accountable ad exchange infrastructure. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 825–848. USENIX Association, 2023.

A Errata

The DP analysis in an earlier version of this work was flawed in computing the (ϵ, δ)-DP parameters for a Laplace mechanism. In this new version we switched to the Gaussian mechanism with computations using Privacy Loss Random Variables for DP-composition. Our set of experiments has been updated and uses a new implementation written in Rust.

B Complexity of Algorithm 1

For each layer $i = 1, \ldots, \lambda$, we consider a call to $\textsc{Layered}(\cdot, \cdot, i, \cdot, \cdot)$. The average complexity is the size of the input dataset to this call, appended with $(L_i - 1)M_{\mathsf{buckets}} + M_{\mathsf{flush}}$ dummy reports. The size of the input dataset is the size of a bucket at the previous $(i-1)$-th layer which is not pruned (larger than t_{att}) and not a dummy bucket. We analyze the complexity in two different cases.

First, consider t_{att} too low, *i.e.*, $t \leq M$ so that the number of non-pruned buckets is exponential. In this case, the number of selected buckets at layer $i-1$ is upper-bounded by $(L_1 - 1)(L_2 - 1) \ldots (L_{i-1} - 1)$. By summing over all layers, we obtain total complexity of $O(\lambda C + (L_1 - 1) \ldots (L_\lambda - 1)M_{\mathsf{buckets}} + (L_1 - 1) \ldots (L_{\lambda-1} - 1)M_{\mathsf{flush}})$ (comparable to $O(\ell C + \bar{M}B)$). This is the complexity of the full histogram.

Next, consider t large enough ($t > M$) to prune effectively. We consider every possible bucket $\mathcal{B}_1, \ldots, \mathcal{B}_{L_1 L_2 \ldots L_{i-1}}$ at layer $i-1$. Denote by a_j the number of true reports in bucket $\mathcal{B}_j$ and by $X_j = \lfloor M + Z_j \rceil$, the number of added dummy reports, where Z_j follows the truncated Gauss distribution. Finally, let Y_j be the number of dummy buckets added in $\mathcal{B}_j$ at layer i. The complexity to treat $\mathcal{B}_j$ at layer i is bounded by $a_j + X_j + Y_j$ if $a_j + X_j \geq t$. Thus, the complexity to treat layer i is $\sum_j (a_j + X_j + Y_j) \cdot 1_{a_j + X_j \geq t}$. Because $\sum_j a_j = C$, this complexity is bounded by $C + \sum_j (X_j + Y_j) \cdot 1_{a_j + X_j \geq t}$. The coins for X_j and Y_j are independent. Since we want to compute the average complexity, we can directly average Y_j and get a complexity of $C + S$ with $S = \sum_j \mathbb{E}\left((X_j + (L_i - 1)M_{\mathsf{buckets}} + M_{\mathsf{flush}}) \cdot 1_{a_j \geq t - X_j}\right)$. We can show that all buckets such that $a_j < t - M$ have little influence on the sum: either there are a few with high a_j, or a_j is so low that X_j has too little chance to exceed $t - a_j$. The sum over buckets such that $a_j \geq t - M$ has a number of terms bounded by $\frac{C}{t-M}$ and is bounded by $\left(\bar{M} + (L_i - 1)M_{\mathsf{buckets}} + M_{\mathsf{flush}}\right)\frac{C}{t-M}$. We sum over all layers and obtain $O(\lambda C + (L_1 + \cdots + L_\lambda - \lambda)M_{\mathsf{buckets}} + \lambda M_{\mathsf{flush}}\frac{C}{t-M})$. In the extreme case, with $\lambda = O(\ell)$ and $L_{\mathsf{m}} = O(1)$, this is $O(\ell C + \bar{M}\frac{C}{t-M})$.

B.1 Expected Value of Noise

$$
\begin{aligned}
\mathbb{E}[X] &= \int_{-M-1/2}^{\infty} x\ \mathsf{PDF}_X(x)\ dx \\
&= \frac{1}{1-\mathsf{p}} \int_{-M-1/2}^{\infty} \frac{x}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}\ dx \\
&= \frac{1}{1-\mathsf{p}} \int_{-M-1/2}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \left[-e^{-\frac{x^2}{2\sigma^2}} \right]' dx \\
&= \frac{1}{1-\mathsf{p}} \frac{\sigma}{\sqrt{2\pi}} e^{-\frac{(-M-1/2)^2}{2\sigma^2}} \\
&= \frac{\sigma^2}{1-\mathsf{p}} \mathsf{PDF}_\sigma(-M-1/2) \\
&= \sigma^2 \mathsf{PDF}_X(-M-1/2)
\end{aligned}
\tag{3}
$$

C Analysis of Security and Privacy

C.1 Privacy of Oblivious Random Shuffling in Honest-but-Curious Model

We allow that a malicious participant U colludes with the Reporting Origin to learn the final histogram. In the worst case, we assume that U learns A' and B' produced by the shuffling protocol based on which the final histogram is computed.

Theorem 1. *Assume that all the participants follow the protocol and are non-colluding (honest but curious). For each participant of the protocol* Π_{RandShuf} *described in Figure 6, there exists an efficient simulator* Sim_U *such that the view of* U *in the protocol can be simulated from the final output* (A', B').

Proof. The view of $U = \mathcal{S}_1$ is that the received shares from each client D^1, the value π_{12}, π_{13}, R_{12} and R_{13}, the value A, and the value B':

$$(D^1, \pi_{12}, \pi_{13}, R_{12}, R_{13}, A, B')$$

$\mathcal{S}_1$ computes A' from (A, π_{13}, R_{13}). Then, its view is equivalent to

$$(D^1, \pi_{12}, \pi_{13}, R_{12}, R_{13}, A, A' + B')$$

where $A' + B' = \pi_{13}(\pi_{23}(\pi_{12}(D)))$. Since π_{13} is known, the view of $\mathcal{S}_1$ is equivalent to

$$(D^1, \pi_{12}, \pi_{13}, R_{12}, R_{13}, A, \pi_{23}(\pi_{12}(D)))$$

The first five terms: $D^1, \pi_{12}, \pi_{13}, R_{12}, R_{13}$ are independently sampled. The last term is a random permutation of D which is independent of the first five terms. The sixth term A is a function of D^2, R_{12}, R_{23} with an independent value R_{23}. Thus, the simulator for this view would independently sample the first six terms and would select an independent permutation of D which can be done from the output of the protocol.

The same procedure applies to $U = \mathcal{S}_2$ and $U = \mathcal{S}_3$ similarly. □

C.2 Privacy Loss Random Variable of Truncated Gaussian Mechanism

Given two neighbouring datasets D and D', we have 2λ regular buckets which differ by 1 and $2(\lambda-1)$ dummy buckets which differ by 1. We add $M+X$ reports to each of these buckets, where X represents the truncated Gaussian and M is the shift (Section 3.1), so $X = \mathcal{N}(\mu_N, \sigma_N^2)$ with truncation point $t_N = -M - 1/2$.

We sample from X the number of dummy reports to be added in the bucket with 1 less report. For each of these buckets, it means we compare samples from $X+1$ and X. The differential privacy boils down to the composition of λ $(X+1, X)$ pairs with "buckets" parameters and $(\lambda-1)$ $(X+1, X)$ pairs with "flush" parameters (Section 4.4). We have $(\sigma, M) = (\sigma_{\mathsf{buckets}}, M_{\mathsf{buckets}})$ composed 2λ times and $(\sigma, M) = (\sigma_{\mathsf{flush}}, M_{\mathsf{flush}})$ composed $2(\lambda-1)$ times.

Given a truncated $\mathcal{N}(\mu, \sigma^2)$ Gaussian distribution with truncation point t,

$$\mathsf{PDF}_{\mu,\sigma,t}(x) = \frac{\mathsf{PDF}_\sigma(x-\mu)}{1-\mathsf{CDF}_\sigma(t-\mu)} \text{ for } x \geq t$$

where PDF_σ is the PDF of $\mathcal{N}(0, \sigma^2)$. PDF of N is $\mathsf{PDF}_{0,\sigma,t}$ while PDF of $N+1$ is $\mathsf{PDF}_{1,\sigma,t+1}$.

We follow the methods from Gopi *et al.* [27]. We compute the *Privacy Loss Random Variable* (PRV) (denoted as Y) of X, deduce the PRV of the composition (denoted as Y_{all}) as the sum of the components of the composed distribution, use the convolution to compute the CDF of Y_{all}, and finally compute the privacy curve of our mechanism. That is, for each ϵ, we can compute a δ such that our mechanism is (ϵ, δ)-DP.

Gopi *et al.* compute the PRV of the Gaussian mechanism in [28, Proposition B.1]. We generalize their computation to a truncated Gaussian noise. By [27, Theorem 3.2],

$$\begin{aligned} Y &= \ln\left(\frac{\mathsf{PDF}_X(X)}{\mathsf{PDF}_{X+1}(X)}\right) = \ln\left(\frac{\mathsf{PDF}_\sigma(X)}{\mathsf{PDF}_\sigma(X-1)}\right) \\ &= \begin{cases} -\frac{2X-1}{2\sigma^2} & \text{if } X > t+1 \\ +\infty & \text{if } t < X < t+1 \end{cases} \end{aligned}$$

Y follows a truncated (on t_Y to $+\infty$) Gaussian distribution with an extra value at $+\infty$ with the following parameters (ignoring the infinite region):

$$\mathsf{p} = \mathsf{CDF}_\sigma(\mathsf{t}); \quad \mu_\mathsf{Y} = \frac{1}{2\sigma^2}; \quad \sigma_\mathsf{Y} = \frac{1}{\sigma}; \quad \mathsf{t_Y} = \mu_\mathsf{Y} - \frac{\mathsf{t}+1}{\sigma^2}$$

$$\begin{aligned} \mathsf{PDF}_Y(x) &= \frac{\mathsf{PDF}_{\sigma_Y}(x-\mu_Y)}{\mathsf{CDF}_{\sigma_Y}(t_Y-\mu_Y)} \text{ for } x < t_Y\,; \\ \Pr[Y=+\infty] &= \frac{\mathsf{CDF}_\sigma(t+1)-\mathsf{p}}{1-\mathsf{p}}\,. \end{aligned}$$

We compute the PRV of Y' of $(X, X+1)$ as we have to take the worst of Y and Y' for the privacy computation. By similar computation, we obtain a random variable Y' such that

$$\mathsf{PDF}_{Y'}(x) = \begin{cases} \frac{\mathsf{PDF}_{\sigma_Y}(y-\mu_Y)}{1-\mathsf{p}} & \text{if } x > t'_Y \\ 0 & \text{otherwise} \end{cases}$$

with the same $\mathsf{p}, \mu_{\mathsf{Y}}, \sigma_{\mathsf{Y}}$, but different $t'_Y = \mu_Y + \frac{t}{\sigma^2}$. There is no infinity case.

Experimentally, we verified that using Y (rather than Y') results in the bigger privacy loss. The PRV of the composition is $Y_{\text{all}} = \sum_i Y_i$, where each Y_i is the PRV of a component in the composition. Finally, [27, Theorem 3.3] gives the privacy curve as

$$\delta(\epsilon) = \int_{\epsilon}^{\infty} (1 - e^{\epsilon - x}) \mathsf{PDF}_{Y_{\mathsf{all}}}(x) + \Pr[Y_{\mathsf{all}} = \infty] \,.$$

We compute the δ for $\epsilon = 2$ this way and adjust the parameters to get δ low enough. To reach $\delta = 2^{-40}$, we observe that we roughly need $M = 7.5\sigma$ to have $\Pr[Y_{\mathsf{all}} = \infty] < 2^{-40}$.

C.3 Security Analysis for Malicious Clients

Theorem 2. *Let* N *be the number of malicious clients. The* L_1 *distance between the true histogram and the incorrect histogram output from* $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ *protocol without noise described in Figure 8 is bounded by* $2\mathsf{N}$.

Proof. We start with one malicious client. Let $\mathsf{rec}_{\mathsf{auth}}$ be the true report of a malicious client. Let a (resp. b) be the number of reports in the correct (resp. incorrect) bucket that $\mathsf{rec}_{\mathsf{auth}}$ belongs to. The consequence of the malicious behaviour is that true bucket will have $\mathsf{a}-1$ reports while incorrect bucket will have $\mathsf{b}+1$. The L_1 distance between true histogram and the is defined as the sum (over all buckets) of the absolute values of the difference between two counts in both histograms. In the case of one malicious client, the L_1 distance is bounded by 2. By triangle inequality, the L_1 distance induced by N clients is bounded by $2\mathsf{N}$. □

C.4 Privacy Analysis of $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$

(Informal) Security Analysis with Malicious Clients. In this section, we give the privacy bound against malicious clients. Note that the malicious behaviour of the client is to modify his reports in a way that it corresponds to a different bucket at the end of the protocol. This holds true because our protocol uses the length preserving secret sharing mechanism. A (malicious) client submits two shares of a report to two corresponding servers. Regardless of the authenticity of shares, the shares will belong to one bucket that an honest client could have submitted.

Informally, the $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ protocol protects against small subset of malicious clients. Since the shares of the attributes preserve the length of the original attribute size, a small subset of client can only secret-share a wrong report to be counted in another bucket. Since the aggregated results are already noisy, removing the report from the original bucket and increasing the count on another bucket only gives the affect of noise as long as only small set of clients are allowed to do that.

Formally, we prove the following result in Section C.3. Let N be the number of malicious clients. The L_1 distance between true histogram and the incorrect histogram output from $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ protocol is bounded by $2\mathsf{N}$.

Other than the above MPC protocol, more measures can be taken to bound user contributions, *e.g.*, browsers only upload one report per time period; reports must be aggregated then dumped per time period; reports contain random nonce under CCA secure encryption so they can't be duplicated.

(Informal) Privacy Analysis with Semi-Honest Servers. The cryptographic protocol $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ is built upon a generic honest-majority three-party computation protocol and a specific three-party oblivious permutation protocol. Overall, it is under honest-majority assumption, which means that the system does not tolerate any collusion. As long as there is no collusion between any pairs of servers, true counts are hidden from each server as well as the reporting origin. We formally prove the privacy of random shuffling in Section C.1.

In the LAYERED protocol, the semi-honest server additionally learns the number of reports in the dummy bucket before discarding it. This bucket includes dummy reports from upper layers and newly added dummy reports. We take it into account in the analysis of the LAYERED protocol.

We analyse the implication for differential privacy for LAYERED protocol in Section 4.4.

D Computing Sums

D.1 3-Party Oblivious Transfer

We depict the multiplication of least significant bit for the sum protocol is given in Figure 11.

D.2 Malicious Clients

Once the helper servers run the protocol in Figure 7 for each report, each server locally sums $\left(d_i^{(\mathsf{b})}\right)'$ modulo p' and reveals the results. Then both results are added and reduced modulo p' into $\left(-\frac{p}{2}, \frac{p}{2}\right)$. Observe that the total sum can be negative, which indicates that too many clients shared a negative report $d_i \in \left[-\frac{p}{2}, -1\right]$.

There are two intervals that a client can share a report maliciously: (1) $d \in \left[-\frac{p}{2}, -1\right]$ and (2) $d \in \left[\frac{p}{2}, p-1\right]$. We let $x = 2d \bmod p$. Then, we observe that

1. $x = 2d + p \in \left]0, p-2\right]$ is odd. In this case, we let $q = 1$.
2. $x = 2d + p \in [0, p-2]$ is odd again. In this case, we let $q = 0$.

In all cases, we have $d = \frac{x+(-1)^q}{2}$ with x odd. When the client wants to inject a negative report by following (1) or inject a positive out of range report by following (2), it sets x and q accordingly, computes the shares of x such that $q = \mathrm{lsb}(x^{(1)}) \oplus \mathrm{lsb}(x^{(2)})$, and prepares the shares of d for the helper servers as follows: $d^{(1)} = \frac{x^{(1)}}{2} \bmod p$ and $d^{(2)} = \frac{x^{(2)}}{2} \bmod p$.

When the helper servers receive $d^{(\mathsf{b})}$, they both compute $x^{(\mathsf{b})}$ to share x, mutually compute q, and $x' = x^{(1)} + x^{(2)} - qp$. However, $x = x^{(1)} + x^{(2)} - \mathbf{carry} \times p$, $\mathbf{carry} = \mathrm{flip}(q)$ because $x \bmod 2 = \mathrm{lsb}(x^{(1)}) \oplus \mathrm{lsb}(x^{(2)}) \oplus \mathbf{carry}$. We represent this flip as $\mathbf{carry} = (q + (-1)^q)$. Then, $x = x^{(1)} + x^{(2)} - \mathbf{carry} \times p = x = x^{(1)} + x^{(2)} - (q + (-1)^q) \times p$ which is different than x' servers obtained: $x' = x + (-1)^q p = 2d$.

When the servers compute $\left(d^{(\mathsf{b})}\right)' = \frac{x^{(\mathsf{b})} + q^{(\mathsf{b})} p}{2} \bmod p'$ in the last step of arithmetic conversion in Figure 7, this yields a sum

$$\left(d^{(1)}\right)' + \left(d^{(2)}\right)' = \frac{x'}{2} \bmod p' = d \bmod p'.$$

The adversary succeeds to add $d \in \left[-\frac{p}{2}, p\right]$ to the total sum.

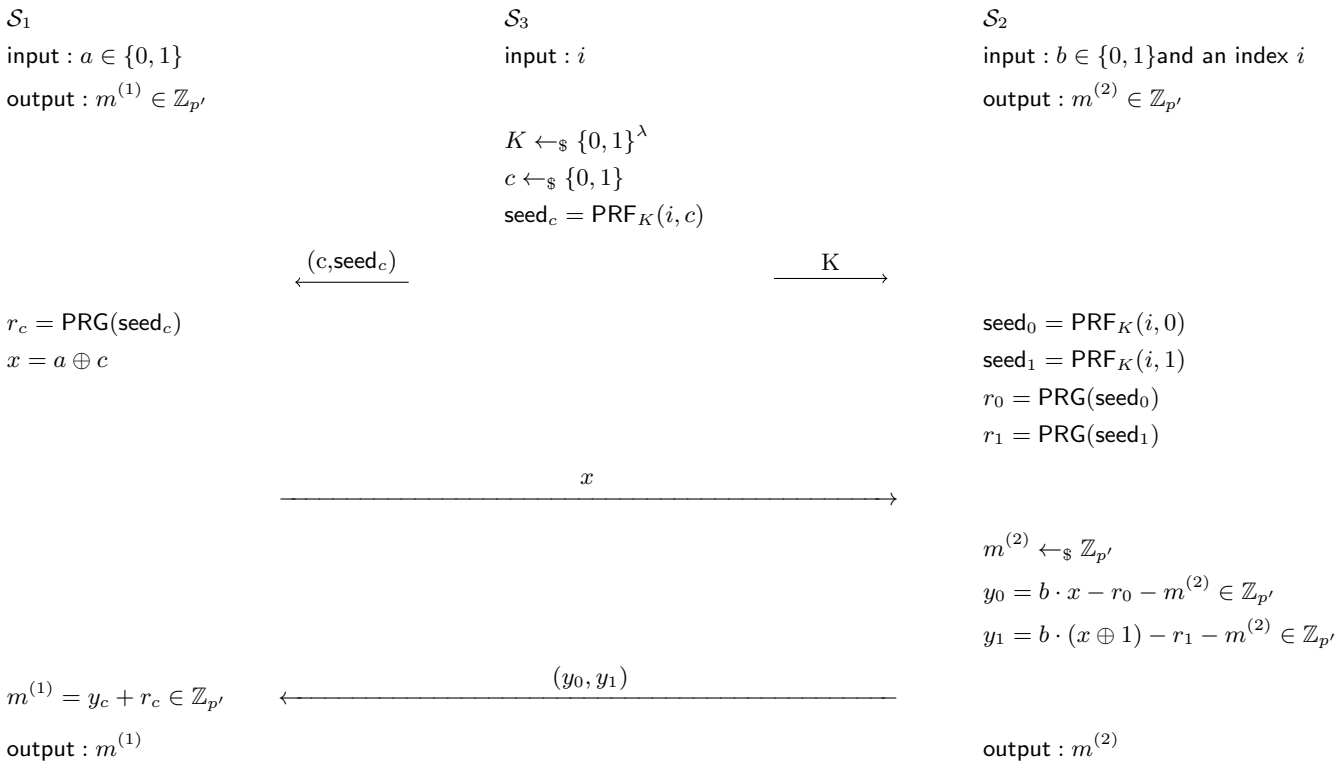

Figure 11: 3-party Oblivious Transfer to compute the arithmetic shares of multiplication of two bits a and b (adapted from ABY[3]). $\mathcal{S}_3$ inputs only the index number i for the i-th iteration. This protocol is run with the same K for each report in parallel. The communication complexity is $\lambda+2$ bits (λ bits for seed_c and 2 bits for c and x) and 2 group elements in $\mathbb{Z}_{p'}$, y_0 and y_1.

Example. Let $p = 201$ and the range of authentic reports be $d \in \{0, \ldots, 100\}$. A malicious client can share a report $d \in \{101, \ldots, 200\}$ with a q that will be computed by the servers from the least significant bits of shares $x^{(1)}$ and $x^{(2)}$. If the client can make $q = 0$, it will add a report which is outside of the range. If the client can make $q = 1$, it will add a negative value to the sum bounded by $\frac{-p}{2}$. So, the client can decide to report $d = 190$ (which is outside range) which will end up with adding -11 to the sum if x is shared with $q = 1$, or it will add 190 to the sum if x is shared with $q = 0$.

A malicious client can decide to cheat by sharing a report $d \in \{-100, \ldots, -1\}$. Let $d = -11$. Notice that -11 and 190 are equal to $d = 190$ in modulo p, but they differ depending on how they are shared in modulo p'. The client "anticipates" the shares of the servers $x^{(\mathsf{b})}$ by inverting what the servers do: *divide d by the 2 modulo p.*

The client computes an x such that $d = \frac{x+(-1)^q p}{2}$. For $d = -11$, $x = 179$ with $q = 1$ following from $x = 2d + p$. Now, the client "prepares" the shares of x such that $\mathrm{lsb}(x^{(1)}) \oplus \mathrm{lsb}(x^{(2)}) = q$.

We let $x^{(1)} = 70$ which makes $x^{(2)} = 109$. Notice that $q = 1$, we need $x = x^{(1)} + x^{(2)}$, the shares

of x can't be bigger than x itself. Then, the adversary shares d modulo p as

$$d^{(1)} = \frac{x^{(1)}}{2} = 35 \bmod p\,, d^{(2)} = \frac{x^{(2)} + p}{2} = 155 \bmod p\,.$$

The adversary succeeds to add $d = -11$ to the total sum which would be interpreted as $p' - 11$ if the final sum were *not* represented in $\left[-\frac{p}{2}, \frac{p}{2}\right)$.

Thus, we require that $p' \geq 2pC + M_{\mathsf{sum}}$ to make sure that the absolute value of the noisy sum does not exceed $p'/2$ so that its representation in $\left[-\frac{p'}{2}, \frac{p'}{2}\right]$ is the correct sum. Then, the difference between the sum with honest clients and the sum with malicious clients reports is in $\left[-\frac{Np}{2}, Np\right]$, where N is the number of malicious clients.

E Existing Proposals

Even though there are many different proposals to solve secure aggregate problems, in this section, we focus on two specific proposals: Prio [14] and Distributed Point Functions (DPF) [7].

E.1 Prio

Prio [14] is the first existing protocol to solve privacy preserving aggregate systems which is robust against malicious clients. It does not rely on any general purpose MPC. The protocol can be used for many different aggregates such as histograms, sum, average, heavy-hitter, and others with different techniques and, as a result, with different costs.

Prio uses two-party computations in order to compute the aggregates. Each client secret shares (defined in $\mathbb{Z}_p$ for a prime p) their data to the servers. In order to provide robustness against malicious clients, Prio integrates a special range proof called SNIP and characterized by a Valid predicate. Each client gives each server a proof that the shared data satisfies this predicate. A data point x (shared by a client) is supposed to satisfy the Valid predicate in order to prove the validity of the data point. The predicate is defined by an arithmetic circuit with N multiplications. Even though constructing such proofs are efficient enough, the size of the proofs are $O(N)$ elements in $\mathbb{Z}_p$. This implies a very expensive communication complexity from clients to servers. When the servers receive the proofs, they run the Valid predicate which only requires 1 MPC multiplication per client no matter how large N is.

Prio encodes data x before sharing and this encoding depends on which aggregate function to compute and what type of proof is required. For example, to prove that x is made of ℓ bits, the client first encodes x as $\mathsf{Encode}(x) = (x, \beta_0, \ldots, \beta_{\ell-1})$ β_i represents bits. Then, it generates a proof that $x = \sum_i \beta_i 2^i$ and that every bit β_i is a root of a polynomial $P(z) = z^2 - z$. Thus, what is shared and proved is $\mathsf{Encode}(x)$.

If Prio is used to compute the histograms (or frequency counts as the paper names it), then the encoding becomes a lot larger. The encoding is defined as $\mathsf{Encode}(x) = (\beta_0, \ldots, \beta_{B-1})$ where B is the number of the buckets ($B = 2^\ell$ for full histograms) and $\beta_x = 1$ while $\beta_i = 0$ for $i \in \{0, \ldots, B-1\} \setminus \{x\}$. Valid predicate requires all β_i to be 0 or 1 as well as $\sum_i \beta_i = 1$. As it can be observed, such a method is inefficient for histogram computations. Therefore, we also omit its performance analysis in our comparisons.

Finally, Prio, as it is proposed, does not provide any differential privacy guarantees. However, as shown in some use-cases, it may be possible to add such guarantee under certain conditions [2].

For now, we are not aware of any effort put in that direction. Instead, another proposal to solve specifically the heavy-hitter problem with differential privacy guarantees is proposed. This new proposal specifically aims to reduce the client-side communication complexity of Prio for heavy-hitter problem, as well as introducing additional differential privacy guarantees. We will explain this new primitive next.

E.2 Distributed Point Functions

Recently, Google proposed an Attribute Reporting API with Aggregate Reports scheme, which strongly aligns with this problem [38]. A potential solution mentioned in their proposal relies on Distributed Point Functions (DPF): a two-party secure computation protocol [7, 23]. More precisely, DPF consists of two protocols: DPF.Gen and DPF.Eval. We pause here to explain the basic idea of DPF. Theoretically, the keys can be represented with a large vector of size of the key space. For an ℓ-bit key k, the key can be represented as a one-hot encoded vector of size 2^ℓ, with the k-th position set to 1 and other positions to 0. Then, this vector can be secret shared and sent to two servers to compute the aggregates. However, this naive approach requires too much communication. The beautiful idea DPF introduces is to generate the secret shares of this vector in a compact form and let the servers expand the keys to the full vectors by executing a series of cryptographic operations. The structure of this expansion is a tree structure, *i.e.*, the expansion happens level by level. Essentially, DPF takes these vectors and treats them as functions, which are equivalent when the representation is a point function.

At the beginning of the data collection clients generate their secret shared reports by DPF.Gen. Then, two servers jointly execute DPF.Eval to generate noisy aggregates. Data users (*e.g.*, advertisers) make queries to two servers and receive differentially private results. The aggregate queries DPF allows are histogram and sum on reported keys and values.

The most recent DPF construction is introduced as a solution to the *private heavy hitters problem* [7]. Particularly, Boneh *et al.* [7] describes three main protocols in their paper.

The first protocol is to build a private subset histogram from collected reports for a given set of keys. The set of keys may or may not be known to the servers. It requires $O(CB)$ DPF.Eval calls, where C is the number of reports and B is the set of keys to build the histogram on (without differential privacy).[9]

The second protocol is to find the most popular keys, which appears with a threshold t (without differential privacy); this is called the t-heavy hitters problem. Boneh *et al.* defines a new DPF called *incremental DPF* (iDPF) to solve this problem more efficiently than with standard DPF. The complexity of the proposed protocol is $O(\ell C^2/t)$ DPF.Eval calls, where ℓ is the (fixed) size of the keys collected from clients. The third protocol is simply to use the t-heavy hitters protocol with threshold $t = 1$. Then, the complexity becomes $O(\ell C^2)$ DPF.Eval calls.

These protocols can be made differentially private by applying the noise addition process at certain steps. The differential privacy parameters proposed in [7] use

$$\epsilon' = \epsilon\sqrt{2q \ln \frac{1}{\delta'}} + \epsilon q e^{\epsilon - 1},$$

with $q = \ell C/t$. They provide example parameters for an (ϵ', δ')-DP protocol, with $\ell = 256$, $\epsilon = 0.001$, $t = C/100$, and $\delta' = 2^{-40}$, resulting in $\epsilon' = 1.22$. However, the impact on the complexity and the accuracy is not analysed.

[9] Note that the complexity of DPF.Eval is exponential in the size of the keys.

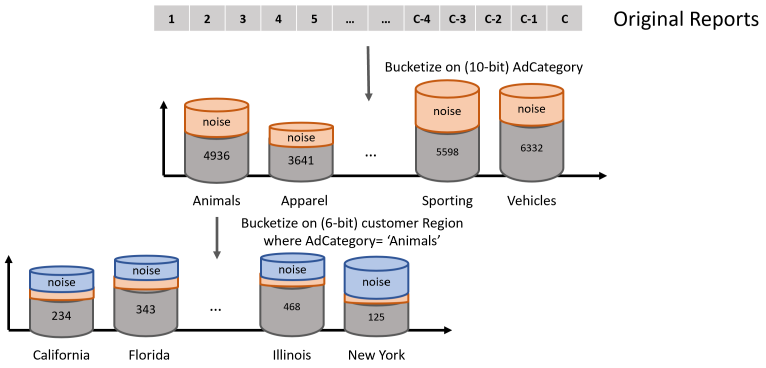

Figure 12: An example demonstrating the flexible functionality of our attribute aggregation protocol. When the computations run with Precio, the outputs will be differentially private histogram counts, *i.e.*, the size of each bucket plus a noise term.

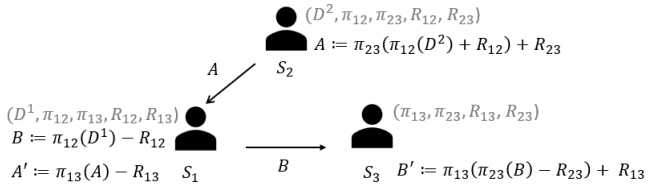

Figure 13: Illustration of Protocol Π_{RandShuf}. Gray tuples indicates the inputs known to each server.

F Description of Figures

We include some figures in this section to support the descriptions of the system design and protocols. The Figure 12 is a visualization of the layered protocol. The Figure 13 shows the data flows between three helper servers during oblivious shuffling. The Figure 14 depicts the main subroutines of Precio. Specifically, it shows how the helper servers add dummy reports to a secretly shared dataset and then randomly shuffle it.

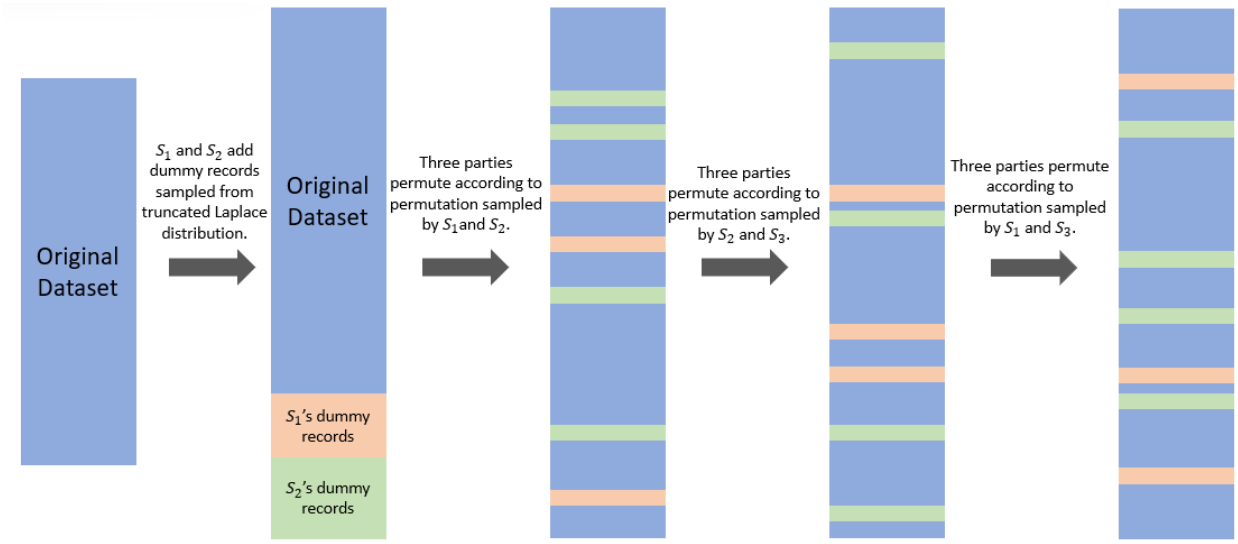

Figure 14: The workflow of sub-routines in Precio starting with dummy report addition and oblivious shuffling. It omits the secret sharing and the generation of output buckets. The dummy reports are colored to show the shuffling, due to the XOR of random masks, they will not be traced. This flow will run with three servers, which carry the shares of the original reports.