Precio: Private Aggregate Measurement via Oblivious Shuffling

F. Betül Durak
Microsoft
United States

Chenkai Weng
North Western University
United States

Erik Anderson
Microsoft
United States

Kim Laine
Microsoft
United States

Melissa Chase
Microsoft
United States

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 (time and 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 [16, 32], private networks [10, 20, 31], 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 [15], 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 [21] 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, at the cost of somewhat stronger trust assumptions.

Web Privacy and Ad Conversion Measurement. People's activities are constantly tracked across websites to gather insights about online marketing campaigns. This is typically achieved using third-party cookies: a website A can drop a cookie linked to website F, which, when loaded on other sites, can subsequently track the user's activities [5]. Due to privacy issues, some browser vendors are strictly limiting third-party cookies, with Safari blocking them and Chrome planning to phase them out with the *Privacy Sandbox* initiative [25]. Yet, information gathering is essential for advertisers who ultimately fund many "free" valuable web services consumers rely on.

Today, consumer (web browser) data is collected with no guarantees that the data is not used for other purposes. The good news is that many analytics scenarios only require simple aggregates, such as computing how often an ad placement results in completing a related purchase. If there was a way to only ever reveal such aggregate results, it would mitigate privacy risks while retaining necessary measurement capabilities for advertisers. In this paper we suggest a privacy-focused aggregate protocol using secure multiparty 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.

  Discussion: Of course this means that one cannot ensure that each client submits accurate inputs. Instead, the goal is to limit the amount of influence any one (or few) clients can have on the final result. That way even if a few malicious clients manage to evade detection, they cannot affect the results too much. In particular, we cannot tolerate a scheme in which a single malicious client could completely obscure the aggregate result.

- **Helper servers**, that together perform the aggregation. In our work we will assume 3 semi-honest non-colluding helper servers.

  Discussion: We assume these will be run by entities with strong reputations – either large tech companies or non-profit organizations. These entities may also be audited to ensure correct code and processes are used.

- A **Reporting Origin (RO)**, that collects reports from the clients, encrypted under the helper servers' keys. It sends them to the helper servers, and chooses which aggregates to compute, and receives the final results. This could be an ad network measuring the effectiveness of an ad campaign. A malicious RO should not learn anything about individual clients' reports.

  Discussion: To be adopted in the ad measurement context, and avoid concerns of unfairness to smaller players, such a system would have to be available to a wide range of ad networks, even those without a strong reputation for privacy. This means it is essential that we not place any trust for privacy in the RO. On

the other hand, accuracy depends on the RO honestly submitting the reports; in our model the RO is the one who wants the results, so this is not a limitation. The RO may also perform (imperfect) client vetting to limit the number of malicious reports.

Our protocol provides privacy against semi-honest helpers colluding with an RO, as long as a majority of the helpers remain honest. In our security analysis, this means that we provide privacy even when the reports are maliciously generated, as the RO's role is simply to choose and distribute the reports to helper servers. We leave the stronger guarantees of privacy/correctness in the presence of malicious helper servers for future work.

Data Reports. Reports from clients are bit strings representing predefined categorical or numerical *attributes*. Each attribute takes up to a specified number of bits in the report to denote its *attribute value*. Categorical attributes may indicate client age, gender, or device info, while a numerical attribute may indicate a dollar value of purchases after seeing an ad. Categorical attributes are used to form histograms and numerical attributes to compute sums. Our goal is to compute complex histograms and histogram-sums over these reports, *e.g.*, "How many conversions were there for ad campaign A for mobile phone users in North America?" or "what was the total dollar value of all conversions from users in Europe age 65+?"

Repeated Partitioning and Filtering. In a non-private ad conversion measurement system [36], 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* (DPF) [8, 9, 23], *incremental* DPF (iDPF) [7], and Prio [1, 15]. iDPF is very similar to DPF; it adds support for longer reports and differential privacy.

In the iDPF 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 is again 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 time and communication complexity linear in the number of reports. The RO learns only differentially private aggregates and no per-client information. The high-level design of Precio is depicted in Figure 1.

Unlike 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. Prio [15] enables sum computation on private values but is vulnerable to malicious clients providing unrealistic inputs, requiring costly range proofs as a mitigation. Prio+ [1] mitigates this using limited size domains and Boolean secret sharing, but requires an expensive share conversion protocol.

Precio, however, allows clients to secret share in a small cyclic group, then converts these to a larger group for the sum. This process (Section 3.3) uses Quotient Transfer [33] and Oblivious Transfer (OT); it avoids Prio's costly range proofs and is more efficient than Prio+'s share conversion.

Privacy and 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 only slightly more than the impact it can have by simply choosing a different value in the range of the numerical attribute.

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. We prove privacy against semi-honest non-colluding helper servers. The RO's role in our protocol is simply to choose and distribute the reports to helper servers. To capture the case of a malicious RO, our privacy guarantees must hold even when the reports are maliciously generated.

We note that Precio focuses on obtaining differential privacy for histograms and sums. The related works (DPF, iDPF, Prio) do not natively provide DP, but can be augmented to provide it. Instead, they provide MPC-type guarantees which say that only aggregates are revealed.

The privacy and security guarantees of Precio are summarized in Table 1, along with a comparison to related works.

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.[1] 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 is in Appendix B.

As the report size ℓ grows, the number of possible buckets $B = 2^\ell$ grows exponentially. We resolve this problem by introducing a layering technique, breaking the report into smaller attributes and

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

exploring those attributes one after another. By pruning away entire reports when some of their attributes fall into uninteresting (*e.g.*, small) buckets, we can create histograms only for buckets with more than t reports with complexity $O\left(\ell C + \ell M \frac{C}{t-M}\right)$.

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.

The complexity of Precio is summarized in Table 2, along with a comparison to related works.

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 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. To begin with, suppose we only want to compute a single histogram. We begin with the proposal of Mazloom and Gordon [35], 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 this determines how many *dummy reports* with that particular attribute value are added. Each server obtains secret shares of these additional reports.

Next, the servers run a 2PC protocol to shuffle the reports, hiding which 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, we can use linearity of the secret sharing to add the numerical values for each category.

Optimizing Building Blocks. The Mazloom and Gordon protocol contains three costly steps: 1) The two servers use costly generic 2PC to jointly generate dummy reports; 2) they shuffle the shares, again using a generic 2PC; 3) a costly 2PC is used to add numerical values in a way that prevents malicious clients from submitting arbitrarily large inputs.

We optimize each step. In Mazloom and Gordon's proposal, the sampling and generation of the dummy reports happens within a 2PC. In Precio, two helpers independently sample noise and generate dummy reports, removing the need for generic 2PC at the cost of doubling the noise. Moreover, to avoid revealing how many dummy reports are added, they must pad the number of dummies; in more detail, they use a noise distribution, which is truncated at some maximum value and pad the number of dummies for each attribute up to this maximum. In Precio, we take advantage of the fact that in our setting the adversary (RO or semi-honest helper) already knows the total number of real (non-dummy) reports, so we can use a variant of DP which takes this into account (see [42, pp. 12] for discussion). This means, in particular, that we can afford to reveal the total number of dummy reports added, although of course not how many are added for each attribute, and omit the additional padding that Mazloom and Gordon require.

Next, we optimize the shuffle protocol. There are two existing approaches to shuffling data according to a hidden random permutation. The first is that taken by Mazloom and Gordon: use a 2PC to evaluate a sorting network, hiding the bits used to control each swap gate, as in [30, 38]. This allows for efficient symmetric key operations, but, since sorting networks require $n \log n$ gates to sort n items, this implies adding an overhead factor of $\log C$ for us.

The other approach is to have one party encrypt the data with a public key rerandomizable encryption and have another party shuffle and rerandomize the ciphertexts.[2] In our context, the helpers would also need to rerandomize the shares, which requires homomorphic encryption, and moreover we need the shares to be compatible with the rest of our protocol. In particular, we need to be able to separately reveal the shares of each attribute. To do this with known PKE schemes would require separately encrypting each attribute in the user's report; as public key ciphertexts are at least 500 bits, and our attributes on the order of 1–10 bits, this would be an order of magnitude increase. Using FHE schemes that typically natively provide message packing capabilities is an option, but comes with a significant overhead.

Instead, we introduce an additional helper party and use a 3-party permutation protocol corresponding roughly to two rounds of the protocol of [37]. 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.

Finally, we combine the Quotient Transfer idea from [33] with a three-party OT to get 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.

[2] Repeat twice to get a permutation that is hidden from both parties.

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 mean 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 reports 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 reports (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 [28].

1.4 Related Work

There exist many prior work in privacy-preserving aggregation, based on general purpose MPC [24, 45], Distributed Point Functions (DPF) [7–9, 23], or a variety of techniques [1, 11, 15, 46, 47].

iDPF and Prio. The closest in terms of functionality are the DPF-based solutions: regular DPF [8, 9, 23] and incremental DPF (iDPF) [7]. Both iDPF and Prio [15] are currently being proposed for standardization by IETF [22, 41, 44].

The DPF solution has a very high time complexity, which iDPF improves, still resulting in complexity quadratic in the number of clients. The communication complexity is linear in the number of clients. We compare iDPF 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, Prio, natively, does not provide DP [15] and does not support layered histograms. To imitate layered histograms, one would need to encode the report into a massive vector of the size of the entire attribute space, which is completely impractical for almost all of our experiments. Due to these differences, we do not compare performance with Prio in Section 5.

It is worth noting that Precio differs from the DPF-based protocols and Prio on the trust assumption. All three, Prio, iDPF, and Precio provide robustness of computations only if all the servers are honest. However, Precio does not protect client privacy against malicious server, whereas Prio and iDPF do, if at least one of the servers is honest.

For malicious clients, Prio defines robustness as follows: "when all servers are honest, a set of malicious clients cannot influence the final aggregate, beyond their ability to choose arbitrary *valid* inputs." iDPF can provide robustness against malicious clients, but this requires an expensive sketching process. Precio is robust against malicious clients, namely, a malicious client can have an effect equivalent to contributing two maliciously chosen inputs instead of one. For more details, see Section 4.3.

The privacy and security guarantees of Prio, iDPF, and the Mazloom and Gordon protocol are presented alongside Precio in Table 1. The time and communication complexities are in Table 2.

Mixnets. Mixnet-based 2-server protocols with verifiable shuffling would incur prohibitive overhead for using public-key operations and zero-knowledge proofs [13, 40]. A private messaging scheme called Vuvuzela [43] employs mixnets and differential privacy, and can potentially be used for simple histogram queries. However, its differential privacy guarantees are undefined for secure aggregation.

Shuffling. Several works [6, 14] 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 makes these protocols a bad fit for our application, since the (potentially malicious) RO will decide which reports to submit for aggregation and the helper servers have no way to verify that the reports came from legitimate clients.

IPA. Another proposal for distributed aggregation in the ads privacy space is the IPA protocol[11]. The proposal has the difference that the aggregation follows a step of attributing conversions (*e.g.*, purchase) to impressions (*e.g.*, display). This inherently makes the problem more challenging and solutions much more complex.

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 $b \in \{1, 2, 3\}$ to denote a server index. Unless explicitly stated otherwise, all indices start from 1.

Protocol	Prio [15]	iDPF [7]	M&G [35]	Precio
client privacy (DP) w/ non-colluding semi-honest servers	possible	possible	yes	yes
client privacy (DP) w/ one malicious server	possible	possible	no (ext.)	no
client privacy (DP) w/ malicious clients (hist.)	possible	possible	yes	yes
client privacy (DP) w/ malicious clients (sum)	possible	possible	yes	yes
robustness w/ one malicious server (hist. and sum)	no	no	no (ext.)	no
robustness w/ malicious clients (hist.)	yes (range proofs)	yes (sketching)	yes	yes (no extra cost)
robustness w/ malicious clients (sum)	yes (range proofs)	yes (sketching)	yes	yes* (quotient transfer)
# of servers	2	2	2	3

Table 1: Threat model and security guarantee comparison for different histogram and sum protocols. "Possible" privacy means that the protocol reveals only the aggregate being accumulated but can be strengthened by DP in a straightforward way with a complexity overhead. "ext." indicates that the protocol can be extended to provide this property, but the extension is non-trivial. The asterisk (*) indicates that a malicious client can have a bounded (small) impact on the result. Mazloom and Gordon do not explicitly discuss malicious clients, but their protocol seems to protect against them, assuming the inputs are shared in a way that limits the maximum value.

Protocol	Prio [15] (no DP)	iDPF [7] (no DP, no sketching)	Precio
server time complexity (histogram)	$O(\ell \log(\ell) C)$ multiplications	$O(\ell C^2)$-PRG calls	$O(C + B\bar{M})$
server time complexity (heavy-hitter)	N/A	$O(\ell C^2/t)$-PRG calls	$O\left(\ell C + \ell M \frac{C}{t-\bar{M}}\right)$
server-to-server communication (histogram)	$O(C \log_2(p))$	$O(\ell C \log_2(p))$	$O(\ell(C + B\bar{M}))$
client time complexity	$O(\ell \log(\ell))$ multiplications	$O(\ell)$-PRG calls	$O(1)$
client-to-server communication	$\ell \log_2(p)$	$O(\kappa \ell)$	ℓ

Table 2: Complexity comparisons of private histogram computation. ℓ is the report length; t is the pruning threshold ($t = 1$ for full histogram); κ is the security parameter; $\log_2(p) \geq \kappa$ is the size of the finite field elements; 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 Precio. DPF **row does not account the DP protection (our scheme without DP would mean $\bar{M} = 0$). We give real time performance results for Precio and** iDPF **for different ℓ in Table 4. Note that, as described above, there are differences in the expressive power of each of the schemes: Prio reports encode attributes as one-hot vectors and does not allow for histograms/sums over combinations of attributes; iDPF only allows for histograms/sums over prefixes of the report.**

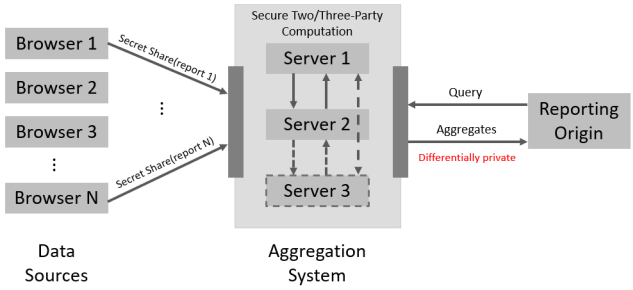

Figure 1: System Architecture.

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_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 in the range $[0, 100]$.

$d_1^{(1)}$ = 10 011 78
$d_2^{(1)}$ = 00 011 45
$d_3^{(1)}$ = 01 111 100
$d_4^{(1)}$ = 11 110 52
$d_5^{(1)}$ = 01 001 11

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

$d_1^{(2)}$ = 10 001 148
$d_2^{(2)}$ = 01 010 188
$d_3^{(2)}$ = 01 111 120
$d_4^{(2)}$ = 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 $\mathcal{S}_2$.

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

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 by an integer in $[0, 100]$). 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.

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 2 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 [18, 19] is well studied technique to protects the privacy of individual rows in a database. 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.

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',

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

This variant of DP is well-known and suitable for situations where the adversary knows the exact size of the database before noise is added [42, pp. 12]. This is natural, because the adversary collects the reports anyway and therefore knows the exact number of real reports.

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 4.5, we explain how we set the distribution parameters by using the techniques to compute privacy random variables [27, 28]. In our protocol, noise is added by both servers independently.

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. For our application we need the secret sharing scheme to be compatible with the way that we encode our attributes; in particular we need a bitwise encoding for categorical attributes and a sharing in an appropriate sized field for numerical attributes.

Multiple oblivious shuffling protocols have been presented in prior work. Chase *et al.* [12] 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.* [37] 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 [37] in Section 3.2.

3 SUBROUTINES

3.1 Differential Privacy with Constraints For Histograms

To ensure differential privacy in histograms, we introduce noise to each bucket's counts. While the standard Gaussian mechanism

is our preferred choice, it poses a challenge: positive noise indicates dummy report additions but negative implies report removals we cannot do. Medina *et al.* [39] suggest a solution for such non-negative constraints, but we prefer to stick with the Gaussian mechanism for its optimal privacy-utility balance and its compatibility with numerical composition [28].

Rounded Shifted Truncated Gaussian Mechanism. 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.[3]

The protocol for noise generation is shown in Figure 4.

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 $\mathcal{S}_1$.[4] This means that a total of $n^{(1)} = 5$ new 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:

$$\mathcal{B}_1 = \{d^{(\mathsf{b})}_1, d^{(\mathsf{b})}_3, d^{(\mathsf{b})}_4, d^{(\mathsf{b})}_6, d^{(\mathsf{b})}_7, d^{(\mathsf{b})}_{11}\}, \quad \mathcal{B}_2 = \{d^{(\mathsf{b})}_2, d^{(\mathsf{b})}_8\},$$

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 $n^{(\mathsf{b})}_j$. 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 $\mathcal{N}^{(\mathsf{b})} = (n^{(\mathsf{b})}_j + M_{\mathsf{buckets}})_{j \in G_\mathsf{m} \setminus \perp} \,||\, (n^{(\mathsf{b})}_\perp + M_{\mathsf{flush}})$.

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 $i \in [n^{(\mathsf{b})}_j + M]$, set $d^{(\mathsf{b})}_{i,j}[\mathsf{m}] = j$ and $d^{(\mathsf{b})}_{i,j}[v] \leftarrow \perp (\in G_v)$ for $v \in [\mu] \setminus \{\mathsf{m}\}$ which form one dummy report $d^{(\mathsf{b})}_{i,j}$ (in the case of a numerical attribute for summing, $d^{(\mathsf{b})}_{i,j}[v] \leftarrow 0$ $(\in G_v)$ for v the attribute to be summed). $\mathcal{S}_\mathsf{b}$ forms all of

$$n^{(\mathsf{b})} = \Big(\sum_{j \in G_\mathsf{m} \setminus \perp} n^{(\mathsf{b})}_j + M_{\mathsf{buckets}} \Big) + (n^{(\mathsf{b})}_\perp + M_{\mathsf{flush}})$$

dummy $d^{(\mathsf{b})}_{i,j}$ reports as $D^{(\mathsf{b})}_{\mathsf{dum}}$. Each server b shuffles $D^{(\mathsf{b})}_{\mathsf{dum}}$.

Appending shares to dummy reports. $\mathcal{S}_1$ and $\mathcal{S}_2$ share the numbers $n^{(1)}$ and $n^{(2)}$ with each other. Set $D^{(1)}_{\mathsf{priv}}$ and $D^{(2)}_{\mathsf{priv}}$ as follows: $(D^{(1)}_{\mathsf{priv}})_i = D^{(1)}_i$ for $i < C$; $(D^{(1)}_{\mathsf{priv}})_i = (D^{(1)}_{\mathsf{dum}})_{i-C}$ for $C < i \leq C + n^{(1)}$; and $(D^{(1)}_{\mathsf{priv}})_i = 0$ for $C + n^{(1)} < i \leq C + n^{(1)} + n^{(2)}$. $\mathcal{S}_2$ 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 $D^{(\mathsf{b})}_{\mathsf{priv}}$.

Figure 4: The protocol of DP noise generation.

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

[4] 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	$d_1^{(2)}$ = 10 001 148
$d_2^{(1)}$ = 00 011 45	$d_2^{(2)}$ = 01 010 188
$d_3^{(1)}$ = 01 111 100	$d_3^{(2)}$ = 01 111 120
$d_4^{(1)}$ = 11 110 52	$d_4^{(2)}$ = 11 100 12
$d_5^{(1)}$ = 01 001 11	$d_5^{(2)}$ = 11 011 42
$d_6^{(1)}$ = 00 111 0	$d_6^{(2)}$ = 00 000 0
$d_7^{(1)}$ = 00 111 0	$d_7^{(2)}$ = 00 000 0
$d_8^{(1)}$ = 01 111 0	$d_8^{(2)}$ = 00 000 0
$d_9^{(1)}$ = 11 111 0	$d_9^{(2)}$ = 00 000 0
$d_{10}^{(1)}$ = 11 111 0	$d_{10}^{(2)}$ = 00 000 0
$d_{11}^{(1)}$ = 00 000 0	$d_{11}^{(2)}$ = 00 111 0
$d_{12}^{(1)}$ = 00 000 0	$d_{12}^{(2)}$ = 11 111 0
(a) $D^{(1)}_{\mathbf{priv}}$	**(b)** $D^{(2)}_{\mathbf{priv}}$

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

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

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

We modify the Mohassel *et al.* [37] protocol into an efficient honest-majority 3-party oblivious random shuffling protocol with linear complexity (Figure 6). 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 sends only one message, resulting in a constant number of rounds of communication. We do not need commitments or interaction because of the third server, because when one of the servers is corrupted the randomness from the (third) honest server is enough to create a uniform distribution for the permutation.

The formal privacy of the protocol in the semi-honest server model is proven 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.

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

Initialize. For $(\mathrm{b}_1, \mathrm{b}_2) \in \{(1,2), (2,3), (1,3)\}$, 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.

3.3 Secure Modulo Conversion for Sum

When computing sums over numerical attributes, the main challenge is preventing out-of-range inputs being secret shared. To address this with minimal client overhead, clients can share values in a small domain that servers can "lift" to a larger field for addition. The secret inputs can be either (1) Boolean secret shares with fixed length or (2) arithmetic secret shares in a small field.

Boolean-to-Arithmetic (B2A) Conversion. Prio+ [1] employs B2A conversion, with clients secret sharing values as $d_i = d_i^{(1)} \oplus d_i^{(2)}$. The shares are capped at ℓ bits, inherently restricting client inputs. To calculate sums, a lift to high-precision arithmetic is essential due to the high costs of sum protocols over Boolean shares.

The most efficient B2A technique in a 2PC semi-honest setting is due to ABY [17]. To convert ℓ-bit Boolean shares, Prio+ [1, pp. 14–18] uses ℓ instances of OT, each communicating $\frac{\ell+1}{2}$ bits on average. Hence, the total communication complexity is $O(\ell^2)$. The OT approach requires pre-computations in an offline phase; alternatively, one could use bit-wise multiplication, 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. Let p be an odd integer such that the client's value d is between 0 and $\frac{p-1}{2}$. Each client secret shares d as $d = d^{(1)} + d^{(2)} \bmod p$. In this case, the modulus p limits the clients' inputs to a desired range. To compute the sum, the shares need to first be lifted to a larger domain, which we show to be efficient using Quotient Transfer [33, 34].

Suppose we want to convert shares of d modulo a small odd integer p into shares modulo a large odd integer p'. First, suppose the client submits shares of $d = d^{(1)} + d^{(2)} \bmod p$, where $d < p/2$. The servers can deduce shares $x^{(\mathrm{b})} = 2d^{(\mathrm{b})} \bmod p$, such that $2d = x^{(1)} + x^{(2)} \bmod p$. This implies

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

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

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

Therefore, if we can compute secret shares mod p' of q using the lsb of $x^{(\mathrm{b})}$, then lifting the shares of d to p' becomes straightforward. This is known as Quotient Transfer [33].

Let $q^{(1)}$ and $q^{(2)}$ be the shares of q modulo p'. We want to obtain two shares $(d^{(\mathrm{b})})' = \frac{x^{(\mathrm{b})} - q^{(\mathrm{b})} p}{2} \bmod p'$ 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 $q^{(\mathrm{b})} = \mathrm{lsb}(d^{(\mathrm{b})}) - 2 \cdot m^{(\mathrm{b})} \bmod p'$ and $(d^{(\mathrm{b})})' = \frac{x^{(\mathrm{b})} - q^{(\mathrm{b})} p}{2} \bmod p'$.

We recall that the input d is required to be in $[0, p/2)$. As shown in Section 4.3, a malicious client, who does not respect this range, can submit a value between $(-p/2, 0)$ or $[p/2, p)$. However, the impact of this deviation is bounded by twice as much as in Prio.

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 report instead of ℓ OTs, where ℓ could be large, for example, 10–20. We describe our 3-party OT protocol in Figure 11.

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

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 reports $D^{(1)} := \{d_i^{(1)}\}_{i\in[n]}$ and $D^{(2)} := \{d_i^{(2)}\}_{i\in[n]}$, resp.

(1) Each server b obtains $d_i^{(\mathrm{b})}$ in $\{0, \ldots, p-1\}$. Compute $x_i^{(\mathrm{b})} = 2d_i^{(\mathrm{b})} \bmod p$ so that it satisfies $2d_i = x_i^{(1)} + x_i^{(2)} \bmod p$.
(2) Each server b computes $\mathrm{lsb}(x_i^{(\mathrm{b})})$. 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 $q_i^{(\mathrm{b})} = \mathrm{lsb}(x_i^{(\mathrm{b})}) - 2m_i^{(\mathrm{b})} \bmod p'$.
(4) Each server b computes $(d_i^{(\mathrm{b})})' = \frac{x_i^{(\mathrm{b})} - q_i^{(\mathrm{b})} p}{2} \bmod p'$.

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.

4 PRECIO

The Precio protocol involves four parties: a Reporting Origin $\mathcal{R}$ and three helper servers $\{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$. The goal is to arrange a set of reports D collected from C clients into buckets according to a subset of encoded attributes while preserving privacy of individual reports.

We assume that no two helper servers collude. Two servers, say $\mathcal{S}_1$ and $\mathcal{S}_2$, receive secret shared inputs and another two, say $\mathcal{S}_1$ and $\mathcal{S}_3$, output a histogram of the bucket sizes. Internally, two of the servers run DP noise generation and all three run the oblivious random shuffling protocol.

The input to Precio is a dataset of reports $D = (d_i)_{i\in[C]}$, initially secret shared between two 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, another two servers output a vector $D_{\mathsf{priv}} = (d_i)_{i\in[C+n']}$, where the values for an agreed-upon attribute are revealed. Note that the size of D_{priv} is $C + n'$, where n' represents appended dummy reports, ensuring that the outputs revealed to the $\mathcal{S}_i$ and $\mathcal{R}$ and helper servers are differentially private.

4.1 Private Histogram Protocol Description

We describe our Precio protocol $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ in Figure 8. It takes the reports collected from clients $D = (d_i \in G)_{i\in[C]}$ and a query index m, indicating which attribute the histogram is built for, as inputs.

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 (Figure 4). This step inputs the shares of the dataset and the query index m and outputs a new dataset with appended dummy reports $D_{\mathsf{priv}} = (d_i)_{i\in[C+n']}$.

Next, $\mathcal{S}_1, \mathcal{S}_2$ and $\mathcal{S}_3$ execute a 3-party oblivious random shuffling protocol (Figure 6) that mixes up real and dummy reports so that none of the servers can trace back any report to the original set or distinguish between real reports and dummy reports. In other words, it outputs a permuted dataset $D_{\mathsf{priv\_perm}} = \left(d_i'\right)_{i\in[C+n']}$ secret shared between $\mathcal{S}_1$ 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.

We then execute a pruning step, discarding the dummy bucket $\mathcal{B}_\perp$ and buckets with fewer than a threshold t reports. The details are discussed in Section 4.5). After pruning, we reorganize output shares from $\mathcal{S}_1$ and $\mathcal{S}_3$ back to $\mathcal{S}_1$ and $\mathcal{S}_2$, enabling repeated use of $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$.[5]

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

4.2 Security and Privacy Analysis of $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$

(Informal) Robustness Analysis with Malicious Clients. In this section, we give a robustness bound against malicious clients. Note that the worst a malicious client can do is input inauthentic attribute values within the valid range in its reports, because Precio uses a length-preserving secret sharing mechanism.

Informally, the $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ protocol protects against a small subset of malicious clients. Since the shares of the attributes preserve the length of the original attribute size, a small subset of clients 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 effect of noise as long as only small set of clients are allowed to do that.

Formally, we prove the following.

Theorem 1. *Let* N *be the number of malicious clients. The* L_1 *distance between a true histogram and an incorrect histogram output by* $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ *(Figure 8) without noise 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 the true bucket will have $\mathsf{a}-1$ reports while the incorrect bucket will have $\mathsf{b}+1$. The L_1 distance between the true histogram and the incorrect histogram is the sum over all buckets of the absolute values of the difference between the two counts (from the two 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 $2\mathsf{N}$. □

(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 from the reporting origin. We formally prove the privacy of random shuffling in Section C.1.

As we will describe shortly, in the layered protocol (Algorithm 1) 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 this into account in the privacy analysis in Section 4.5.

[5]This reorganizing is not crucial for functionality or security but streamlines presentation. Alternatively, the next layer could run with reversed roles for S_2 and S_3.

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

Parameters. C as the total number of collected reports. Pruning threshold t. Parameters $\epsilon_{\mathsf{buckets}}, \epsilon_{\mathsf{flush}}, M_{\mathsf{buckets}}, M_{\mathsf{flush}}$ 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 datasets 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 $d'_i[\mathsf{m}]$.

(4) (**Pruning and organizing**) Discard the dummy bucket $\mathcal{B}_\perp$ and each bucket $\mathcal{B}_j$ revealed in previous step with $|\mathcal{B}_j| < t$, along with all the reports in them. For the shares of noisy datasets 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 it 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 5).

Output. Output the attribute value of index m for each remaining bucket. Servers keep the remaining dataset shares as private output for use as inputs in further executions of the protocol.

Figure 8: Our Histogram Protocol from Oblivious Shuffling.

4.3 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 report using modulus p'. 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 computed sum is always between $-p'/2$ and $p/'2$. The lower bound on p' allows us to guarantee that the computation of the sum modulo p' does not overflow.

Our valid values are in $[0, p/2)$. In the worst case, a malicious client can change $p/2$ into $-p/2$, so cheat by $-p$ or change 0 to p, so cheat by p. Thus, the potential error from cheating is $[-p, p]$. This is roughly equivalent to a client submitting two adversarially chosen inputs in the range of the numerical attribute. In comparison, Prio restricts clients to submit a single value in the valid range.

In the end, the computed sum is in $(-pC/2 - M_{\mathsf{sum}}, pC + M_{\mathsf{sum}})$, where M_{sum} is a bound on the noise added by the helper servers. As for robustness, we show in our full analysis that if the number of malicious clients is bounded by N, then the difference between true sum and the obtained sum is in $(-Np, Np)$.

Malicious Clients. As explained above, after the helper servers run the protocol in Figure 7 for each report, each server locally sums $\left(d_i^{(\mathrm{b})}\right)'$ modulo p' reduced in the interval $(-p'/2, p'/2)$ and reveals the results. This means that what matters is, for each report, the value of $d'^{(1)} + d'^{(2)} \bmod p'$ reduced into $(-p'/2, p'/2)$ which we denoted as d. Honest clients make the computation of d into $[0, p/2)$. However, malicious clients can share d out of range in two specific intervals:

(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) $x = 2d + p \in (0, p-2]$ 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 $d^{(\mathrm{b})}$, they both compute $x^{(\mathrm{b})}$ to share x, mutually compute q, and $x' = x^{(1)} + x^{(2)} - qp$. However, we now have $x = x^{(1)} + x^{(2)} - (1-q) \times p$, because $q = \mathrm{lsb}(x^{(1)}) \oplus \mathrm{lsb}(x^{(2)})$ and x is odd. We use $1 - q = q + (-1)^q$. Then, $x = x^{(1)} + x^{(2)} - (1-q) \times p = x = x^{(1)} + x^{(2)} - (q + (-1)^q) \times p = x' - (-1)^q \times p$ which is different than x' servers obtained: $x' = x + (-1)^q p = 2d$.

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

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

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

It is easy to show that (1) and (2) are the only possible malicious injection out of range d by a malicious client. Indeed, what matters is $x = x^{(1)} + x^{(2)} - q \times p$ with $x^{(1)}, x^{(2)} \in [0, p)$ and $q \in \{0, 1\}$ chosen by the client.

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 $x^{(\mathrm{b})}$ 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 $\left(-\frac{p}{2}, \frac{p}{2}\right)$.

The sum with malicious clients reports is in $\left(-\frac{Cp}{2}, Cp\right)$, where C is the number of clients. Thus, we require that $p' \geq 2pC + 2M_{\mathsf{sum}}$ 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.

A malicious client can inject a value in the sum which is in $[d - p + 1, d + p - 1]$, where d is the true value. If the number of malicious clients are bounded by N, the difference between the true sum and the obtained sum is in $[-N(p-1), N(p-1)]$, thus N **malicious clients cannot make the result more than Np off from the true sum.**

4.4 Layered Protocol

Our protocol can aggregate reports with multiple attributes by recursively invoking $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$, allowing queries like

SELECT COUNT(Ads Category) FROM D
WHERE Gender = "She" GROUP BY Ads Category

provided sufficient data in the bucket. This also offers performance gains for handling large attributes with sparse domains. For example, if we have 2^{16} reports with an attribute of 32 bits, we can divide the attribute into three smaller ones and prune after each layer. Details on the layered algorithm are in Algorithm 1 and complexity analysis in Appendix B.

The layered protocol works so that after it has processed the first attribute m_1, only the bits corresponding to m_1 are revealed. Then, it proceed to 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. This is repeated until all desired attributes are processed.

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),

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 $\mathcal{M}$.

procedure LAYERED($\mathcal{M}$, label , $i, D^{(1)}, D^{(2)}$)
  if (label, i) = ($\perp$, $\perp$) **then**
    Set label = null, $i = 1$.
  end if
  if $i \le \lambda$ **then**
    Call $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$ with inputs ($\mathsf{m}_i, D^{(1)}, D^{(2)}$).
    for each produced bucket $\mathcal{B}_j$ **do**
      Set v to the attribute value $\mathcal{B}_j$.
      Set $D^{(1)}$ and $D^{(2)}$ to the shares of the bucket.
      Call LAYERED($\mathcal{M}$, label $\mid v, i+1, D^{(1)}, D^{(2)}$).
    end for
  else
    Output label.
    Subtract the parameter $2M$ (used in $\Pi^{\mathsf{Hist}}_{\mathsf{Precio}}$) from the number of reports in $D^{(1)}$ and output the result.
  end if
end procedure

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, $B' = \{\mathcal{B}_i : |\mathcal{B}_i| \ge t\}$, and continue bucketization for the next attribute for each remaining bucket in B'. We will detail the meaning and computation of the threshold shortly in Section 4.5.

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 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 reports 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.[6]

4.5 Differential Privacy and Pruning

The protocol in Algorithm 1 performs depth-first exploration of buckets layer by layer. Layers are based on logically separate attributes or smaller subdivisions of large attributes. The algorithm outputs noisy histograms per layer or when the algorithm ends. Analysts can prune smaller buckets at each layer, which is vital due to long-tailed attribute value distributions.

Differential Privacy Parameters. Our histogram protocol uses DP in two ways: 1) dummy reports are added for each (non-dummy) bucket at each layer to provide $(\sigma_{\mathsf{buckets}}, \delta_{\mathsf{buckets}})$-DP for bucket counts; 2) dummy reports are added to the dummy bucket to provide $(\sigma_{\mathsf{flush}}, \delta_{\mathsf{flush}})$-DP for the total number of dummies added. All reports in the dummy bucket are *flushed* before moving to the next layer. This separation into two sets of DP parameters improves both efficiency and privacy. Because the flush noise has no impact on accuracy, we can use *less* privacy budget for that. The two sets of DP parameters together determine the total DP parameters (ϵ, δ). We use the work of Gopi *et al.* from [27–29] for accurate DP composition.

Let us work through some examples. Our goal is to reach (ϵ, δ)-DP for the layered protocol. We start with a single attribute, running only one layer. No dummies are needed for the dummy bucket $\mathcal{B}_\perp$ (flush noise) because no original report falls in it, so we only need parameters for $(\sigma_{\mathsf{buckets}}, \delta_{\mathsf{buckets}})$. Two datasets are neighbors (according to our DP definition) when they differ in one report having a different attribute value. Thus, we have non-dummy buckets that differ in counts and both contribute to ϵ equally. To achieve $(\epsilon, \delta) = (2, 2^{-40})$, we use $(\sigma_{\mathsf{buckets}} = 4.75, \delta_{\mathsf{buckets}} = 2^{-40})$, and $M_{\mathsf{buckets}} = 35$.

For two layers, we will need two attributes. Again, our neighboring notion means that one report is replaced with another report. If the reports differ in both attributes, in the first layer two regular buckets are impacted and the sensitivity will be 2. We reveal noisy counts for both buckets.

In the second layer, noisy counts for each attribute bucket and the dummy bucket are revealed. For the attribute buckets the sensitivity is again 2, as for the first. Analyzing the privacy impact of the dummy bucket is trickier, as it is not directly affected by either of the reports. Let m_1, m_2 be the chosen attributes for the first and second layer. For each $i \in G_{m_1}$, let the noisy count for the corresponding bucket $\mathcal{B}_i$ be c_i and the exact count e_i. The exact dummy count is $c_i - e_i$, which is a function of the dataset depending on the previous result c_i. The exact dummy counts $c_i - e_i$ differ in at most two of the buckets $\mathcal{B}_i$, so sensitivity of the count in the dummy bucket in the second layer is again 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 [27]. Then, we set $M_{\mathsf{buckets}} = 50$ and $M_{\mathsf{flush}} = 250$.

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. See sample parameters in Table 3 and detailed analysis in Section C.2.

Pruning Parameters. The analyst may want to say that with high probability buckets with a sufficiently high real report count (*i.e.*, not dummies) will not be pruned. Unfortunately, it is not straightforward how to make such a statement based on a chosen pruning threshold t.

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

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

Table 3: Example parameters for an analyst who needs λ layers, $(\epsilon, \delta) = (2, 2^{-40})$-DP, $C = 10^7$, $t_{\text{true}} = 1000$, and error threshold $q = 1\%$.

To address this, we introduce a reparameterization t_{true} of the pruning threshold t. The analyst chooses t_{true} as a threshold as well as a maximum error probability q, such that no buckets with less than t_{true} real reports are pruned, except with probability $\leq q$.

Note that the analyst's query parameters, including t_{true}, are not private and can be known to all parties.

To choose t accordingly, let n be a Gaussian noise so that $M + n$ dummy reports are added to a bucket. Note that if we have C reports, there can be at most C/t_{true} buckets that we need to retain. Set

$$q = \frac{\lambda C}{t_{\text{true}}} \Pr\left[t_{\text{true}} + 2M + n^{(1)} + n^{(2)} < t\right]$$

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

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

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

Given this formula, the analyst could query all buckets with $C = 10^7, \lambda = 8$, and pruning threshold $t_{\text{true}} = 1000$ for true counts with 1% error, *i.e.*, $q = 0.01$. For $\sigma = 13.5$, we obtain $M = 101$ and $t = 1104$. 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.

We can observe that the number of layers λ has very little effect on t. Indeed, for λ from 1 to 32 (and the rest of the parameters fixed), the value of t goes from 1112 down to 1099. Similarly, for C from 10^3 to 10^9, t goes from 1145 to 1089.

Example. Consider a scenario where an analyst has a collection of $C = 10^7$ 128-bit reports and wants to investigate where their ad revenue is coming from. The analyst is interested in four attributes: a location identifier (8 bits), a product identifier (12 bits), an age bracket (3 bits), and the number of dollars spent (numerical attribute with 16 bits). They tell the helpers they want a 4-layer query ($\lambda = 4$) and the helpers set the DP parameters accordingly. The analyst wants to find demographics that have fairly large impact so they set $t_{\text{true}} = 1000$. They set $q = 0.01$ as the maximum error.[7]

The analyst chooses to first partition by product identifier, because they want to investigate which products are most successful. They choose to further explore the top 10 products. For each, they filter by location identifier and next by age bracket.[8] They compute the dollars spent per each combination of (product, location, age).

The next day, the analyst can take the next set of 10^7 reports and decide to examine other combinations of attributes, *e.g.*, (location, device type, browser vendor/version). This would not require pushing any change to the clients, as it would not require any change in the format of the reports.

5 PERFORMANCE EVALUATION

We implement layered Precio in Rust. To demonstrate its performance, we run experiments with both full-domain and subset-histograms comparing the results with prior works when possible. 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 Azure Standard E16ads v5 VM with 16 vCPUs (2.45 GHz) and 128 GB RAM. Our experiments didn't exactly 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 [7, 8] and Prio [1, 15]. 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 3. For each execution, $C = 10^7$ 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 Precio with the iDPF code from [26], which is non-interactive and produces a full histogram. The benchmarks in [26] measure the average processing times for one report on one server; multiplying by $C = 10^7$ gives their total runtime.[9] The results are shown in Table 4. For $\ell \in \{16, 18, 20, 22\}$, our protocol is at least **four orders of magnitude more efficient** than the iDPF protocol for generating full histograms from short reports.

5.3 Constructing a Subset-Histogram via Pruning

We benchmark a subset histogram aggregation. We tested Precio on two layers with Zipf-distribution parameter ranging from 1.0 to 1.5; the results are in Figure 9 and Figure 10. As anticipated, the attribute value distribution greatly affects performance in layered histograms. Clearly full domain histograms may not be practical for large domain sizes if the report distribution is too close to uniform.

[7] The helper servers would impose limits on $\epsilon, \delta, t_{\text{true}}$ and q, or charge more for values that make the query more costly.

[8] Alternatively, they can combine the age bracket and location identifier into a single 11-bit attribute. Note that this would not require any change to the reports.

[9] [26] provide no communication measurement, however running times are so much longer than ours that the communication cost would hardly make a difference.

ℓ	16	18	20	22
Precio time (seconds)	6.46	16.9	64.28	310.80
Precio comm. (MB)	233	453	1334	4857
iDPF [8, 26] time (days)	1.15	4.75	19	76
iDPF comm. (MB)	2560	2880	3200	3520

Table 4: Running times (single thread) and communication for Precio compared to iDPF. $C = 10^7$ reports are generated from a Zipf-distribution with parameter 1.03. iDPF's communication complexity is estimated based on the server-to-server asymptotic complexity (full histograms) in Table 2.

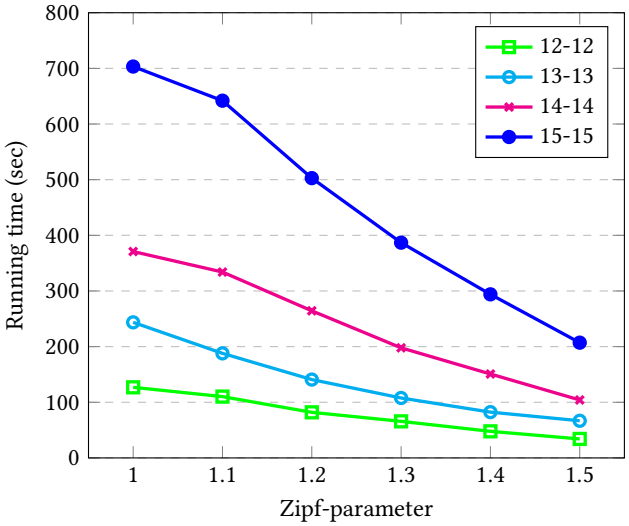

Figure 9: Running time of 2-layer Precio for different distributions of $C = 10^7$ reports. The privacy parameters are from Table 3 with $\lambda = 2$ and the threshold set to 1053.

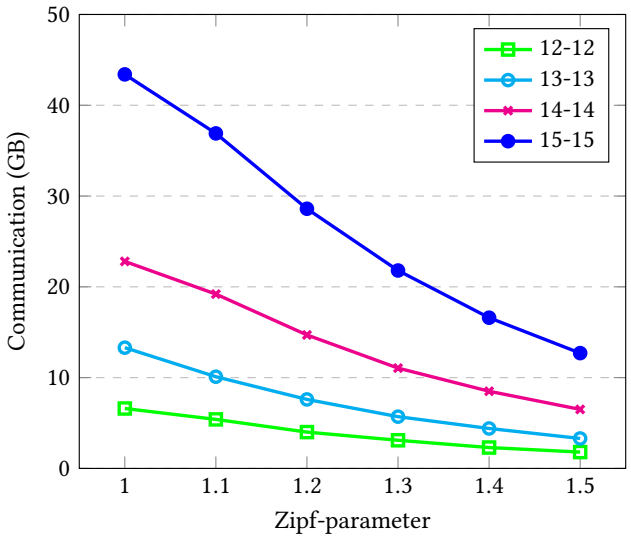

Figure 10: Server-to-server communication for 2-layer Precio for different distributions of $C = 10^7$ reports. Privacy parameters are from Table 3 with $\lambda = 2$ and threshold 1053.

	C = 400K		C = 1M		C = 10M	
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 5: 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.

32-bit Attributes. We sample a sparse set of large 32-bit logical attributes from a Zipf-distribution with parameter 1.03 (as set in [7]). This results in many (near) empty buckets and mimics a setting where the RO is only interested in popular values ignoring 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 4.5. We first synthesize a dataset of 32-bit reports with various C and pruning thresholds 552 and 1053 (obtained from our analysis in Section 4.5 with $t_{\mathsf{true}} = \{500, 1000\}$). Running Algorithm 1 with 2-layer Precio ($\lambda = 2$) yields the results in Table 5. This shows how even large 32-bit attributes can be explored by breaking them down into smaller chunks with layering. Larger λ results in smaller time complexity (Appendix B), but larger communication complexity due to dummy reports being communicated.

5.4 Heavy-Hitters

Here we compare Precio with the subset-histogram in the end-to-end performance evaluation of iDPF [7]. We chose the experiment parameters to be the same as in [7] to make a fair comparison.

In their experiment, $C = 400\,000$ input reports are sampled from a Zipf-distribution with parameter 1.03 and support limited to 10 000. Their report bit-length is 256 and prune threshold $t = \frac{C}{1000}$.

The experiment in [7] is done with two servers (32 vCPUs each) and 61.9ms round-trip latency. Their protocol (as reported in [7]) takes around 53 minutes to generate a subset-histogram. For Precio, we use $\lambda = 16$ with parameters from Table 3 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 $(\epsilon, \delta) = (2, 2^{-40})$.

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 14-bit modulus p and 10^7 reports, Precio takes 43 seconds (on a single thread) and 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 present an efficient 3-party protocol, Precio, for computing privacy-preserving histogram queries. 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 outperforms prior work in many practical scenarios.

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, Aug 2023.

[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] Benjamin Case, Richa Jain, Alex Koshelev, Andy Leiserson, Daniel Masny, Thurston Sandberg, Ben Savage, Erik Taubeneck, Martin Thomson, and Taiki Yamaguchi. Interoperable Private Attribution: A Distributed Attribution and Aggregation Protocol. Cryptology ePrint Archive, Paper 2023/437.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

[20] 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.

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

[22] Tim Geoghegan, Christopher Patton, Eric Rescorla, and Christopher Wood. Privacy Preserving Measurement, April 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. Protecting your privacy online. https://privacysandbox.com. (accessed: October 7, 2023).

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

[27] Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. Privacy Random Variable (PRV) Accountant. https://github.com/microsoft/prv_accountant. (accessed: October 7, 2023).

[28] 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.

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

[30] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] Meta. Make smarter business decisions with actionable insights. https://www.facebook.com/business/measurement. (accessed: October 7, 2023).

[37] 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.

[38] Payman Mohassel and Saeed Sadeghian. How to hide circuits in mpc an efficient framework for private function evaluation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 557–574. Springer, 2013.

[39] 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.

[40] 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.

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

[42] Salil Vadhan. The complexity of differential privacy. https://privacytools.seas.harvard.edu/files/privacytools/files/complexityprivacy_1.pdf.

[43] 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.

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

[45] 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.
[46] 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.
[47] 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 LAYERED PROTOCOL

In this section we analyze the 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.*, $t \le M$ 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 $a_j + X_j \ge t$. Thus, the complexity to treat layer i is $\sum_j (a_j + X_j + Y_j) \cdot 1_{a_j + X_j \ge t}$. Because $\sum_j a_j = C$, this complexity is bounded by $C + \sum_j (X_j + Y_j) \cdot 1_{a_j + X_j \ge t}$. 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 $S = \sum_j \mathbb{E}\big[(X_j + (L_i - 1)M_{\mathsf{buckets}} + M_{\mathsf{flush}}) \cdot 1_{a_j \ge t - X_j}\big]$. 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 $a_j \ge t - M$ has a number of terms bounded by $\frac{C}{t-M}$ and is bounded by $\big(\bar{M} + (L_i - 1)M_{\mathsf{buckets}} + M_{\mathsf{flush}}\big)\frac{C}{t-M}$. We sum over all layers and obtain $O\big(\ell C + (L_1 + \cdots + L_\lambda)M_{\mathsf{buckets}} + \lambda M_{\mathsf{flush}}\frac{C}{t-M}\big)$. Having $L_i = 2^{\ell/\lambda}$ for all i, the formula becomes $O\big(\ell C + \lambda M 2^{\ell/\lambda} + \lambda M \frac{C}{t-M}\big)$. We can see the effect of layering, with optimality reached for $\lambda \approx \ell \ln 2$. In the extreme case, with $\lambda = \ell$ and $L_i = 2$, this is $O\big(\ell C + \ell M \frac{C}{t-M}\big)$.

B.1 Expected Value of Noise

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

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 2. *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 4.5). We have $(\sigma, M) = (\sigma_{\text{buckets}}, M_{\text{buckets}})$ composed 2λ times and $(\sigma, M) = (\sigma_{\text{flush}}, M_{\text{flush}})$ 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.* [28]. 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 [29, Proposition B.1]. We generalize their computation to a truncated Gaussian noise. By [28, 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):

$$\mathsf{p} = \mathsf{CDF}_\sigma(\mathsf{t});\ \mu_Y = \frac{1}{2\sigma^2};\ \sigma_Y = \frac{1}{\sigma};\ \mathsf{t}_Y = \mu_Y - \frac{\mathsf{t}+1}{\sigma^2}$$

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

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 $\mathsf{p}, \mu_Y, \sigma_Y$, 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, [28, Theorem 3.3] gives the privacy curve as

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

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 $\Pr[Y_{\text{all}} = \infty] < 2^{-40}$.

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.

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 [15] and Distributed Point Functions (DPF) [7].

E.1 Prio

Prio [15] 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

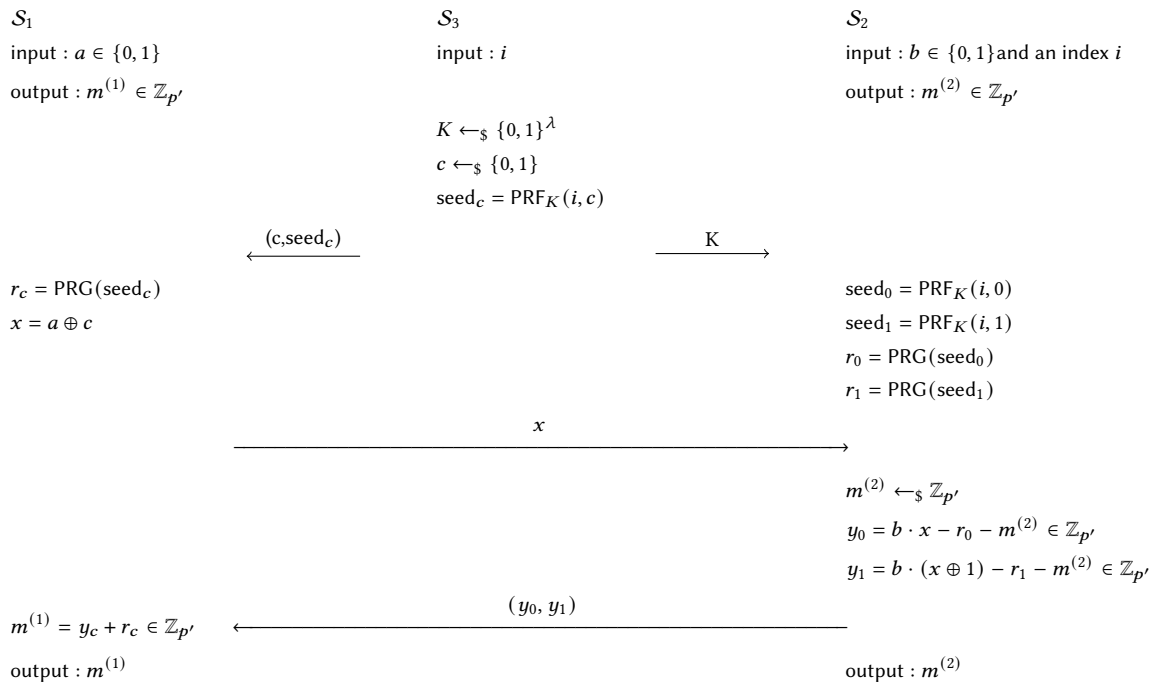

Figure 11: 3-party Oblivious Transfer to compute the arithmetic shares of multiplication of two bits a and b (adapted from ABY[3]). $\mathcal{S}_3$ inputs only the index number i for the i-th iteration. This protocol is run with the same K for each report in parallel. The communication complexity is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

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 [41]. 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).[10]

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

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.