Balanced Additive Randomized Encodings for Shuffle Differential Privacy

Yu Wei[1] Jaspal Singh[1,2] Adya Agrawal[1] Vassilis Zikas[1]

[1]Georgia Institute of Technology
[2]Purdue University

Abstract

Shuffle differential privacy (shuffle DP) offers an attractive distributed alternative to standard differential privacy. It uses a secure shuffler to permute users' randomized encodings, providing individual data privacy without a central trusted entity. A key challenge, however, is to achieve both generality and client efficiency. Under information-theoretic shuffle-DP guarantees, protocols that nearly match central-model utility are restricted to statistical tasks such as summation and histograms. In contrast, in the computational setting, *additive randomized encodings* (ARE), introduced by Halevi et al. (CRYPTO 2023), yield a generic compiler that achieves central-model utility for arbitrary mechanisms. However, their construction incurs prohibitive worst-case computation and communication costs for clients, making it impractical for resource-constrained devices such as mobile phones, IoT sensors, and web browsers.

In this work, we present a client-efficient one-round compiler from central DP to non-robust computational shuffle DP. To achieve this, we first construct a new *balanced* ARE scheme, where all the clients almost equally share the computational burden in the protocol. For a mechanism M over n clients, this reduces the worst-case per-client computation from $O(|M|)$ in prior work to $O(|M|/n)$, where $|M|$ denotes the circuit complexity of the mechanism. The key technical ingredient is a new *permute-XOR* ARE primitive that enables wire splicing across independently generated garbled subcircuits. Secondly we design a more efficient ARE-to-shuffle compiler, whose client bandwidth scales with the sparsity and cross-partition structure parameter of the ARE encoding—a quantity that is always sublinear in n. This is an improvement over Halevi et al. (CRYPTO 2023) where each client bandwidth is $\Omega(n)$. At a high level, the first contribution improves client computation while the second improves client bandwidth.

We provide an implementation of our generic compiler for differentially private tasks including selection, distinct elements, and linear contextual bandits. We obtain shuffle protocols that match the utility of their central-model counterparts with reasonable client overhead. We evaluate our method across a range of practical settings, and observe substantial gains over the compiler of Halevi et al., with per-client computational and bandwidth speedups increasing linearly in the number of clients.

1 Introduction

Differential privacy (DP) [Dwo06] provides a robust framework for ensuring that the output of a randomized algorithm remains statistically insensitive to small changes in its input, thereby safeguarding the privacy of individual data contributors. In the traditional *central model* of DP, a trusted analyzer has access to the raw data of all users and it applies a randomized algorithm to produce differentially private outputs. While this model has been extensively studied and deployed, it relies

on the presence of a trusted central authority, an assumption that may not hold in many real-world scenarios. In such cases, it is necessary to compute DP guarantees over distributed data. Notable examples include Apple's use of DP for telemetry data collection [App25], Google's deployment of the RAPPOR scheme in Chrome browsers to gather usage statistics [EPK14], and distributed DP protocols supporting cross-institutional healthcare data analysis [CGDS+19, AKC+22].

One such practically relevant distributed model is the *shuffle differential privacy* model [BEM+17, CSU+19a], which assumes the presence of a secure randomized shuffler, that is realizable in practice via mix-nets or anonymous routing networks. Each client locally encodes its input through an algorithm Enc; the shuffler applies a uniformly random permutation; and the analyzer applies a decoder Dec to the shuffled batch. Because the analyzer sees only an unordered multiset of encodings, privacy can be amplified beyond what each client achieves locally. In this work we focus on the *non-robust* variant of the model, in which all clients follow the protocol honestly and only the analyzer is adversarial. This is the relevant threat model when the analyzer is operated by a (potentially malicious) service provider while (honest) clients are running an attested or open-source app.

While significant progress has been made in the central DP model, the shuffle DP model remains relatively less explored, with existing results limited to specific functionalities such as summation [BBGN20, IKOS06], d-bin histograms [BBGN19, BC20, CZ22a], and the pointer chasing problem [BC20]. These results demonstrate that, for certain tasks, shuffle DP can achieve accuracy and sample complexity comparable to the central model. However, for general and more complex functions that's not true. To illustrate this gap in the information-theoretic setting, we consider three representative problems:

- **Selection**: Each user provides a bit string $x_i \in \{0,1\}^m$, and the goal is to output an index $j \in [m]$ such that $\sum_{i\in[n]} x_i[j] \geq \left(\max_{j^*} \sum_{i\in[n]} x_i[j^*]\right) - n/10$. In the central model, the exponential mechanism [MT07] achieves ϵ-DP with a sample complexity of $O_\varepsilon(\log m)$. In contrast, the best known shuffle DP mechanism for selection [CSU+19a] requires a sample complexity of $O_\varepsilon(\sqrt{m})$, which is significantly larger for large domains.
- **Distinct Elements**: This task estimates the number of distinct elements from n user inputs over a domain $\mathcal{X}$. In the central model, the Laplace mechanism [Dwo06] efficiently solves this problem. However, the best known (ε,δ)-DP protocol in the shuffle model [CGKM21] incurs an additive error of $\sqrt{\min(n,|\mathcal{X}|)}\cdot\mathrm{poly}(\log(1/\delta)/\epsilon)$, which scales with the square root of the domain size, potentially exponentially larger than n.
- **Linear Contextual Bandits**: This task learns an unknown parameter θ^* from T rounds of contextual interactions in order to minimize cumulative pseudo-regret. In the central model, Shariff and Sheffet [SS18] give an algorithm achieving $O_\varepsilon(\sqrt{T})$ regret. In contrast, the best known (ε,δ)-DP protocol in the shuffle model [CZ22b] attains $O_\varepsilon(T^{3/5})$ regret, strictly weaker than central but improving over local DP's result, $O_\varepsilon(T^{3/4})$.

These gaps are not artifacts of suboptimal constructions: several arise from information-theoretic lower bounds intrinsic to the shuffle model. However, all real-world instantiations of the shuffle model would inherently rely on cryptography, and in most cases public-key cryptography. Therefore, an interesting and realistic relaxation, which we also adopt here, focuses on *computational* DP; here the adversary is computationally bounded and is allowed a negligible advantage. This setting, and in particular it ability to circumvent information-theoretic limitations was explored by Halevi et al. [HIKR23], which introduced *computational additive randomized encodings* (ARE). In a nutshell, an ARE is a cryptographic primitive that enables one-round secure computation assuming an ideal functionality for summation in presence of a malicious evaluator. Halevi et al. [HIKR23] showed how to compile an ARE scheme for a central DP mechanism to the shuffle DP setting. Their

Function	Computational speedup	Communication speedup
Gaussian sum	**386**×	**402**×
Selection	**76**×	**198**×
Distinct elements	**114**×	**19**×
Linear contextual bandits	**148**×	**18**×

Table 1: Per-client speedup of the balanced protocol over the unbalanced baseline of [HIKR23] at n=2^{10} clients.

privacy/utility trade-off matches the central model, generically. This demonstrates the advantages of transitioning to a computational setting. However, their transformation incurs significant costs: it has worst-case client computation $O(|\mathcal{M}|)$, where $|\mathcal{M}|$ is the circuit complexity of the central mechanism, and bandwidth as large as $O(n)$, where n is the total number of clients. These overheads are prohibitively large for resource-constrained devices, such as those commonly found in distributed DP settings.

1.1 Our Contribution

We design and implement client-efficient, one-round, non-robust shuffle CDP protocols whose privacy-accuracy trade-offs match the central model. Our protocols achieve sublinear per-client computation *and* sublinear per-client bandwidth in both the mechanism circuit complexity[1] $|\mathcal{M}|$ and the number of clients n. Concretely, our contributions to the ARE literature and the shuffle DP setting are as follows:

1. **Balanced ARE.** We construct a new ARE scheme that distributes the heaviest operation, i.e., garbling work *evenly* across all n clients, dropping the worst-case per-client computation from $O(|\mathcal{M}|)$ in [HIKR23] to $O(|\mathcal{M}|/n)$. The key technical ingredient is a new *permute-XOR* (PX) ARE primitive that lets independently garbled subcircuits be spliced together at runtime. For mechanisms with subquadratic circuit complexity, this ensures sublinear per-client computation.
2. **ARE-to-shuffle compiler with sparse encodings.** We give a new compiler that exploits the structural sparsity of balanced ARE encodings. By tagging non-zero shares with their (public) coordinate before secret-sharing into the shuffler, per-client bandwidth drops from $O(n)$ to $O(t)$, where t is the maximum number of cross-subcircuit wires in a near-balanced n-way partition of $\mathcal{M}$. In the worst case, $t = O(|\mathcal{M}|/n)$, ensuring sublinear bandwidth for subquadratic mechanisms. For circuits with favorable partitionings (e.g., leveled circuits with fixed width), bandwidth complexity can be further reduced, as discussed in Section 5.1.
3. **Implementation and evaluation.** We provide an end-to-end C++ proof-of-concept implementation built on the BN254 pairing curve and emp-toolkit's half-gates garbling. We evaluate multiple circuit-partitioning strategies and benchmark our protocol against a faithful re-implementation of the [HIKR23] compiler. We instantiate both the compilers for multiple concrete DP tasks, including Selection, Distinct Elements, and Linear Contextual Bandits. These protocols are realized via leveled, data-oblivious circuits keeping per-client computation and communication polylogarithmic in n. We observe orders of magnitude speedup in per-client computation and bandwidth. (refer Table 1)

These advancements enable client-efficient shuffle DP mechanisms that closely align with the privacy and accuracy guarantees of the central model, paving the way for practical deployment in resource-constrained distributed environments.

[1]Throughout this paper we denote by $\mathcal{M}$ the circuit that computes the mechanism; but for brevity we often refer to $\mathcal{M}$ as the mechanism.

Related work. We defer an additional related work on central DP, shuffle DP, computational shuffle protocols, multi-server model of computation, and recent ARE constructions in Appendix B.

2 Technical Overview

In this section we recall how the ARE scheme of Halevi et al. [HIKR23] compiles a central DP mechanism into a one-round shuffle CDP protocol, and identify the two bottlenecks — $O(|\mathcal{M}|)$ client computation and $\Omega(n)$ client bandwidth. We then sketch our two main techniques: *Balanced ARE*, which spreads the garbling work evenly across clients via a new *permute-XOR* (PX-ARE) splicing gadget, and a *sparse-encoding ARE-to-shuffle compiler* that exploits the structure of balanced ARE encodings to reduce per-client bandwidth.

2.1 Shuffle DP from ARE [HIKR23]

A computational ARE scheme for any polynomial-size function f on n inputs consists of three algorithms $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$.

- $\widetilde{\mathsf{Setup}}$ generates public parameters pp shared by all n clients and the evaluator.
- $\widetilde{\mathsf{Enc}}$ inputs (pp, i, x_i) and it outputs an encoding $\hat{x}_i$ for client i's input x_i.
- $\widetilde{\mathsf{Dec}}$ takes $(\sum_{i=1}^{n} \hat{x}_i, \mathsf{pp})$ as input and outputs $f(\vec{x})$ for $\vec{x} = (x_1, \ldots, x_n)$.

Correctness ensures that $\widetilde{\mathsf{Dec}}$ recovers $f(\vec{x})$, and privacy guarantees that the evaluator learns nothing beyond $f(\vec{x})$ itself. The compiler also assumes an efficient shuffle protocol $(\mathsf{Enc}_{\mathrm{Sum}}, \mathsf{Dec}_{\mathrm{Sum}})$ for the summation functionality [IKOS06], where an adversary learns nothing more than the sum of the user inputs. Hence, a natural way as suggested in [HIKR23] to compute a central DP-private mechanism $\mathcal{M}(x_1, \ldots, x_n)$ on n inputs in shuffle model is to use an ARE scheme $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ as follows:

ARE-to-shuffle-DP compiler [HIKR23]

Client i: send $\mathsf{Enc}_{\mathrm{Sum}}(\widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i))$ into shuffler.

Analyzer: Given output of the shuffler y, compute and return $\widetilde{\mathsf{Dec}}(\mathsf{pp}, \mathsf{Dec}_{\mathrm{Sum}}(y))$.

By correctness of shuffle sum functionality $\mathsf{Dec}_{\mathrm{Sum}}(y) = \sum_i \widetilde{\mathsf{Enc}}(x_i)$ and by the correctness of the ARE scheme, this protocol outputs exactly $\mathcal{M}(\vec{x})$. By privacy property of ARE, y can be simulated against a computational adversary given just $\mathcal{M}(\vec{x})$, which by definition satisfies central DP privacy guarantee. This in turn implies that y, the output of the shuffler or the view of the adversary satisfies computational DP.

This provides a clean compiler from ARE to shuffle-DP protocols in the computational setting. Unfortunately this construction has poor client computation and bandwidth complexity. To highlight this limitation we unpack the ARE construction in a little more detail.

2.1.1 ARE construction from [HIKR23]

This scheme is built out of two cryptographic primitives:

- Garbling schemes: Consists of a pair of functions $(\widehat{\mathsf{Garble}}, \widehat{\mathsf{Eval}})$ such that, $\widehat{\mathsf{Garble}}$ takes as input a circuit $\mathcal{C}$ (let say on m-bit input x) and it outputs a garbled circuit $\widehat{\mathcal{C}}$ and garbled input keys

$\{k_{i0}, k_{i1}\}$ for each input $i \in [m]$. Further, the $\widehat{\mathsf{Eval}}$ function can be used to evaluate the garbled circuit on any given garbled inputs:

$$y \leftarrow \widehat{\mathsf{Eval}}(\widehat{\mathcal{C}}, k_{ix_i})$$

where correctness implies $y = \mathcal{C}(x)$ and privacy implies the input to $\widehat{\mathsf{Eval}}$ can be computationally simulated given just the final output y

- OT-ARE: An ARE scheme for the oblivious transfer (OT) functionality $f_{OT}((m_0, m_1), b) = m_b$, where one client inputs a pair of messages, and the other client inputs the selection bit. Unlike a traditional OT protocol between two parties, one can think of this as a three party scheme where 2 clients provide inputs and the evaluator is the output party. This can be built from public-key assumptions [HIKR23] or more recently from information-theoretic constructions [BEGI25].

Let $\mathcal{C}$ correspond to a circuit representation for $\mathcal{M}$. In the [HIKR23] ARE scheme, one client garbles $\mathcal{C}$ and sends its garbled circuit $\widehat{\mathcal{C}}$ to the evaluator. To evaluate $\widehat{\mathcal{C}}$, the evaluator needs the input labels $\{k_{ix_i}\}$ for every bit x_i of the m-bit input. This can be achieved using the OT-ARE primitive. For each input i, two parties run an OT-ARE: the garbler (holding $\{k_{i0}, k_{i1}\}$) and the input owner (holding x_i) — this results in the evaluator learning just the correct key label for input wire i. Since all m OT-AREs for each input wire must be executed in parallel, the resulting scheme has ARE encodings that are vectors of size $m = \Omega(n)$.

This scheme requires a single client to garble the entire mechanism, leading to worst case client complexity $O(|\mathcal{M}|)$. Further, the scheme has bandwidth $\Omega(n)$ per client, since secure summation (in ARE to shuffle DP compiler) must operate on n-length vectors, and all known shuffle-based sum protocols have cost linear in input length [IKOS06].

2.2 Reducing computation: Balanced ARE

We introduce a new *balanced* ARE scheme, which has a more balanced computational load across all clients compared to [HIKR23]. To achieve this, we partition $\mathcal{C}$ into n disjoint, near-equal-size subcircuits $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ and have client i garble only $\mathcal{C}_i$. Each client then ships its own garbled subcircuit $\widehat{\mathcal{C}}_i$ to the analyzer. The challenge is that the analyzer must now evaluate the joint garbled circuit even though the subcircuits were garbled *independently* — so the wire labels of $\mathcal{C}_i$ are unrelated to those of $\mathcal{C}_j$, even when an output wire of $\mathcal{C}_i$ feeds an input wire of $\mathcal{C}_j$. We supply two gadgets which together let the analyzer evaluate the spliced circuit gate-by-gate in topological order.

Gadget 1 (Input wire): For each external input wire w of $\mathcal{C}$ that feeds gate $g \in \mathcal{C}_i$, the input owner of w runs an OT-ARE with client i (the garbler of $\mathcal{C}_i$): client i supplies the two wire labels (K_{w0}, K_{w1}), the input owner supplies the bit x_w, and the analyzer learns K_{wx_w}. This is the same primitive used in [HIKR23], but we apply it to individual subcircuits.

Gadget 2 (wire splicing): For each *boundary* wire w that is an output of $\mathcal{C}_i$ and an input of $\mathcal{C}_j$, the analyzer must translate the label $K_w \in \{K_{w0}, K_{w1}\}$ produced by $\widehat{\mathcal{C}}_i$ at run-time into the corresponding label $s \in \{s_0, s_1\}$ used by $\widehat{\mathcal{C}}_j$, without ever revealing the underlying logical bit on w. We achieve this with a new ARE primitive we call *permute-XOR* (PX-ARE), which realizes the two-party functionality

$$f_{\mathsf{pxt}}((s_0, s_1), (b, (s'_0, s'_1))) = (s_b \oplus s'_0, s_{1-b} \oplus s'_1).$$

We also use "point-and-permute" garbled circuits [NPS99], where each wire is assigned a secret permutation bit α_w, and each label carries a publicly readable *color* bit (its LSB) chosen so that $\mathrm{LSB}(K_{wv}) = v \oplus \alpha_w$. The evaluator uses color bits to pick which row of each garbled gate to

decrypt, and at run-time learns the color of K_w but not the logical bit v itself.

We next describe our splicing technique assuming PX-ARE in a little more detail: client i (the garbler of $\mathcal{C}_i$, who knows both K_{w0} and K_{w1}) sets bit $b \leftarrow \mathrm{LSB}(K_{w0})$ and reorders its labels by color: $(K'_{w0}, K'_{w1}) = (K_{w0}, K_{w1})$ if $b = 0$, else $(K'_{w0}, K'_{w1}) = (K_{w1}, K_{w0})$. After this reordering, K'_{wc} is unconditionally the label of color c. Client i then feeds $(b, H(K'_{w0}), H(K'_{w1}))$ into the PX-ARE, while client j feeds (s_0, s_1). Hence, the PX output is

$$\big(s_b \oplus H(K'_{w0}),\ s_{1-b} \oplus H(K'_{w1}) \big)$$

At evaluation time, the analyzer holds some $K_w \in \{K_{w0}, K_{w1}\}$ from $\widehat{\mathcal{C}}_i$ with color $c = \mathrm{LSB}(K_w)$; by the reordering, $K_w = K'_{wc}$. It selects the c-th component of the PX output and XORs with $H(K_w) = H(K'_{wc})$, recovering $s_{b \oplus c} = s_v$, where $v = b \oplus c$ is the actual bit on wire w. Since $\widehat{\mathcal{C}}_j$'s gates are themselves point-and-permute and route on color, the recovered s_v is exactly what the analyzer needs to continue evaluation. Two privacy properties make this safe: the analyzer never sees the unselected label s_{1-v}, because it is one-time-pad-masked by $H(K'_{w,1-c})$, whose preimage is unknown; and the analyzer never sees the permutation bit b.

Further in Section 4.1 we describe how to construct efficient ARE for PX-ARE functionality by reducing it to ARE for OT functionality.

To summarize, using these two gadgets the analyzer can evaluate the joint garbled circuits in gate by gate fashion in a topological ordering, using gadget 1 to receive each input wire key label and using the wire-splicing gadget 2 whenever evaluation crosses subcircuits. This technique of splicing together subcircuits is partly inspired by LEGO framework of Nielsen and Orlandi [NO09]. They optimize malicious secure garbled circuit design using fault tolerant circuits and a somewhat similar splicing of subcircuits, most of which are correctly generated.

In our balanced ARE scheme, the computational load of each party is dominated by the cost to garble their subcircuit. With balanced partitioning, each client computation is worst-case $O(|\mathcal{C}|/n)$ — a factor n improvement over previous works. For the DP mechanisms we study (Section 6), $|\mathcal{C}|$ is at most quasi-linear in n, so per-client computation is sublinear, and in fact polylogarithmic in n for our leveled-circuit instantiations.

2.3 Reducing bandwidth: ARE with sparse encoding

Using the ARE framework to achieve shuffle DP requires computing secure sum of ARE encoding vectors assuming a shuffler. Balanced ARE encodings are length-k vectors over some group $\mathbb{G}$, but only a small t subset of coordinates are non-zero on any given client. Concretely, client i writes into exactly the coordinates corresponding to (a) the input wires it owns, (b) the input wires of subcircuit $\mathcal{C}_i$, and (c) the boundary wires of $\mathcal{C}_i$. The set of these coordinates is determined by the public circuit partition, so the *positions* of non-zero entries are public, even though their *values* are not.

We exploit this to refine of the split-and-mix shuffle-summation protocol of [IKOS06] for sparse vectors. Rather than secret-share the full k-vector (and pay $O(k)$ per client as in [IKOS06]), the client secret-shares only its t non-zero entries, tagging each share with its (public) coordinate index so that the analyzer can re-bucket shuffled shares by index. At a high level, our optimized sparse vector sum protocol in the shuffle model is as follows:

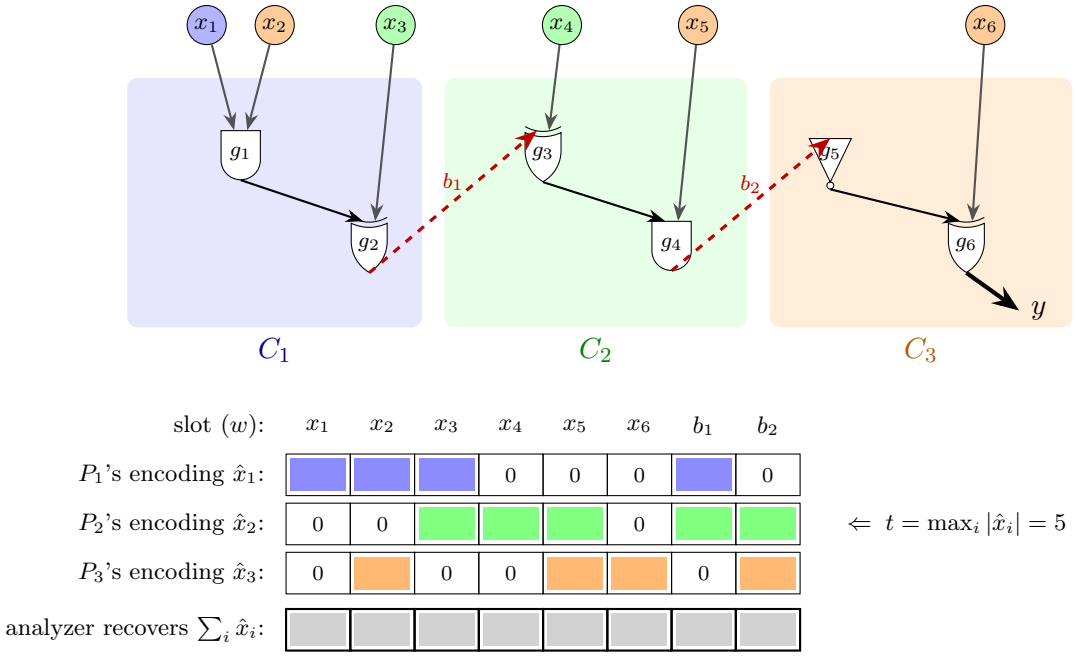

Figure 1: Balanced ARE on a circuit $\mathcal{C}$ with $n = 3$ clients and $|\mathcal{C}| = 6$ gates. **Top:** the circuit is partitioned so client P_i garbles a subcircuit C_i of size 2. Inputs $x_1, \ldots, x_6$ are color-coded by owner: x_1 owned by P_1 (blue); x_3, x_4 by P_2 (green); x_2, x_5, x_6 by P_3 (orange). Only two wires $b_1 = g_2 \to g_3$ and $b_2 = g_4 \to g_5$ (red dashed) cross partition boundaries. Other wires are external inputs (gray) or internal to a partition (black). **Bottom:** each client's ARE encoding is a length-k vector with $k = 8$ slots — one per external input wire, one per boundary wire. A coloured cell denotes a non-zero contribution; in the sparse compiler only these slots are secret-shared into the shuffler. Client P_i writes into the slots for (a) the input wires it owns, (b) the input wires of C_i, and (c) the boundary wires incident to C_i.

Sparse split-and-mix

Input: sparse vector $v \in \mathbb{G}^k$ with non-zero support $\mathcal{I}_i \subseteq [k]$, $|\mathcal{I}_i| \le t$.

Encoding: for each $j \in \mathcal{I}_i$, client i sample λ random shares $v_{j,1}, \ldots, v_{j,\lambda} \in \mathbb{G}$ with $\sum_\ell v_{j,\ell} = v_j$; submit the set $\{(j \,\|\, v_{j,\ell}) : j \in \mathcal{I}_i,\ \ell \in [\lambda]\}$ to the shuffler.

Decoding: on receiving the shuffled set S, the analyzer computes $y_j \leftarrow \sum_{(j\|u)\in S} u$ for each $j \in [k]$ and outputs vector y.

Note that Tagging shares with their coordinate indices does not leak any private information because the support pattern is fixed by the public circuit partition. Or in other words, the number of shares the analyzer receives for each coordinate by each party is a function of only the circuit partitioning and is independent of their inputs.

In this scheme per-client bandwidth is $O(t\lambda \log |\mathbb{G}|)$, where λ is the security parameter. This also matches the client bandwidth in our Balanced ARE scheme when its encodings are t-sparse. In Section 6 we show that for all DP mechanisms of interest, t is sublinear in n, asymptotically improving over prior work.

Composing this sparse compiler with Balanced ARE gives our final shuffle CDP construction, which strictly improves on [HIKR23] in both per-client computation and bandwidth for every subquadratic-circuit DP mechanism. A pictorial representation of our circuit partitioning and sparse encoding technique on a concrete and small circuit is also provided in Figure 1.

3 Preliminaries

As our work discusses new Additive Randomized Encoding (ARE) constructions and its application to shuffle DP protocols, this section reviews the concepts of AREs and (computational) differential privacy in the shuffle model. A brief introduction to other primitives and protocols used in the result can be found in Appendix C.

Notations. We denote by $\mathtt{negl}(\cdot)$ a negligible function. We denote by $\mathsf{Perm}(\mathcal{S})$ a random permutation over the set $\mathcal{S}$. Let $\mathcal{M}$ be an algorithm (or mechanism), and let $\mathrm{Dom}(\mathcal{M})$ denote its input domain. Throughout, λ denotes the computational security parameter.

3.1 DP in the Shuffle Model

Our compiler takes a differentially private (DP) mechanism and outputs a shuffle protocol whose analyzer's view satisfies a computational analogue of the same privacy guarantee. We briefly recall the notion of computational differential privacy (CDP) in the shuffle model, and defer the full definitions of related concepts to Appendix C.2.

Differential Privacy requires that the output of mechanism $\mathcal{M}$ is similar on all *neighboring* inputs D, D' that differ in exactly one data point (we also call D, D' *neighbors*). Informally, if $\mathcal{M}$ is (ε, δ)-DP, any adversary's ability to decide whether $\mathcal{M}$ was run on D or D' is bounded by δ and e^{ε}:

$$\Pr[\mathsf{Adv}(\mathcal{M}(\mathrm{D})) = 1] \leq e^{\varepsilon} \Pr[\mathsf{Adv}(\mathcal{M}(\mathrm{D}')) = 1] + \delta.$$

In the computational setting, we restrict the class of adversaries Adv to PPT distinguishers and allow an additional negligible term $\mathtt{negl}(\lambda)$ to the privacy parameter δ.

A shuffle protocol is a pair of algorithms $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ that enables n clients and an analyzer to compute a multiparty function over the clients' inputs. Each client i locally encodes its input $(x_i', \hat{x}_i) \leftarrow \mathsf{Enc}(i, x_i)$, after which the encodings are randomly permuted by a shuffler (such as a mixnet). The shuffled encodings are then released to the analyzer. Finally, the analyzer applies Dec to produce the protocol's output. We say that Π is (ε, δ)-computationally differentially private if the analyzer's view

$$\mathsf{View}_{\Pi}(\mathrm{D}) \stackrel{\text{def}}{=} \left(x_1', \ldots, x_n', \mathsf{Perm}(\hat{x}_1, \ldots, \hat{x}_n)\right)$$

forms an (ε, δ)-CDP mechanism in the above sense.

3.2 Additive Randomized Encodings

Additive Randomized Encodings (ARE) [HIKR23] is a cryptographic primitive for studying secure multiparty computation in settings where parties can perform arbitrary local computation, but the only joint operation is a *one-shot* secure summation. More concretely, ARE allows n clients and a server to securely compute a multiparty function f over the clients' inputs. Each client locally encodes its input as an Abelian group element, after which these encodings are securely aggregated via summation over the Abelian group. The final sum is then released to the server. Finally, the server decodes the sum to get the function's output without revealing any additional information about clients' inputs. The formal definition is given below.

Definition 1 (Additive Randomized Encodings (ARE) Scheme)**.** *Let $f : (\{0,1\}^*)^* \mapsto \{0,1\}^*$ be a multiparty function. An ARE scheme for f is a triple of algorithms*

$$\Pi = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$$

with the following syntax:

- $\widetilde{\mathsf{Setup}}(1^\lambda, 1^n, 1^{\ell_A}) \to \mathsf{pp}$ *is a PPT setup algorithm that, given security parameter* λ*, number of parties* n*, and input length* ℓ_A*, generates public parameters* pp*. The public parameters include* λ, n, ℓ_A*, and an explicit description of a direct product of two abelian groups* $\mathcal{G}_1$ *and* $\mathcal{G}_2$*, denoted as* $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2$*. They can also include some randomness, such as random generators of* $\mathcal{G}$ *and/or a common reference string (CRS). We will sometimes eliminate Setup and consider* $\mathcal{G}$ *as being fixed.*
- $\widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i) \to (x_i', \hat{x}_i)$ *is a PPT encoding algorithm that maps an input* x_i *of party* i *to a pair of group elements* $(x_i', \hat{x}_i) \in \mathcal{G}$*, where* $x_i' \in \mathcal{G}_1$ *and* $\hat{x}_i \in \mathcal{G}_2$*. We refer to* $\hat{x}_i$ *as the ARE encoding of* x_i*, which will be securely aggregated later, while* x_i' *is sent directly to the analyzer*[2]*.*
- $\widetilde{\mathsf{Dec}}(\mathsf{pp}, x_1', \cdots, x_n', \hat{y}) \to y$ *is a PPT decoding algorithm that maps a group element vector*

$$(x_1', \cdots, x_n', \hat{y}) \in \mathcal{G}_1^n \times \mathcal{G}_2$$

to an output y*.*

Π *is* ***correct****, with a possible error of* $\beta = \beta(\lambda)$*, if for all* λ, n, ℓ_A*, and* $x_1, \ldots, x_n \in \{0,1\}^{\ell_A}$*:*

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \widetilde{\mathsf{Setup}}(1^\lambda, 1^n, 1^{\ell_A}); \\ (x_i', \hat{x}_i) \leftarrow \widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i); \\ \hat{y} = \sum_{i=1}^n \hat{x}_i; \\ \widetilde{\mathsf{Dec}}(\mathsf{pp}, x_1', \cdots, x_n', \hat{y}) = f(x_1, \ldots, x_n) \end{array}\right] \geq 1 - \beta(\lambda),$$

where summation is taken over the group $\mathcal{G}_2$ *specified by* pp*.*

We say that an ARE scheme Π is computationally secure if any efficient server (or adversary) executing Π learns no more than what is revealed by the output of the multiparty function f. In other words, given its received message, the server cannot distinguish whether it was generated by the actual execution of Π or by an efficient simulator with access only to the output of f. We state it formally in the Appendix C.4

Preliminaries for ARE Construction. Our ARE construction builds on garbled circuits and relies on balanced circuit partitioning. Brief introductions to these concepts can be found in Section 3.3 and Appendix C.5.

Secure sum in shuffle Model. To construct shuffle DP protocols from the ARE scheme, we simulate the secure aggregator of ARE using the shuffler in the shuffle model. For this purpose, we employ the split-and-mix protocol of [IKOS06], which is a secure summation protocol in the shuffle model. A copy of the split-and-mix protocol is given in Appendix C.3.

3.3 Garbled Circuits

Our computational ARE construction builds on garbled circuits [3] with a modular approach. Roughly speaking, a garbling scheme consists of a tuple of algorithms satisfying a set of correctness and security properties. We use the garbling schemes abstraction introduced by [BHR12] and formally state it below.

Definition 2 (Garbling Scheme [BHR12])**.** *A garbling scheme is a quadruple of algorithms* $\Pi = \left(\widehat{\mathsf{Garble}}, \widehat{\mathsf{Enc}}, \widehat{\mathsf{Eval}}, \widehat{\mathsf{Dec}}\right)$ *with the following syntax:*

[2] This definition is equivalent to the original one given in [HIKR23]. For completeness, we include the original definition in appendix C.4.

[3] For a comprehensive treatment of Yao's classical construction, we refer readers to [LP09]

- $\widehat{\mathsf{Garble}}\left(1^\lambda, \mathcal{C}\right) \to (\widehat{\mathcal{C}}, eK, dK)$ *is a PPT garbling algorithm that, given security parameter* λ*, maps a boolean circuit* $\mathcal{C}$ *to a* ***garbled circuit*** $\widehat{\mathcal{C}}$ *with encoding set* eK *and decoding set* dK*.*
- $\widehat{\mathsf{Enc}}\left(eK, x\right) \to \tilde{x}$ *is a PPT encoding algorithm that, given the encoding set* eK *and a valid input* $x \in \mathrm{Dom}(\mathcal{C})$ *for circuit* $\mathcal{C}$*, generates a* ***garbled input*** $\tilde{x}$ *for* x*.*
- $\widehat{\mathsf{Eval}}\left(\widehat{\mathcal{C}}, \tilde{x}\right) \to \tilde{y}$ *is a PPT evaluating algorithm that, given the garbled circuit* $\widehat{\mathcal{C}}$ *and garbled input* $\tilde{x}$*, outputs the* ***garbled output*** $\tilde{y}$*.*
- $\widehat{\mathsf{Dec}}\left(dK, \tilde{y}\right) \to y$ *is a PPT decoding algorithm that maps the garbled output* $\tilde{y}$ *with decoding set* dK *to an output* y*.*

Π *is* ***correct*** *if, given* $\widehat{\mathsf{Garble}}\left(1^\lambda, \mathcal{C}\right) \to (\widehat{\mathcal{C}}, eK, dK)$*, for all* $x \in \mathrm{Dom}C$*:*

$$\widehat{\mathsf{Dec}}\left(dK, \widehat{\mathsf{Eval}}\left(\widehat{\mathcal{C}}, \widehat{\mathsf{Enc}}(eK, x)\right)\right) = \mathcal{C}(x).$$

Π *is* ***input projective*** *if the encoding set* eK *consists of two labels per input bit and encoding algorithm* $\widehat{\mathsf{Enc}}$ *selects for each input bit the corresponding label. Formally, for all* $x \in \mathrm{Dom}(C) \subseteq \{0,1\}^\ell$*, there exists a* ℓ*-bits string* $b \in \{0,1\}^\ell$ *such that*

$$eK = \begin{pmatrix} K_{10}, \cdots, K_{\ell 0} \\ K_{11}, \cdots, K_{\ell 1} \end{pmatrix} \quad \text{and} \quad \tilde{x} = (K_{1b_1}, \cdots K_{\ell b_\ell}).$$

Π *is* ***gate-by-gate garbled*** *if the garbling algorithm* $\widehat{\mathsf{Garble}}$ *garbles the circuit* $\mathcal{C}$ *by garbling separately each gate in topological order. Formally, let the circuit be represented as a sequence of gates* $\mathcal{C} = \{\mathcal{C}_1, \cdots, \mathcal{C}_{|\mathcal{C}|}\}$*,* $|\mathcal{C}|$ *denote the total number of gates, each* $\mathcal{C}_i$ *is a gate with two input wires and one output wire, and the gates are ordered topologically. The garbled circuit is then* $\widehat{\mathcal{C}} = \{\widehat{\mathcal{C}}_1, \cdots, \widehat{\mathcal{C}}_{|\mathcal{C}|}\}$*, where each* $\widehat{\mathcal{C}}_i$ *is the garbled version of* $\mathcal{C}_i$ *and generated sequentially. Each garbled gate* $\widehat{\mathcal{C}}_i$ *is represented by a* ***garbled table*** $T_{\widehat{\mathcal{C}}_i}$ *consisting of 4 entries, each corresponding to the evaluation of* $\mathcal{C}_i$ *under one of possible assignments to the input wires of* $\mathcal{C}_i$*.*

Π *supports* ***Point-and-Permute*** *technique [NPS99] if the evaluating algorithm* $\widehat{\mathsf{Eval}}$ *can determine which row of a garbled table to decrypt using a permutation bit associated with each wire label. Formally, let* $a, b \in \{0,1\}$ *be the input bits to a Boolean gate* $\mathcal{C}_i$*, and* W^a, W^b *denote the corresponding wire labels in the garbled gate* $\widehat{\mathcal{C}}_i$*. Each wire label* W *is associated with a permutation bit, typically defined as* $\mathrm{LSB}(W)$*, the least significant bit of* W*. The* $(2\mathrm{LSB}(W^a) + \mathrm{LSB}(W^b))$*-th entry in the garbled table* $T_{\widehat{\mathcal{C}}_i}$ *corresponds to the evaluation of the gate* $\mathcal{C}_i$ *with input* a, b*.*

We adopt the security properties of Garbling Schemes from [BHR12, ZRE15]. A Garbling Scheme Π is private if given the garbled circuit $\widehat{\mathcal{C}}$, garbled input $\tilde{x}$ and the decoding set dK, any efficient server (or adversary) executing Π learns no more about the circuit's input x than the output of the circuit $\mathcal{C}(x)$. The scheme Π is oblivious if any efficient eavesdropper (or adversary), given the garbled circuit $\widehat{\mathcal{C}}$ and the garbled input $\tilde{x}$, learns no information about the input x. The definition in the Appendix C.6 states it formally.

4 Balanced ARE: A New Computational Additive Randomized Encoding Scheme

In this section, we present our new ARE construction, which we call Balanced ARE. It is designed to distribute the computational load evenly across all clients. In Section 4.1, we define two building blocks for the construction: OT-ARE and Permute-Xor ARE (PX-ARE). We show that PX-ARE can be constructed from OT-ARE in a black-box manner, while for OT-ARE we rely on existing constructions [HIKR23, BEG25, BEGI25]. Then, in Section 4.2, we describe how Balanced ARE is constructed from these two building blocks.

4.1 Building Blocks

OT-ARE is a primitive that extends Oblivious Transfer (OT) to the ARE setting. In OT-ARE, one client holds a selection bit b, and another client holds two strings s_0 and s_1. The protocol allows the server to obtain exactly one string x_b, without learning which string was chosen and anything about the other string x_{1-b}. The OT-ARE scheme is formally presented in Theorem 3, and our Balanced ARE can be constructed with any existing constructions [HIKR23, BEG25, BEGI25]. For completeness, we provide one such concrete OT-ARE construction in Appendix D.

Definition 3 (Oblivious Transfer (OT) ARE Scheme)**.** *An additive randomized encodings 1-out-of-2 String Oblivious Transfer (ARE OT) scheme* $\Pi^{\mathsf{ot}}_{ARE} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ *is a secure ARE scheme for the two party function* $f_{\mathsf{ot}} : \{0,1\} \times \{0,1\}^{2\ell_A} \to \{0,1\}^{\ell_A}$ *defined as*

$$f_{\mathsf{ot}}((s_0, s_1), b) = \begin{cases} s_0 & \text{if } b = 0 \\ s_1 & \text{otherwise.} \end{cases}$$

To achieve a balanced computational load on the client side in our ARE construction, we define the primitive Permute-Xor ARE (PX-ARE). In PX-ARE, one client holds a pair of strings (s_0, s_1), while the other holds a permutation bit b and another pair of strings (s_0, s_1). The protocol allows the server to obtain the pair of XORs $(s_b \oplus s'_0, ; s_{1-b} \oplus s'_1)$, while without learning the permutation bit b and any additional information about the clients' inputs. The formal definition is given below.

Definition 4 (Permute-Xor ARE Scheme)**.** *A 1-out-of-2 Permute Xor ARE scheme* $\Pi^{\mathsf{pxt}}_{ARE} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ *is a secure ARE scheme for the two party function* $f_{\mathsf{pxt}} : \{0,1\}^{2\ell_A} \times \{0,1\}^{2\ell_A+1} \to \{0,1\}^{2\ell_A}$ *defined as*

$$f_{\mathsf{pxt}}((s_0, s_1), (b, (s'_0, s'_1))) = (s_b \oplus s'_0, s_{1-b} \oplus s'_1).$$

We propose a Permute-Xor (PX-ARE) scheme that can be constructed from any OT-ARE scheme, whose formal description is given in Algorithm 1. In the protocol, the ARE encoding for both clients consists of two messages. In the first message, the first client generates a random mask r, and the second client prepares the concatenation of its two input strings $(s'_0 \| s'_1)$. The second message is the ARE encoding of an OT-ARE instance. Concretely, the first client's input to this instance is the concatenation of its own input strings $(s_0 \| s_1)$, masked with r, and provided in both the original and reversed orders: $(s_0 \| s_1) \oplus r$ and $(s_1 \| s_0) \oplus r$. The second client's input is simply the permutation bit b.

During the decoding phase, the server receives two messages: the masked input from the second client $(s'_0 \| s'_1) \oplus r$ and the permuted masked input from the first client $(s_b \| s_{1-b}) \oplus r$. By XORing these two masked messages, the server computes the desired permute Xor messages $(s_b \oplus s'_0, s_{1-b} \oplus s'_1)$, without learning which inputs were paired and other information about two clients' input.

Theorem 5 states that the Permute Xor scheme presented in Algorithm 1 is correct and secure. We defer the simple but formal proof of the theorem to Appendix F.1.

Theorem 5 (Security of Algorithm 1)**.** *Algorithm 1 is a correct and secure Permute Xor ARE scheme.*

4.2 Balanced ARE

In this section, we present our Balanced ARE scheme (detailed in Algorithm 2), which can be constructed in a black-box manner using OT-ARE, Permute-Xor ARE primitives, and a garbling

Algorithm 1 Permute Xor ARE Scheme

Parameters:

- $\Pi^{\mathsf{ot}}_{\mathrm{ARE}}$: An ARE scheme for f_{ot}.
- The pairing parameter-generator (Theorem 23) $\mathcal{A}$

$\widetilde{\mathsf{Setup}}(1^\lambda, 1^n, 2^{2\ell_A})$

1: Let $\lambda' = \max(\lambda, \ell_A + 1)$, and $\mathsf{pp} = (q, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T, g_1, g_2, g_T, e) \leftarrow \mathcal{A}(1^{\lambda'})$.
2: Let $\mathsf{emb} : \{0,1\}^{\ell_A} \to [1, \frac{q-1}{2}]$ be injective (e.g., $\mathsf{emb}(s) = \mathsf{bin}(s) + 1$, where $\mathsf{bin}(\cdot)$ is the binary-to-integer conversion function).

Local Encoding Function $\widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i)$

Input: Party index $i \in \{0,1\}$: Input of party who's circuit is producing the wire: $x_0 = (b, (s'_0, s'_1)) \in \{0,1\} \times \{0,1\}^{\ell_A} \times \{0,1\}^{\ell_A}$, and the party who's circuit is consuming the wire: $x_1 = (s_0, s_1) \in \{0,1\}^{\ell_A} \times \{0,1\}^{\ell_A}$
Output: $\hat{x}_i \in \{0,1\}^{2\ell_A} \times (\mathcal{G}_1 \times \mathcal{G}_2 \times \mathcal{G}_T)$- encoding of input x_i.

1: **if** Producing circuit $(i = 0)$ **then**
2: $\quad \hat{x}^2_i \leftarrow \Pi^{\mathsf{ot}}_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i = 1, x_i = b)$
3: $\quad$ Send $\hat{x}_i = (s'_0 \| s'_1, \hat{x}^2_i)$
4: **else**
5: $\quad$ Choose at random $r \leftarrow \{0,1\}^{2\ell_A}$
6: $\quad$ Compute OT-ARE encoding

$$\hat{x}^2_i \leftarrow \Pi^{\mathsf{ot}}_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i = 0, x_i = ((s_0 \| s_1) \oplus r, (s_1 \| s_0) \oplus r))$$

7: **end if**

Decoding Function $\widetilde{\mathsf{Dec}}\,(\mathsf{pp}, \hat{y})$

Input: The sum of the encodings: $\hat{y} \in \{0,1\}^{2\ell_A} \times (\mathcal{G}_1 \times \mathcal{G}_2 \times \mathcal{G}_T)$
Output: An element $y = f_{\mathsf{pxt}}(x_0, x_1) \in \{0,1\}^{2\ell_A}$

1: Compute $y_1 \leftarrow \hat{y}_1, y_2 \leftarrow \Pi^{\mathsf{ot}}_{\mathrm{ARE}}.\widetilde{\mathsf{Dec}}(\mathsf{pp}, \hat{y}_2)$
2: Output $y \leftarrow y_1 \oplus y_2$

scheme. The Balanced ARE scheme serves as a general framework for computing arbitrary n-party input circuits $\mathcal{C}$ in the ARE model, while ensuring that both the computational and communication loads are evenly distributed among all clients.

At a high level, the Balanced ARE scheme partitions the to-compute circuit into equally sized subcircuits, with each client acting as the garbler for one subcircuit. In addition to sending its garbled subcircuit and decoding set to the server, each client also prepares ARE encodings that has two additional pieces of information for server's evaluation. First, the Balanced ARE's encoding has garbled labels for each input wire of the joint circuit, which are implemented using OT-ARE instances. Second, the Balanced ARE's encoding has wire-splicing gadget, which are implemented using permute-xor (PX) ARE instances to translate garbled labels consistently across subcircuits.

More concretely, in the encoding phase, each client generates a garbled circuit for its assigned subcircuit and prepares its encoding as follows. To prepare the ARE encodings for the input wires, for each of its own input bits, the client runs a OT-ARE instance as the receiver, with its input bit as the selection bit. For each input bit for which the corresponding input wire is in the client's subcircuit, it runs a OT-ARE instance as the sender, providing the two wire labels as the OT-ARE encodings.

To prepare the ARE encodings for wire-splicing gadgets, for each boundary wire w that is the input wire of the client's subcircuit, the client runs a PX-ARE instance as the first party, with its two wire labels for w as the PX-ARE encoding. For each boundary wire w that is the output wire of the client's subcircuit, the client runs a PX-ARE instance as the second party, with the hash of its two wire labels for w as the PX-ARE encoding. The permutation bit b is set to the least significant bit (LSB) of the output wire label corresponding to bit 0. Running these PX-ARE allows the server to learn the input wire label, with a mask derived from the output wire label for every boundary wires. This information allows it to translate garbled labels consistently across subcircuits without learning which bit the label represents and the alternative label.

In the decoding phase, the server receives all garbled subcircuits, as well as the garbled circuit's decoding set. It also receives encodings of the OT-ARE and Permute-Xor ARE employed in the encoding phase. The server then evaluates the joint garbled circuit as follows. It first extracts the garbled input labels from the outputs of the OT-AREs. It then evaluates the joint garbled circuit with these inputs, using the evaluation algorithm of the underlying garbling scheme. The only exception is in handling boundary wires. For a boundary wire w, the server first computes its input garbled label $\tilde{w}$ by evaluating the last-layer gate in the preceding subcircuit. It then obtains the second garbled label for w from the output of the corresponding Permute-Xor ARE instance. This is done by using $\tilde{w}$: the least significant bit of $\tilde{w}$ indicates which of the two messages contains the masked second key, and the value $H(\tilde{w})$ (the hash of $\tilde{w}$) is used to remove the mask and recover the output wire label for w. After evaluating all garbled subcircuits and reconstructing the full garbled output, the server applies the decoding function of the underlying garbling scheme using the decoding sets to recover and return the final output. The full algorithmic description is presented in Algorithm 2.

Theorem 6 states that the Balanced ARE is a correct and secure ARE scheme. We defer the formal proof of the theorem to Appendix F.2

Theorem 6 (Security of Balanced ARE)**.** *Let f be a n-party function with a boolean circuit representation. Assuming the underlying OT ARE scheme Π^{ot}_{ARE}, permute Xor ARE Π^{pxt}_{ARE}, and the garbling scheme Π_{Garble} are correct and secure, then Algorithm 2 is a correct and secure ARE scheme for f.*

Algorithm 2 Generic Balanced ARE Scheme Π_{gARE}

Parameters:

- $\mathcal{C}$: A Boolean circuit computing n-party function f; $H : \{0,1\}^{\ell} \mapsto \{0,1\}^{\ell}$: A random oracle.
- ϕ: A mapping from a wire w in the circuit $\mathcal{C}$ to an index of the encoding vector $\hat{x}$.
- Π_{Garble}: A garbling scheme (Theorem 2); $\Pi^{\mathsf{ot}}_{\mathrm{ARE}}$, $\Pi^{\mathsf{pxt}}_{\mathrm{ARE}}$: OT ARE scheme, 1-of-2 Permute Xor ARE Scheme

$\widetilde{\mathsf{Setup}}(1^{\lambda}, 1^{n}, 2^{\ell_A})$

Setup a balanced circuit partitioning $(\mathcal{C}_1, \cdots, \mathcal{C}_n)$ for circuit $\mathcal{C}$

Local Encoding Function $\widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i)$

Input: Party index $i \in \{0,1\}$; player i's input $x_i \in \{0,1\}^{\ell_A}$
Output: The encoding of input x_i: $(x'_i, \hat{x}_i)$.

Compute $(\widehat{\mathcal{C}}_i, eK, dK_i) \leftarrow \Pi_{\mathrm{Garble}}.\widehat{\mathsf{Garble}}(1^{\lambda}, \mathcal{C}_i)$
Set $x'_i \leftarrow (\widehat{\mathcal{C}}_i, dK_i)$ and initialize $\hat{x}_i$ as a zero vector over the group.
For each external input wire w corresponding to party i's input x_i:
Update $\hat{x}_i[\phi(w)] \leftarrow \hat{x}_i[\phi(w)] + \Pi^{\mathsf{ot}}_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i = 1, x_i = b)$, where b is the bit carried by the wire w.
For each external input wire w of $\mathcal{C}$ with respect to $\mathcal{C}_i$:
Update $\hat{x}_i[\phi(w)] \leftarrow \hat{x}_i[\phi(w)] + \Pi^{\mathsf{ot}}_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i = 0, x_i = (K_{w0}, K_{w1}))$, where $(K_{w0}, K_{w1}) \in eK$ is the wire labels for wire w.
For each boundary output wire w from the producing circuit of $\mathcal{C}$ with respect to $\mathcal{C}_i$:
  Compute $b \leftarrow \mathrm{LSB}(K_{w0})$, where (K_{w0}, K_{w1}) is the wire labels used in $\widehat{\mathcal{C}}_i$ for the wire w.
  Set $(K'_{w0}, K'_{w1}) \leftarrow (K_{w0}, K_{w1})$ if $b = 0$, else $(K'_{w0}, K'_{w1}) \leftarrow (K_{w1}, K_{w0})$.
  Update $\hat{x}_i[\phi(w)] \leftarrow \hat{x}_i[\phi(w)] + \Pi^{\mathsf{pxt}}_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i = 1, x_i = (b, H(K'_{w0}), H(K'_{w1})))$
For each boundary input wire w from the consuming circuit of $\mathcal{C}$ with respect to $\mathcal{C}_i$:
  Update $\hat{x}_i[\phi(w)] \leftarrow \hat{x}_i[\phi(w)] + \Pi^{\mathsf{pxt}}_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i = 0, x_i = (K_{w0}, K_{w1}))$, where (K_{w0}, K_{w1}) is the wire labels used in $\widehat{\mathcal{C}}_i$ for the wire w.

Decoding Function $\widetilde{\mathsf{Dec}}(\mathsf{pp}, \hat{y})$

Input: The sum of the encodings: $\hat{y}$; and the encoding $y' = (x'_1, \cdots, x'_n)$
Output: $y = f(x_1, \cdots, x_n)$, function f's output over $x_1, \cdots, x_n$

For each gate $C \in \mathcal{C}$ in the topological order:
  Denote by u, v the gate C's two input wires and w its output wire
  If C is a first-layer gate of $\mathcal{C}$:
    Compute the garbled key for u, v: $(\tilde{u}, \tilde{v}) \leftarrow \left(\Pi^{\mathsf{ot}}_{\mathrm{ARE}}.\widetilde{\mathsf{Dec}}(\hat{y}[\phi(u)]), \Pi^{\mathsf{ot}}_{\mathrm{ARE}}.\widetilde{\mathsf{Dec}}(\hat{y}[\phi(v)])\right)$
  ElseIf C is a first-layer gate of some circuits part $\mathcal{C}_i$:
    Update the garbled key $\tilde{u}$ for u: $\tilde{u} \leftarrow H(\tilde{u}) \oplus \Pi^{\mathsf{pxt}}_{\mathrm{ARE}}.\widetilde{\mathsf{Dec}}(\hat{y}[\phi(u)])[\mathrm{LSB}(\tilde{u})]$
    Update the garbled key $\tilde{v}$ for v: $\tilde{v} \leftarrow H(\tilde{v}) \oplus \Pi^{\mathsf{pxt}}_{\mathrm{ARE}}.\widetilde{\mathsf{Dec}}(\hat{y}[\phi(v)])[\mathrm{LSB}(\tilde{v})]$
  Compute the garbled key for w: $\tilde{w} \leftarrow \Pi_{\mathrm{Garble}}.\widehat{\mathsf{Eval}}(\widehat{\mathcal{C}}, (\tilde{u}, \tilde{v}))$, where $\widehat{\mathcal{C}} \in y'$ is a garbled circuit part for C
Compute $y \leftarrow \Pi_{\mathrm{Garble}}.\widehat{\mathsf{Dec}}(dK, \mathsf{key})$, where key denotes the garbled keys of the external output wires for $\mathcal{C}$, and $dK \in y'$.

5 A Compiler from Central DP to Shuffle DP via Balanced ARE

In this section, we present a general compiler (detailed in Algorithm 3) that takes as input an arbitrary central differential privacy (DP) mechanism and outputs a computational shuffle DP protocol. The shuffle DP protocol is constructed from our Balanced ARE and inherits the desirable property that all clients have even computation workloads. Moreover, it takes advantage of the sparsity of balanced ARE encodings: the size of the shuffle DP encoding depends only on the number of non-zero entries in the ARE encodings, rather than its entire length.

We start by formalizing the sparsity property of an ARE scheme's encoding. We say that an ARE scheme is called an (k, t, b)-ARE scheme if its client's ARE encoding $\hat{x}_i$ can be represented as a length-k group element vector, where each component is representable using at most b bits and at most t components are non-zero. It is formally given at below.

Definition 7 (ARE Scheme with Sparse Encodings)**.** *Let $k, t, b, n \in \mathbb{N}$ and let each $\mathcal{G}_i$ for $i \in [k]$ be a abelian additive group consist with group size at most c. Let $\Pi_{ARE} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ be an ARE scheme for a n-party function f, and let $(x'_i, \hat{x}_i)$ be the output of the encoding function $\widetilde{\mathsf{Enc}}$ for client i, where $i \in [n]$. We say that Π_{ARE} is a (k, t, b)-ARE scheme if, for every $i \in [n]$, the structured encoding satisfies $\hat{x}_i \in \mathcal{G}_1 \times \cdots \times \mathcal{G}_k$ and contains at most t non-zero elements, each representation using at most b bits.*

The Balanced ARE scheme fits into the definition as a (k, t, b)-ARE scheme. Here, k is the total number of input bits of the to-compute circuit plus twice the number of boundary wires. t is the maximum, over all clients, of three values combined: the number of input bits the client contributes to the circuit, the number of global input wires connecting to its subcircuit, and twice the number of boundary wires connected to that subcircuit. b is twice the size of the ARE encoding used in the underlying OT-ARE scheme.

Our proposed compiler, given in Algorithm 3, takes as input an arbitrary central DP mechanism and outputs a computational shuffle DP protocol using any (k, t, b)-ARE scheme. At a high level, the compiler first compiles the central DP mechanism into a computational ARE protocol, and then encodes the resulting ARE encodings into shuffle DP encodings by applying the Split-and-Mix techniques from [IKOS06]. To reduce the size of Shuffle DP encodings, we only apply Split-and-Mix to the non-zero components of the ARE encodings. Each non-zero component is tagged with its position in the encoding vector before being secret-shared, so that after shuffling the analyzer can correctly collect the shares for each coordinate and decode them. We note that this optimization does not compromise privacy: the positions of non-zero entries are determined by the public circuit partitioning and are therefore public.

Theorem 8 states that the general compiler yields a (ε, δ) -computationally differentially private protocol. We defer the formal proof of the theorem to Appendix G.1.

Theorem 8 (Computationally Differential Privacy of Algorithm 3)**.** *Let f be a (ε, δ)-differentially private mechanism. Let Π_{ARE} be a correct and secure ARE scheme for f and Π_{Sum} a secure sum shuffle protocol. Then Algorithm 3 is (ε, δ)-computationally differentially private.*

Bandwidth Improvements via Sparsity. We compare client-side bandwidth to highlight the improvements of Algorithm 3 over a naive compiler that directly converts an ARE protocol into a shuffle protocol using the Split-and-Mix technique. In the naive approach, each client's bandwidth is $O(|C|b\lambda)$, scaling with the full ARE encoding size $|C|$. In contrast, the compiler Π_{oSHARE} reduces this cost to $O(tb\lambda)$, where the dependence is only on the (typically much smaller) sparsity parameter t. A detailed description of the naive compiler can be found in Algorithm 9, Appendix G.2.

Algorithm 3 Optimized ARE-based Shuffle Protocol (Π_{oSHARE})

Parameters:

- $f : \left(\{0,1\}^{\ell}\right)^n \mapsto \{0,1\}^*$: An n-party (randomized) function where each party holds an ℓ bits input.
- $\Pi_{\text{ARE}} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$: A (k,t,b)-ARE scheme for f (Theorem 7).
- ℓ_A: Input length of the encoding function $\widetilde{\mathsf{Enc}}$.
- $\Pi_{\text{Sum}} = (\mathsf{Enc}_{\text{Sum}}, \mathsf{Dec}_{\text{Sum}})$: The shuffle Sum protocol (shown in Algorithm 4).
- m: The number of output messages from the encoding function $\mathsf{Enc}_{\text{Sum}}$.

Compute the setup $\mathsf{pp} \leftarrow \Pi_{\text{ARE}}.\widetilde{\mathsf{Setup}}(1^\lambda, 1^n, 1^{\ell_A})$

Local Encoding Function $\mathsf{Enc}(i, x_i)$

Input: Index $i \in [n]$, an element $x_i \in \mathcal{G}$, represented by an ℓ-bits string.
Output: $(x_i', \hat{x}_i)$, the encoding of input x_i.

Compute $(x_i', \hat{z}_i) \leftarrow \Pi_{\text{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i)$, where $\hat{z}_i = (\hat{z}_i^1, \cdots, \hat{z}_i^k)$.
Initialize $\hat{x}_i$ as an empty set.
Compute the index set $\mathcal{I}$ containing the index of all non-zero element in $\hat{z}_i$.
for $j \in \mathcal{I}$ **do**
    Compute $\mathcal{S}_j \leftarrow \Pi_{\text{Sum}}.\mathsf{Enc}_{\text{Sum}}(i, \hat{z}_i^j)$
    For every element $g \in \mathcal{S}_j$, update $\hat{x}_i \leftarrow \hat{x}_i \bigcup \{j \| g\}$
end for
Output $(x_i', \hat{x}_i)$

Decoding Function $\mathsf{Dec}(x_1', \cdots, x_n', \hat{y}_1, \cdots \hat{y}_{mnt})$

Input: The shuffled encodings $\hat{y}_i^1, \ldots, \hat{y}_i^{nmt}$ from n clients, along with the encoding x_i' from client i.
Output: An element $y \leftarrow f(x_1, \cdots, x_n)$

Initialize $\hat{y} \leftarrow 0^k$ as a k-length zero element vector.
for $j \in \{1, 2, \cdots, k\}$ **do**
    Set $\mathcal{S}_j$ as the set of encodings from $\{\hat{y}_i^1, \ldots, \hat{y}_i^{nmt}\}$ with prefix j
    Compute $\hat{y}_j \leftarrow \mathsf{Dec}_{\text{Sum}}(\mathcal{S}_j)$
end for
Compute $y \leftarrow \widetilde{\mathsf{Dec}}\,(\mathsf{pp}, x_1', \cdots, x_n', \hat{y})$
Output y.

5.1 Bandwidth Bounds in Shuffle Protocols

The client bandwidth of our shuffle protocols depends on the circuit partition used in the Balanced ARE construction. In particular, each client's bandwidth is governed by the number of input wires it contributes and the number of wires crossing the boundary of its assigned subcircuit. For the purpose of bandwidth analysis and protocol implementation, we discuss the circuit-partitioning problem in this section.

Theorem 9 defines a (γ, n, t)-circuit partitioning of $\mathcal{C}$ as a decomposition of $\mathcal{C}$ into n disjoint subcircuits, each containing at most a $(1+\gamma)/n$ fraction of the gates, such that each subcircuit has at most t boundary-crossing wires.

Definition 9 ((γ, n, t)-Circuit Partitioning)**.** *Let $\gamma > 0$ and $n, t \in \mathbb{N}$. A Boolean circuit $\mathcal{C}$ admits a (γ, n, t)-partition if there exists a collection of disjoint gate subsets $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ such that $\bigcup_{i\in[n]} \mathcal{C}_i = \mathcal{C}$, $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for all $i \neq j$, and*

$$|\mathcal{C}_i| \leq (1+\gamma)\frac{|\mathcal{C}|}{n} \quad \text{and} \quad \max_{i\in[n]} \mu(\mathcal{C}_i) \leq t.$$

Here $\mu(\mathcal{C}_i)$ denotes the number of wires with exactly one endpoint in $\mathcal{C}_i$ and the other endpoint in $\mathcal{C} \setminus \mathcal{C}_i$.

In our compiler, XOR and NOT gates are computed by the clients generally for free, while non-XOR gates dominate the client-side garbling work. For this reason, our circuit splitting strategies balance the number of non-XOR gates and optimize the boundary wires that trigger PXT-ARE operations. Specifically, let V be the gate set of $\mathcal{C}$, and let $E \subseteq V \times V$ denote its pin-level dependency edges, where $(u \to v) \in E$ represents a wire from the output of gate u to an input pin of gate v. Define

$$w(v) = \mathbf{1}[v \text{ is a non-XOR gate}], \qquad W_{\text{tot}} = \sum_{v\in V} w(v).$$

For a partition $\pi : V \to [n]$, let $V_i = \{v \in V : \pi(v) = i\}$ denote the gates assigned to client i. The main partitioning objective used in our experiments is the following min-max boundary problem.

Problem 1 (Balanced non-XOR min-max incoming boundary partitioning)**.** *Let $\mathcal{C}$ have gate set V and pin-level dependency edges $E \subseteq V \times V$. Let $w : V \to \{0, 1\}$ be the non-XOR indicator weight, $w(v) = \mathbf{1}[v \text{ is non-XOR}]$, and let $W_{\text{tot}} = \sum_{v\in V} w(v)$. Given parameters $n \in \mathbb{N}$ and $\gamma > 0$, find a partition $\pi : V \to [n]$ minimizing the maximum incoming boundary:*

$$\min_{\pi:V\to[n]} \quad \max_{i\in[n]} \ \mu_{\text{in}}(V_i)$$
$$s.t. \quad \sum_{v\in V_i} w(v) \leq (1+\gamma)\frac{W_{\text{tot}}}{n} \qquad \forall i \in [n],$$

where $V_i = \{v \in V : \pi(v) = i\}$ and

$$\mu_{\text{in}}(V_i) = \big|\{(u \to v) \in E : \ u \notin V_i, \ v \in V_i\}\big|.$$

Besides the purpose of constructing concrete protocols, Theorem 10 states the upper bound on the client-side bandwidth of our shuffle protocols. In the general case, with no assumption made on $\mathcal{C}$, the client bandwidth is $O(\frac{|\mathcal{C}|}{n})$. If $\mathcal{C}$ is leveled with width t, meaning that its gates can be grouped into layers so that wires only stay within a layer or go to the next layer, then the client bandwidth improves to $O(t)$.

Task	Protocol	Client Computation	Client Bandwidth
Selection	Baseline	$O(dn)$	$O(dn)$
	Ours	$O(d)$	$O(d)$
Distinct Elements	Baseline	$O(n \log^2 n \log \lvert\mathcal{X}\rvert)$	$O(n \log \lvert\mathcal{X}\rvert)$
	Ours	$O(\log^2 n \log \lvert\mathcal{X}\rvert)$	$O(\log^2 n \log \lvert\mathcal{X}\rvert)$
SGD	Baseline	$O(n \lvert\theta\rvert)$	$O(n)$
	Ours	$O(\lvert\theta\rvert)$	$O(\lvert\theta\rvert)$
Linear Contextual Bandits	Baseline	$O(nTd^2)$	$O(Td^2)$
	Ours	$O(Td^2)$	$O(Td^2)$

Table 2: Client-side overhead of our shuffle CDP compiler compared with shuffle protocols obtained from standard ARE plus Split-and-Mix.

Theorem 10 (Upper Bounds for Bandwidth in Shuffle Protocols)**.** *Let f be a multi-input function that admits a Boolean circuit $\mathcal{C}$. For an arbitrary $\mathcal{C}$, the shuffle protocol Π_{oSHARE} for f has per-client bandwidth $O(|\mathcal{C}|/n)$, and this bound is tight in general. If $\mathcal{C}$ is a leveled Boolean circuit with width t, then the bandwidth for each client is $O(t)$.*

We defer the additional partitioning objectives, formal definitions, and proofs to Appendix E.

6 Experimental Evaluation

In this section, we evaluate the client-side computation and communication costs of our generic shuffle CDP compiler on representative differentially private computations, including Gaussian Sum, Selection, Distinct Elements, and Linear Contextual Bandits. We report the asymptotic per-client computation and communication costs of the compiled protocols, together with the an end-to-end proof-of-concept (PoC) implementation. As a baseline, we also implement the unbalanced ARE protocol of Halevi et al. [HIKR23]. We conducted extensive experiments varying circuit sizes, number of clients, and input dimensions to evaluate the trade-offs between computational and communication efficiency in both settings. The formal task definitions, the central mechanisms being compiled, the noise-sampling details, and the inherited utility guarantees are deferred to Appendix H.1.

Asympotic client-side overhead Table 2 summarizes the per-client asymptotic overhead of our compiled protocols and compares it with the baseline approach derived from standard ARE [HIKR23, BEG25, BEGI25] combined with Split-and-Mix techniques [IKOS06]. Table 2 shows that our protocols impose modest client-side costs across tasks. In particular, for selection problem, both computation and bandwidth improve over the baseline $O(dn)$, which scales linearly in n, to $O(d)$. For Distinct Element problem, the client-side overhead improves from $O(n \log n)$ to $O(\log n)$. We implement the computation via an oblivious–sorting–based circuit, and our compiler ensures that each client's load keep logarithmic in n.

6.1 Implementation details

Our protocols are implemented in C++17. We utilize the BN254 curve and implement pairing-based operations (G_1, G_2, G_T, and $\mathbb{F}_r$ field arithmetic) using Herumi's `mcl` library [Mit15]. For networking, half-gates garbling, and circuit evaluation, we leverage the `emp-toolkit` [WMK16]. Numerical visualizations are generated using Python (`pandas` and `matplotlib`).

We implement the three ARE primitives required by the protocol: Rabin-OT-ARE, String-OT-ARE, and Permute-XOR-ARE (PXT-ARE), following the constructions in Appendix D. Decoding for Rabin-OT-ARE requires inverting the discrete logarithm of g_T^s. To optimize this, we precompute and persist a binary lookup table mapping serialized g_T^s elements to their corresponding scalars s. These tables are serialized in a sorted binary layout and loaded via `mmap` for efficiency. We store two table sizes: ℓ_A=12 ($\approx$ 1.6 MiB, used by the evaluator for input wires) and ℓ_A=20 ($\approx$ 406 MiB, used by clients for boundary wires). The code for our implementation is available at: https://github.com/stoneboat/shuffle-DP-protocols.

6.2 Experimental setup

Network protocol: Each experiment consists of one evaluator and n client processes communicating over TCP via `emp-toolkit`'s `NetIO`. The evaluator processes clients sequentially: for each client C_i, it mediates the transfer of boundary hashes from upstream clients, receives PXT-ARE replies, and collects C_i's garbled gates, output labels, and OT-ARE encodings before proceeding to C_{i+1}. Boundary transfers between clients are mediated by the evaluator. In a real world implementation, there should be a secure shuffler who co-ordinates input and boundary transfers for an added layer of privacy. For the unbalanced ARE protocol baseline of [HIKR23], a single client garbles the entire circuit and executes OT-ARE for every other client's input bits.

We ran all our experiments on a shared HPC cluster, where each node has two Intel Xeon Gold 6226 processors providing 24 physical cores at 2.7 GHz and 600 GB of RAM.

6.3 Circuit splitting strategies

We evaluated three strategies for partitioning a Bristol-format circuit across N clients, formally defined in §5.1.

- **Topologically Balanced:** Each client is assigned an equal-sized contiguous range of the topologically sorted gate list (i.e., client c owns gates $[c \cdot |G|/N, (c+1) \cdot |G|/N)$). All gates are treated equally, including cost-free XOR and NOT gates. This requires no structural analysis of the circuit.
- **Non-XOR Balanced:** Only AND gates contribute to the cost, as XOR/NOT gates are free under half-gates garbling. Cut points are selected such that each client is assigned an equal number of AND gates using a prefix-sum count over the gate list.
- **Min-max Boundary:** This implements Problem 1, aiming to minimize the maximum number of incoming boundary wires μ_{in} across all clients. The implementation initializes the partition using the better of the Min-Cut or Topologically Balanced splits and refines it via coordinate descent. We apply a greedy local search within a feasibility window for each cut.

As shown in Figure 2, both communication and computation costs are dominated by boundary wire transfers. This is expected, as the protocol is sequential; the total wall-clock time is governed by the client receiving the most boundary wires, where PXT-ARE complexity is $O(|\text{boundary_in}|)$.

Figure 3 illustrates $\max_c |\text{boundary_in}[c]|$ for $N \in \{8, \ldots, 128\}$. The Min-max strategy outperforms the alternatives, showing a slight improvement over the topological split, while Non-XOR balancing performs poorly regarding boundary counts. Consequently, we utilize the **Min-max boundary split** for all subsequent scaling experiments.

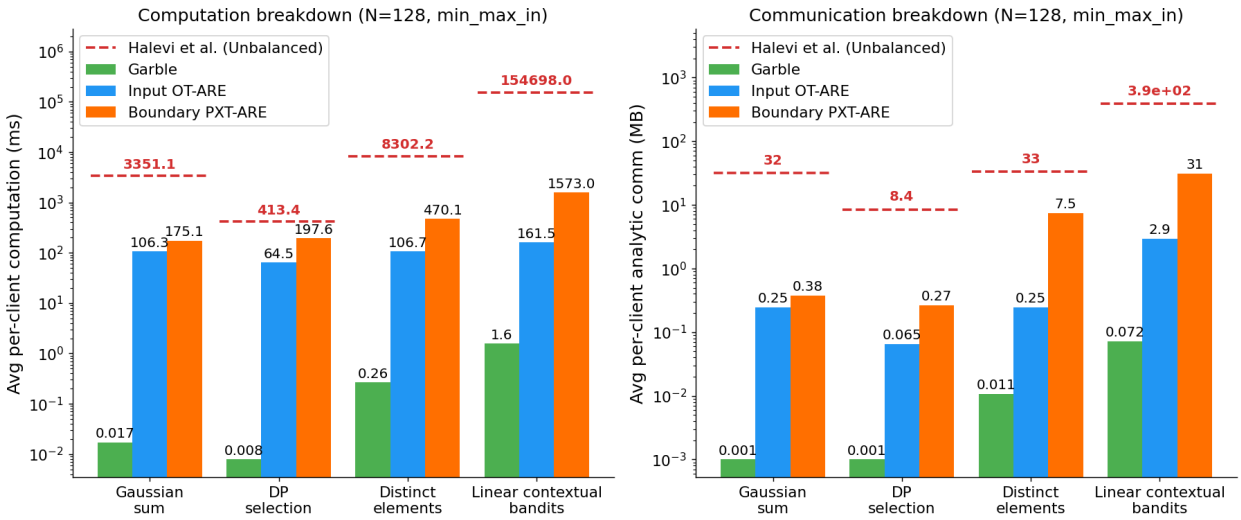

Figure 2: Average client cost breakdown for garbling, input transfer and boundary wire transfers for N=128, Partitioning = Min Max.

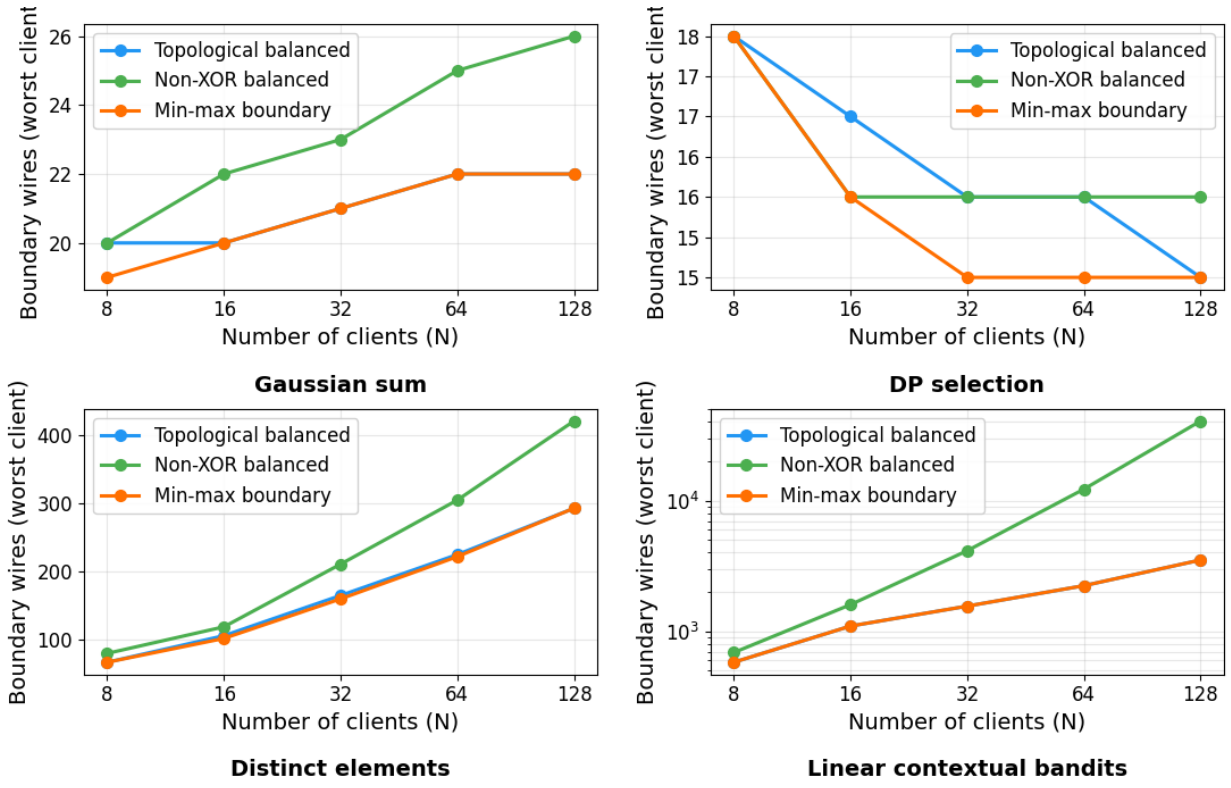

Figure 3: Worst case input boundary wire count for different circuit Splitting Techniques

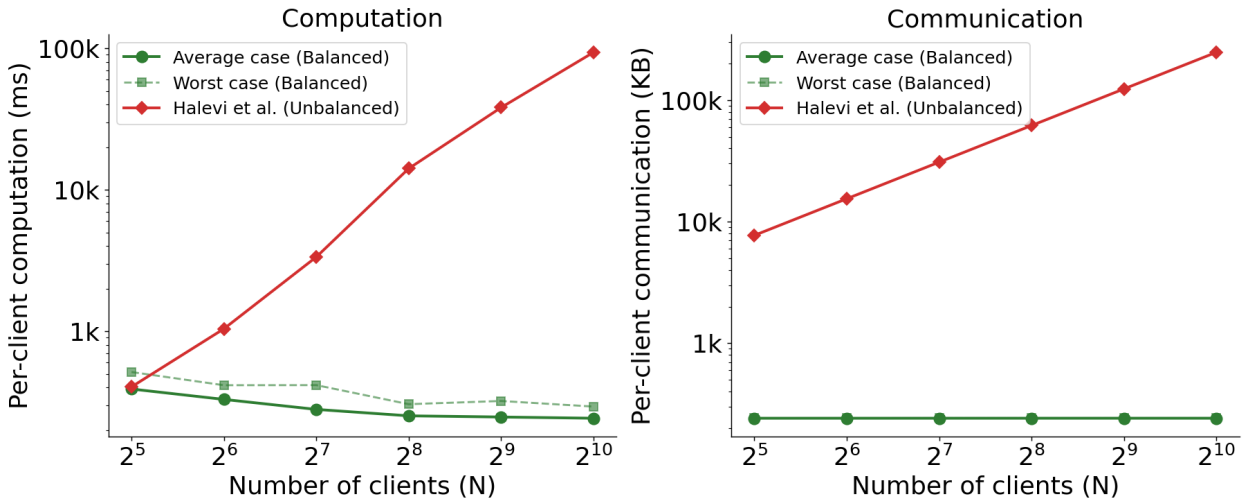

Figure 4: Per-client computation and communication costs for $N = 2^{10}$ clients for Gaussian Sum.

6.4 Client-side efficiency comparision

We analyzed per-client costs for four DP functions. For each, noise is sampled in plaintext (either locally or by a designated party) and bit-decomposed. Noise addition is implemented as a ripple-carry adder (for Sum and Distinct Elements) or a series of adders for the tree mechanism. All experiments use ε=1.

Differentially-Private Sum (Gaussian): We implement a balanced binary tree-sum circuit with added discrete gaussian noise. Each client i adds Gaussian noise to its K-bit private value x_i in plaintext:

$$\tilde{x}_i = [\lfloor x_i + z_i \rceil]_0^{2^K-1} \quad z_i \sim \mathcal{N}\left(0, \frac{\sigma^2}{N}\right)$$

where $\sigma = \frac{(2^K-1)\sqrt{2\ln(1.25N^2)}}{\varepsilon}$ and $\delta = \frac{1}{N^2}$.

This could be extended to applications like DP-SGD where clipped and noised gradients are aggregated in each step.

Selection Problem: The selection problem is implemented as a leveled circuit that performs a tree-sum of the one-hot vectors held by each client. We add shifted and quantized Gumbel noise to each aggregated value and identify the index of the noisy maximum using a tournament-style argmax, which performs pairwise comparisons. This tree structure ensures a circuit depth of $\log N$ and maintains clean boundary cuts. The Gumbel noise g_j, which is standard for selection tasks, is defined as:

$$g_j = \frac{-\ln(-\ln U_j)}{\varepsilon} \quad U_j \sim \text{Uniform}\{0,1\}, \quad j \in [D]$$

Distinct elements: We address the distinct elements problem using a bitonic sorting network followed by the Laplace mechanism. Since the counting operation has an L_1 sensitivity of 1, a single Laplace sample with scale $1/\varepsilon$ suffices for ε-DP. We generate the noise sample as:

$$z = u \cdot E, \quad u \sim \text{Uniform}\{-1,1\}, \quad E \sim \text{Exp}(\varepsilon),$$

which follows the $\text{Lap}(0, 1/\varepsilon)$ distribution. This sample is clamped to a signed $\lceil \log |\mathcal{X}| \rceil$-bit range and encoded as a two's complement value across the dedicated noise input wires. The bitonic sorting network recursively constructs and merges bitonic sequences to produce a sorted array.

Linear Contextual Bandits: We implement a Fenwick tree-based mechanism with Laplace noise, a standard building block for differential privacy in streaming data. The Fenwick tree allows us to

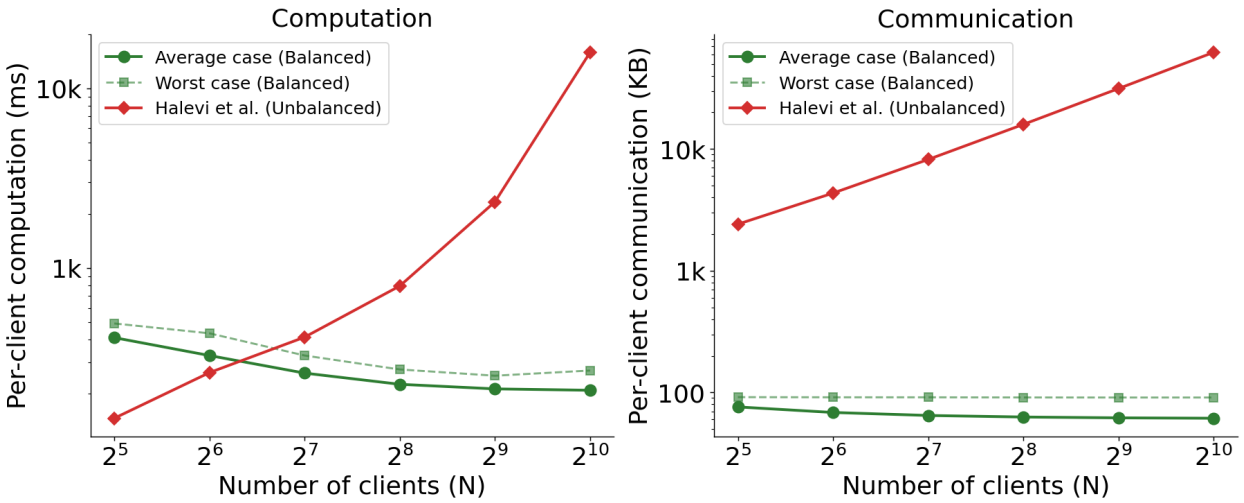

Figure 5: Per-client computation and communication costs for $N = 2^{10}$ clients for Selection Problem.

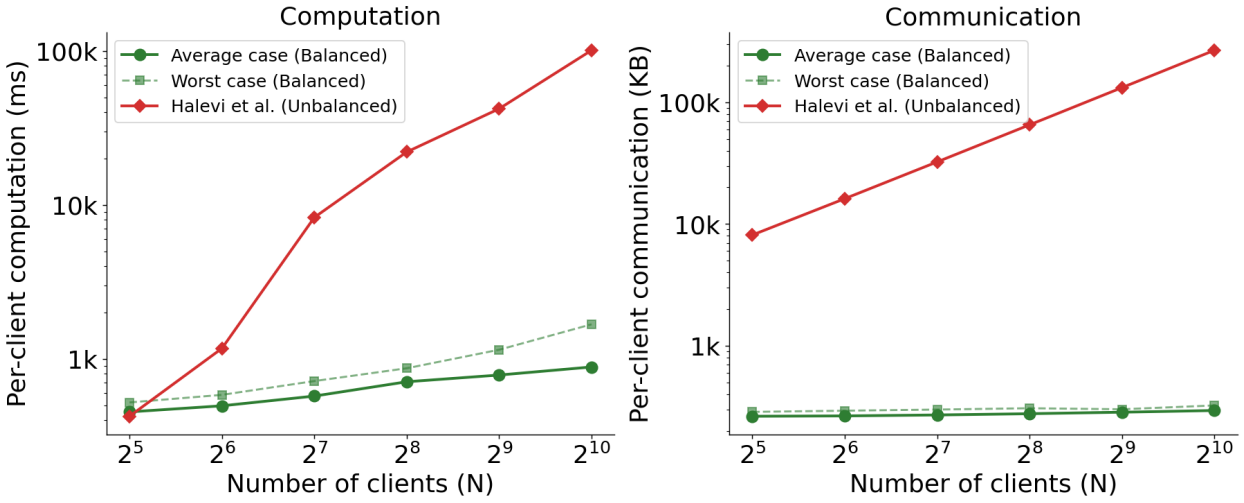

Figure 6: Per-client computation and communication costs for $N = 2^{10}$ clients for Distinct Elements.

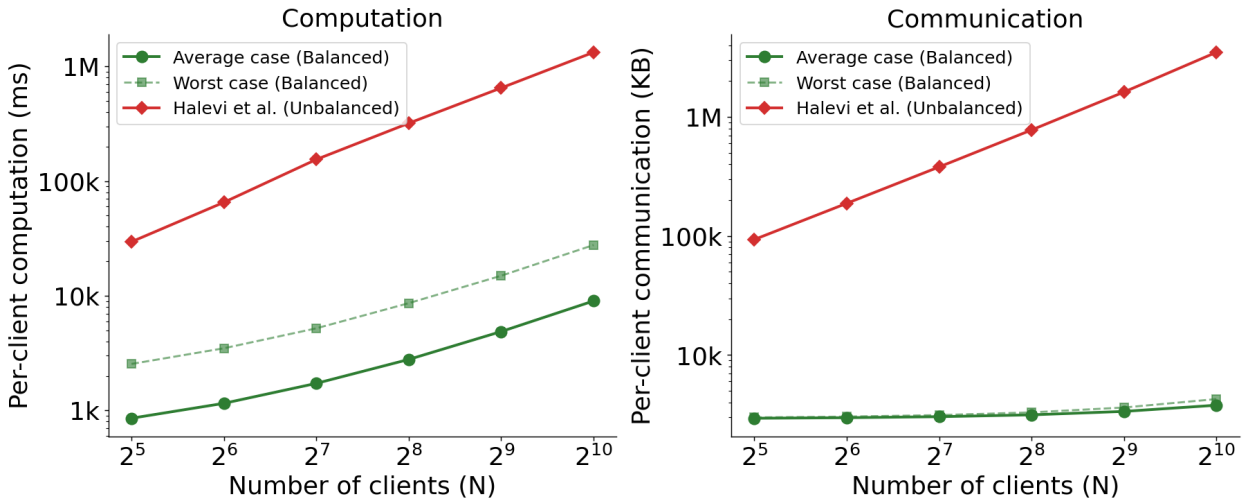

Figure 7: Per-client computation and communication costs for $N = 2^{10}$ clients for Linear Contextual Bandits.

amortize the noise by adding it once per node rather than per query. Independent Laplace noise is added to each Fenwick node $i \in [1, T]$ and each dimension $k \in [d + d^2]$:

$$\eta_{i,k} \sim \mathrm{Lap}\left(0, \frac{\log T}{\varepsilon}\right)$$

A prefix sum query aggregates at most $\lceil \log_2(t+1) \rceil$ nodes, ensuring ε-DP for the sequence.

Our balanced protocol exhibits near-constant per-client overhead as the number of clients N scales to 2^{10}. As illustrated in Figures 4–7, the balanced setting achieves a reduction in both computation and communication costs of up to two orders of magnitude compared to the unbalanced baseline, demonstrating the effectiveness of our circuit-partitioning strategies in real-world, large-scale DP applications.

7 Conclusion

We presented a client-efficient, one-round compiler from central differential privacy to non-robust computational shuffle DP that matches central-model privacy/utility tradeoff on arbitrary mechanisms. The core technical contribution in our work is a new *balanced additive randomized encoding* scheme, that allows for evenly distributing the garbling cost of mechanism $\mathcal{M}$ across n clients: dropping worst-case per client garbling work from $O(|\mathcal{M}|)$ in prior works to $O(|\mathcal{M}|/n)$. The sparse encoding of our balanced ARE scheme also helps reduce the asymptotic complexity of client bandwidth. Our end-to-end implementation on concrete DP tasks like selection, distinct element and linear contextual bandits realizes these asymptotic gains in practice, with up to **386**× computation and **402**× bandwidth speedups over prior work for $n = 2^{10}$ clients.

Our results are for the non-robust shuffle model. We leave it as future work to extend our framework to the *robust shuffle* DP setting (that allows for malicious clients and a malicious evaluator). We believe it might be possible to leverage a stronger ARE variant called *robust additive randomized encoding* [HIKR23, BF24]. The key technical challenge of this setting is, however, that the adversary learns not just the final output of the secure computation, but it also learns the ideal residual function on honest party inputs when using robust ARE. Hence, a malicious adversary can "try all possible malicious party inputs," weakening the DP guarantee provided by the entire scheme.

Acknowledgements

The authors would like to thank Yuval Ishai for insightful discussions and suggestions during the early stages of the project. Adya Agrawal was supported in part by NSF Awards No. 2448339 and No. 2531010. Yu Wei was supported in part by NSF Award No. 2448339, Halcyon Futures via the AI Security Institute, and Sunday Group, Inc. Jaspal Singh was supported in part by AnalytiXIN and by Sunday Group, Inc. Vassilis Zikas was supported in part by NSF Awards No. 2448339 and No. 2531010, JPMorgan Chase, Halcyon Futures via the AI Security Institute, the Stellar Development Foundation, and Sunday Group, Inc. This research was supported in part through research cyberinfrastructure resources and services provided by the Partnership for an Advanced Computing Environment (PACE) at the Georgia Institute of Technology, Atlanta, Georgia, USA. Work done in part while Vassilis Zikas and Jaspal Singh were at the Simons Institute for the Theory of Computing, UC Berkeley.

References

[ACG+16] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 308–318, New York, NY, USA, 2016. ACM.

[AKC+22] Mohammed Adnan, Shivam Kalra, Jesse C Cresswell, Graham W Taylor, and Hamid R Tizhoosh. Federated learning and differential privacy for medical image analysis. *Scientific Reports*, 12(1):1953, 2022.

[App25] Apple Inc. Differential privacy overview. https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf, 2025.

[BBGN19] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. The privacy blanket of the shuffle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 638–667, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[BBGN20] Borja Balle, James Bell, Adria Gascon, and Kobbi Nissim. Private summation in the multi-message shuffle model. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 657–676, New York, NY, USA, 2020. ACM.

[BC20] Victor Balcer and Albert Cheu. Separating local & shuffled differential privacy via histograms. In Yael Tauman Kalai, Adam D. Smith, and Daniel Wichs, editors, *ITC 2020*, volume 163 of *LIPIcs*, pages 1:1–1:14, Boston, MA, USA, June 17–19, 2020. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[BEG25] Nir Bitansky, Saroja Erabelli, and Rachit Garg. Additive randomized encodings from public key encryption. In *Advances in Cryptology – CRYPTO 2025 – 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2025, Proceedings, Part IV*, volume 16003 of *Lecture Notes in Computer Science*, pages 241–260, Cham, Switzerland, 2025. Springer.

[BEGI25] Nir Bitansky, Saroja Erabelli, Rachit Garg, and Yuval Ishai. Shuffling is universal: Statistical additive randomized encodings for all functions. Cryptology ePrint Archive, Paper 2025/1442, 2025.

[BEM+17] Andrea Bittau, Ulfar 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*, pages 441–459, New York, NY, USA, 2017. ACM.

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[BF24] Nir Bitansky and Sapir Freizeit. Robust additive randomized encodings from IO and pseudo-non-linear codes. In *Advances in Cryptology – CRYPTO 2024 – 44th Annual*

International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part VIII, volume 14927 of *Lecture Notes in Computer Science*, pages 109–135, Cham, Switzerland, 2024. Springer.

[BHNS20] Amos Beimel, Iftach Haitner, Kobbi Nissim, and Uri Stemmer. On the round complexity of the shuffle model. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 683–712, Durham, NC, USA, November 16–19, 2020. Springer, Cham, Switzerland.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796, New York, NY, USA, 2012. ACM.

[CGB17] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 259–282, Boston, MA, USA, 2017. USENIX Association.

[CGDS+19] Olivia Choudhury, Aris Gkoulalas-Divanis, Theodoros Salonidis, Issa Sylla, Yoonyoung Park, Grace Hsu, and Amar Das. Differential privacy-enabled federated learning for sensitive health data. arXiv preprint arXiv:1910.02578, 2019.

[CGKM21] Lijie Chen, Badih Ghazi, Ravi Kumar, and Pasin Manurangsi. On distributed differential privacy and counting distinct elements. In James R. Lee, editor, *ITCS 2021*, volume 185, pages 56:1–56:18, Virtual Conference, January 6–8, 2021. LIPIcs.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[CSU+19a] Albert Cheu, Adam D. Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 375–403, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[CSU+19b] Albert Cheu, Adam D. Smith, Jonathan R. Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 375–403, Cham, Switzerland, 2019. Springer.

[CU21] Albert Cheu and Jonathan R. Ullman. The limits of pan privacy and shuffle privacy for learning and estimation. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 1081–1094, Virtual Event, Italy, June 21–25, 2021. ACM Press.

[CZ22a] Albert Cheu and Maxim Zhilyaev. Differentially private histograms in the shuffle model from fake users. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 440–457, Piscataway, NJ, USA, 2022. IEEE.

[CZ22b] Sayak Ray Chowdhury and Xingyu Zhou. Shuffle private linear contextual bandits. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 3984–4009, Baltimore, MD, USA, 2022. PMLR.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 265–284, New York, NY, USA, March 4–7, 2006. Springer Berlin Heidelberg, Germany.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, Berkeley, CA, USA, 2004. USENIX Association.

[DP20] Damien Desfontaines and Balázs Pejó. SoK: Differential privacies. *PoPETs*, 2020(2):288–313, April 2020.

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[Dwo06] Cynthia Dwork. Differential privacy (invited paper). In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 1–12, Venice, Italy, July 10–14, 2006. Springer Berlin Heidelberg, Germany.

[EFM+19] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In Timothy M. Chan, editor, *30th SODA*, pages 2468–2479, San Diego, CA, USA, January 6–9, 2019. ACM-SIAM.

[EPK14] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 1054–1067, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[GIK+24] Adrià Gascón, Yuval Ishai, Mahimna Kelkar, Baiyu Li, Yiping Ma, and Mariana Raykova. Computationally secure aggregation and private information retrieval in the shuffle model. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 4122–4136, New York, NY, USA, 2024. ACM.

[HIKR23] Shai Halevi, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. Additive randomized encodings and their applications. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 203–235, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[IKLM24] Yuval Ishai, Mahimna Kelkar, Daniel Lee, and Yiping Ma. Information-theoretic single-server pir in the shuffle model. In *5th Conference on Information-Theoretic Cryptography (ITC 2024)*, volume 304 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:23, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[IKOS06] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from anonymity. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 239–248, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[JKS19] Woohwan Jung, Suyong Kwon, and Kyuseok Shim. Tidy: Publishing a time interval dataset with differential privacy. *IEEE Transactions on Knowledge and Data Engineering*, 33(5):2280–2294, 2019.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[Mit15] Shigeo Mitsunari. MCL: A portable and fast pairing-based cryptography library. https://github.com/herumi/mcl, 2015. GitHub repository.

[MT07] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *48th FOCS*, pages 94–103, Providence, RI, USA, October 20–23, 2007. IEEE Computer Society Press.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 368–386, San Francisco, CA, USA, March 15–17, 2009. Springer Berlin Heidelberg, Germany.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, New York, NY, USA, 1999. ACM.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 73–85, New York, NY, USA, 1989. ACM.

[SS18] Roshan Shariff and Or Sheffet. Differentially private contextual linear bandits. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 4301–4311, Red Hook, NY, USA, 2018. Curran Associates, Inc.

[SU17] Thomas Steinke and Jonathan R. Ullman. Tight lower bounds for differentially private selection. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 552–563, Los Alamitos, CA, USA, 2017. IEEE Computer Society.

[THHA15] Florian Tramèr, Zhicong Huang, Jean-Pierre Hubaux, and Erman Ayday. Differential privacy with bounded priors: Reconciling utility and privacy in genome-wide association studies. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1286–1297, Denver, CO, USA, October 12–16, 2015. ACM Press.

[War65] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient Multi-Party computation toolkit. https://github.com/emp-toolkit, 2016.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250, Berlin, Heidelberg, 2015. Springer.

Appendices

A Ethical Considerations

This work makes a cryptographic contribution to the domain of distributed differential privacy. The study is in alignment with ethical guidelines, ensuring responsible practices in methodology, data handling, and reporting. The results are based on theoretical analysis and artificial evaluation, posing no negative effects to any individuals or groups. This work involves no human subjects, no user studies, and no personally identifiable information. All experiments are run on synthetically generated circuits and data, which is used to benchmark our protocols.

B Related Work

Differential privacy (DP) has been studied extensively in the central model, where a trusted curator collects users' data and applies a (randomized) mechanism that aims to provide utility while preserving differential privacy guarantee [DMNS06, DR14, DP20, War65, THHA15, JKS19]. In distributed settings, two major alternatives are local differential privacy [EFM+19, War65] and shuffle differential privacy [War65, BEM+17, CSU+19b]. We focus on the shuffle model, whose appeal in part is that the shuffler can often be instantiated using anonymous communication network such as mixnets and onion-routing networks [Cha81, DMS04], and can amplify privacy by hiding the connection between users and their messages.

The shuffle model can nearly match central-model utility [BBGN20, BC20] for several widely used statistical queries, such as summation, histograms, and uniformity testing. For more broader tasks, however, the best known information-theoretic shuffle protocols remain weaker than their central counterparts; this is already visible for selection, distinct elements, and linear contextual bandits, and is consistent with broader lower-bound evidence for high-dimensional learning and estimation in the shuffle model [CSU+19a, CGKM21, CZ22b, CU21].

To bridge this gap, a separate line of work studies computationally secure protocols in the shuffle model. Beimel et al. [BHNS20] showed that any randomized functionality can be computed in two rounds in the computational shuffle model with an honest majority. Other works optimize concrete secure computation tasks in the shuffle model, including secure aggregation and private information retrieval [IKOS06, GIK+24, IKLM24]. This literature demonstrates that the computational shuffle model is expressive, but it typically either operates in a multiple-round regime or targets specific functionalities rather than a generic one-round compiler.

To the best of our knowledge, the only known generic one-round compiler from central DP mechanisms to computational shuffle DP that preserves the central-model utility relies on additive randomized encodings (ARE) [HIKR23, BEG25, BEGI25]. In particularly, their ARE construction yields such a compiler in the non-robust shuffle model, but incurs a designated-client garbling bottleneck: the worst-case per-client computation is linear in the size of the mechanism circuit $O(|M|)$, and the client bandwidth is linear in the number of clients $O(n)$. Our work advances this one-round ARE-based compiler line by making it substantially more client-efficient and more balanced across users. Concretely, we reduce the worst-case client computation from $O(|M|)$ to $O(|M|/n)$, and reduce the client bandwidth from $O(n)$ to a bound governed by the sparsity and cross-partition structure of the compiled circuit.

Our work is also complementary to other distributed models and architectural assumptions. For example, multi-server systems such as Prio [CGB17] assume a small number of non-colluding servers and can additionally provide robustness to malicious clients, whereas we work in the shuffle model

with only one secure shuffler, and a single analyzer. There can also be a server-aided ARE approach that offloads much of the clients' burden to an additional trusted or non-colluding helper. Shuffle model, however, is to avoid this form of re-centralization. More broadly, the ARE literature has recently expanded in several directions, including constructions from public-key encryption, robust ARE in the plain model, and statistical ARE for all functions [BEG25, BF24, BEGI25]. In contrast, our work focuses on ARE constructions that have balanced load on client-side.

C Additional Preliminaries

C.1 Notation

For ease of reference, Table 3 gives an overview of notations and their definitions used throughout this paper.

Table 3: Overview of Notation Used in the Paper

Notation	Description
n	Number of clients
m	Number of messages each client sends
ℓ, ℓ_S, ℓ_A	Client's input length
λ	Computational security parameter
κ	Statistical security parameter
ε, δ	Differential Privacy parameter
$\mathtt{negl}(\cdot)$	Negligible function
$\mathsf{Perm}(\mathcal{S})$	Random permutation over the set $\mathcal{S}$
$\mathcal{M}$	An algorithm (or mechanism)
$\mathrm{Dom}(f), \mathrm{Range}(f)$	Input and output domain of a function f

C.2 DP in the Shuffle Model

Differential Privacy requires that the output of mechanism $\mathcal{M}$ is similar on all *neighboring* inputs D, D' that differ in exactly one data point (we also call D, D' *neighbors*). Informally, if $\mathcal{M}$ is (ε, δ)-DP, an adversary's ability to decide whether $\mathcal{M}$ was run on D or D' is bounded by δ and e^{ε}. We recall the formal definition below.

Definition 11 (Differential Privacy (DP) [DR14]). *A mechanism $\mathcal{M} : \{0,1\}^n \to \{0,1\}^*$ is (ε, δ)-differentially private if the following holds. For every neighboring databases $D, D' \in \{0,1\}^n$ and every distinguisher $\mathsf{Adv} : \{0,1\}^* \to \{0,1\}$,*

$$\Pr[\mathsf{Adv}(\mathcal{M}(D)) = 1] \leq \Pr[\mathsf{Adv}(\mathcal{M}(D')) = 1] \cdot e^{\varepsilon} + \delta \, .$$

If $\delta = 0$, we say that $\mathcal{M}$ is ε-differentially private.

To support a wider class of interesting functionality and improved utility, we consider a variant of the (information-theoretic) differential privacy notion, the computational differential privacy (CDP) [4]. In comparison with the differential privacy, computational differential privacy adopts an asymptotic approach: (1) it restricts the class of distinguishers to those that run in polynomial

[4]There are two main approaches to defining computational differential privacy (CDP): the indistinguishability-based approach and the simulation-based approach. In this work, we adopt the more flexible and widely used indistinguishability-based definition, and use a stronger variant of the one presented in the seminal work [RB89].

time, and (2) it allows a negligible relaxation of the catastrophic failure parameter δ. The formal definition is given below:

Definition 12 (**Computational Differential Privacy (CDP)** [RB89])**.** *A PPT mechanism ensemble*

$$\mathcal{M} \stackrel{\text{def}}{=} \{\mathcal{M}_\lambda : ((\{0,1\}^*)^*) \to \mathbb{R}^*\}$$

is (ε, δ)-computationally differentially private if the following holds. For every PPT distinguisher $\mathsf{Adv} \stackrel{\text{def}}{=} \{\mathsf{Adv}_\lambda\}_{\lambda\in\mathbb{N}}$*, every large enough λ, and every neighboring databases $D, D' \in \mathrm{Dom}(\mathcal{M}_\lambda)$,*

$$\begin{aligned}&\Pr[\mathsf{Adv}_\lambda(\mathcal{M}_\lambda(D)) = 1]\\ \leq\ &\Pr[\mathsf{Adv}_\lambda(\mathcal{M}_\lambda(D')) = 1] \cdot e^\varepsilon + \delta + \mathtt{negl}(\lambda).\end{aligned}$$

Shuffle differentially private protocols [BEM+17, CSU+19b] are a cryptographic primitive for studying differential privacy in settings where parties can perform arbitrary local computation, but the only joint operation is a random permutation. Throughout this paper, we also refer to them as *shuffle protocols* for convenience. A shuffle protocol enables n clients and a server to compute a multiparty differentially private function f over the clients' inputs. Each party locally encodes its input, after which the encodings are randomly permuted by a shuffler (such as a mixnet). The shuffled encodings are then released to the server. Finally, the server decodes them to get an estimate of the function's output under the differential privacy constraint. Formally, the shuffle protocol is defined in the following.

Definition 13 (Shuffle Protocol [BEM+17][CSU+19a])**.** *Let $f : (\{0,1\}^*)^* \mapsto \mathbb{R}^*$ be a multiparty function. A protocol in the shuffle model for f is a tuple of algorithms $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ with the following syntax:*

- $\mathsf{Enc}(i, x_i) \to (x_i', \hat{x}_i)$*, where $\hat{x}_i = \{\hat{x}_i^1, \cdots, \hat{x}_i^m\}$, is a PPT encoding algorithm that maps an input x_i of party i to a set of messages.*
- $\mathsf{Dec}(x_1', \cdots, x_n', \hat{y}_1, \cdots \hat{y}_{mn}) \to y$ *is a PPT decoding algorithm that maps a set of messages to an output y.*

Π is $(\alpha(n), \beta(n))$-accurate if for all $x_1, \cdots, x_n \in \{0,1\}^{\ell_S}$, and large enough n:

$$\Pr\left[\begin{array}{c}\{x_i', \hat{x}_i^1, \cdots, \hat{x}_i^m\} \leftarrow \mathsf{Enc}(i, x_i);\\ \hat{y} = \{\hat{y}_1, \cdots \hat{y}_{mn}\} \leftarrow \mathsf{Perm}(\{\hat{x}_i^1, \cdots, \hat{x}_i^m\}_{i\in[n]});\\ \|\mathsf{Dec}(x_n', \cdots, x_n', \hat{y}) - f(x_1, \ldots, x_n))\|_2 \leq \alpha(n)\end{array}\right] \geq 1 - \beta(n).$$

We say that a shuffle protocol Π is (ε, δ)-computationally differentially private (CDP) if, for any efficient server (or adversary), the ability to distinguish — based on its received message — whether the protocol Π was executed on clients' input D or a neighboring input set D′ is bounded by δ and e^ε. We recall the formal definition below.

Definition 14 (Computational Differential Privacy (CDP) of Shuffle Protocols)**.** *A shuffle protocol $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ is (ε, δ)-computationally differentially private if the algorithm $(x_1', \cdots, x_n', \mathsf{Perm}(\hat{x}_1, \cdots, \hat{x}_n))$ is (ε, δ)-computationally differentially private, where $(x_i', \hat{x}_i) = \mathsf{Enc}(i, x_i)$ for $i \in [n]$.*

Definition 15 (Computational Security of ARE)**.** *An ARE scheme Π for multiparty function f (Theorem 1) is secure if for all n, ℓ_A, $x_1, \cdots, x_n \in \{0,1\}^{\ell_A}$, and every PPT distinguisher* $\mathsf{Adv} \stackrel{\text{def}}{=} \{\mathsf{Adv}_\lambda\}_{\lambda\in\mathbb{N}}$*, there exists a PPT simulator Sim such that the following holds for every large enough λ:*

$$\begin{aligned}&\Big|\Pr\left[\mathsf{Adv}(\mathsf{Sim}(1^\lambda, 1^n, 1^{\ell_A}, f(x_1, \ldots, x_n))) = 1\right]\\ &- \Pr\left[\mathsf{Adv}(\Pi(1^\lambda, x_1, \ldots, x_n)) = 1\right]\Big| \leq \mathtt{negl}(\lambda).\end{aligned}$$

C.3 Split-and-Mix Summation Protocol

Algorithm 4, proposed by [IKOS06], gives a protocol showing how to do secure summation in the shuffle model.

Algorithm 4 Split-and-Mix Summation Protocol [IKOS06]

Local Encoding Function $\mathsf{Enc}(i, x_i)$

Input: Index $i \in [n]$, an element $x_i \in \mathcal{G}$, represented by an ℓ_S-bits string.
Parameters: Message domain $\mathcal{G}$ (a group), number of output messages (shares) m, and statistical security parameter κ.
Output: $\hat{x}_i \in \mathcal{G}^m$, the encoding of input x_i, represented as a vector of group elements in $\mathcal{G}^m$.

1: Additively share the input x_i into m group elements in $\mathcal{G}$, denoted by $\hat{x}_i^1, \ldots, \hat{x}_i^m$
2: Output $\hat{x}_i = (\hat{x}_i^1, \ldots, \hat{x}_i^m)$

Decoding Function $\mathsf{Dec}(\hat{y}_1, \cdots \hat{y}_{mn})$

Input: The shuffled encodings $\hat{y}_i^1, \ldots, \hat{y}_i^{nm} \in \mathcal{G}$ from n clients.

Output: An element $y \in \mathcal{G}$, representing the sum of all clients' inputs, i.e., $y = \sum_{i=1}^{n} x_i$.

1: Output $y = \sum_{i=1}^{mn} \hat{y}_i$.

C.4 Original Additive Randomized Encoding (ARE) Definition

Definition 16 (Additive Randomized Encodings (ARE) Scheme [HIKR23])**.** *Let $f : (\{0,1\}^*)^* \mapsto \{0,1\}^*$ be a multiparty function. An ARE scheme for f is a triple of algorithms $\Pi = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ with the following syntax:*

- *$\widetilde{\mathsf{Setup}}(1^\lambda, 1^n, 1^\ell) \to \mathsf{pp}$ is a PPT setup algorithm that, given security parameter λ, number of parties n, and input length ℓ, generates public parameters pp. The public parameters include λ, n, ℓ, and an explicit description of an Abelian group $\mathcal{G}$. They can also include some randomness, such as random generators of $\mathcal{G}$ and/or a common reference string (CRS). We will sometimes eliminate Setup and consider $\mathcal{G}$ as being fixed.*
- *$\widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i) \to \hat{x}_i$ is a PPT encoding algorithm that maps an input x_i of party i to a group element from $\mathcal{G}$. We refer to $\hat{x}_i$ as the encoding of x_i, or the ARE message of party i.*
- *$\widetilde{\mathsf{Dec}}(\mathsf{pp}, \hat{y}) \to y$ is a PPT decoding algorithm that maps a group element $\hat{y} \in \mathcal{G}$ to an output y.*

Π is correct, with a possible error of $\beta = \beta(\lambda)$, if for all λ, n, ℓ, and $x_1, \ldots, x_n \in \{0,1\}^\ell$:

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \widetilde{\mathsf{Setup}}(1^\lambda, 1^n, 1^\ell); \\ \hat{x}_i \leftarrow \widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i); \\ \hat{y} = \sum_{i=1}^n \hat{x}_i : \widetilde{\mathsf{Dec}}(\mathsf{pp}, \hat{y}) = f(x_1, \ldots, x_n) \end{array}\right] \geq 1 - \beta(\lambda),$$

where summation is taken over the group $\mathcal{G}$ specified by pp.

Definition 17 (Computational Security of ARE)**.** *An ARE scheme Π for multiparty function f (Theorem 1) is secure if for all $n, \ell_A, x_1, \cdots, x_n \in \{0,1\}^{\ell_A}$, and every PPT distinguisher $\mathsf{Adv} \stackrel{\text{def}}{=} \{\mathsf{Adv}_\lambda\}_{\lambda \in \mathbb{N}}$, there exists a PPT simulator Sim such that the following holds for every large enough*

λ:

$$\Big|\Pr\left[\mathsf{Adv}(\mathit{Sim}(1^\lambda, 1^n, 1^{\ell_A}, f(x_1, \ldots, x_n))) = 1\right] - \Pr\left[\mathsf{Adv}(\Pi(1^\lambda, x_1, \ldots, x_n)) = 1\right]\Big| \leq \mathtt{negl}(\lambda).$$

C.5 Circuit Partition

In this work, we represent the (differentially private) computation task as a Boolean circuit. To jointly compute it in a distributed setting, our ARE construction partitions the circuit, assigning each party the task of preparing an encoding for one partition. We define below what constitutes a good circuit partitioning — one that balances the size of each partition and has a small number of wires (edges) crossing between partitions.

Definition 18 ((γ, n, t)-Circuit Partitioning)**.** *Let $\gamma > 0$ and $n, t \in \mathbb{N}$. A Boolean circuit $\mathcal{C}$ admits a (γ, n, t)-partition if there exists a collection of disjoint gates subsets $\{\mathcal{C}_1, \cdots, \mathcal{C}_n\}$ satisfying the following:*

- ***Partition:*** *$\bigcup_{i\in[n]} \mathcal{C}_i = \mathcal{C}$ and $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for all $i \neq j \in [n]$.*
- ***Balanced size:*** *Each part has at most $(1+\gamma)\frac{|\mathcal{C}|}{n}$ gates; that is, $|\mathcal{C}_i| \leq (1+\gamma)\frac{|\mathcal{C}|}{n}$ for all $i \in [n]$.*
- ***Bounded boundary:*** *The maximum number of wires leaving any part is at most t, i.e.,*

$$\max_{i\in[n]} \mu(\mathcal{C}_i) \leq t,$$

$\mu(\mathcal{C}_i)$ denotes the number of wires that cross the boundary between $\mathcal{C}_i$ and $\mathcal{C} \setminus \mathcal{C}_i$, i.e., wires that are inputs to one and outputs of the other.

We refer readers to Appendix E for a discussion of how such a partitioning can be achieved, based on the well-studied problem of partitioning the circuit's underlying graph representation.

A circuit $\mathcal{C}$ can be represented as an undirected graph G that abstracts away the functionality and directionality of gates, capturing only the circuit's connectivity structure. Theorem 19 provides the formal definition of this graph representation.

Definition 19 (Graph Representation of a Circuit)**.** *Let $\mathcal{C} = \{\mathcal{C}_1, \cdots, \mathcal{C}_{|\mathcal{C}|}\}$ be a Boolean circuit, where each $\mathcal{C}_i$ is a logic gate with two input wires and one output wire. We define the* ***structural graph representation*** *of $\mathcal{C}$ as an undirected graph $G = (V, E)$, where:*

- *Each node $v_i \in V$ corresponds to a gate $\mathcal{C}_i$,*
- *There is an edge $(v_i, v_j) \in E$ if the output of $\mathcal{C}_i$ is connected to an input of $\mathcal{C}_j$, or vice versa.*

Definition 20 ((γ, n, t)-Graph Partitioning)**.** *Let $\gamma > 0$ and $n, t \in \mathbb{N}$. An undirected graph $G = (V, E)$ admits a (γ, n, t)-graph partitioning if there exists a collection of disjoint vertex subsets $\{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ satisfying the following:*

- ***Partition:*** *$\bigcup_{i\in[n]} \mathcal{S}_i = V$ and $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for all $i \neq j \in [n]$.*
- ***Balanced size:*** *Each part has size at most $(1+\gamma)\frac{|V|}{n}$; that is, $|\mathcal{S}_i| \leq (1+\gamma)\frac{|V|}{n}$ for all $i \in [n]$.*
- ***Bounded boundary:*** *The maximum number of edges leaving any part is at most t, i.e.,*

$$\max_{i\in[n]} \mu(\mathcal{S}_i) \leq t,$$

where $\mu(\mathcal{S}_i)$ denotes the number of edges crossing the cut $(\mathcal{S}_i, V \setminus \mathcal{S}_i)$.

C.6 Security of Garbled Circuits

Definition 21 (Computational Security of Garbling Scheme)**.** *Let the garbled circuit $\widehat{\mathcal{C}}$, garbled input $\tilde{x}$ and the decoding set dK be the output of the garbling scheme Π on every circuit $\mathcal{C}$, circuit's input x and security parameter λ. A garbling scheme Π is secure if it has the following properties:*

- ***Privacy.*** *The garbling scheme Π is* private *if there exists a PPT simulator $\mathsf{Sim}_{\text{priv}}$ that takes input $(1^\lambda, \mathcal{C}, \mathcal{C}(x))$ and whose output is computationally indistinguishable from $(\widehat{\mathcal{C}}, \tilde{x}, dK)$.*
- ***Obliviousness.*** *The garbling scheme Π is* oblivious *if there exists a PPT simulator $\mathsf{Sim}_{\text{obliv}}$ that takes input $(1^\lambda, \mathcal{C})$ and whose output is computationally indistinguishable from $(\widehat{\mathcal{C}}, \tilde{x})$.*

D Construction of OT-ARE Scheme

Definition 22 (Squaring XDH)**.** *The decision Squaring-XDH holds for a paring parameter-generator (Theorem 23) $\mathcal{G}$, if the following distribution ensembles are computationally indistinguishable:*

$$\mathsf{SQ}_\lambda \stackrel{\text{def}}{=} (pp_\lambda,\ \rho \cdot g_1,\ \rho^2 \cdot g_1), \quad \mathsf{U}_\lambda \stackrel{\text{def}}{=} (pp_\lambda,\ \rho \cdot g_1,\ \rho' \cdot g_1),$$

where $pp_\lambda = (q, G_1, G_2, G_T, g_1, g_2, g_T, e) \leftarrow \mathcal{G}(1^\lambda)$ and $\rho, \rho' \leftarrow \mathbb{Z}_q$.

Pairing Groups. The ARE constructions in our work are pairing-based and we use notation of pairing-friendly groups that are similar to [HIKR23, BF03].

Definition 23 (Pairing Parameter-Generator)**.** *A* pairing parameter-generator *is an efficient (randomized) algorithm $(q, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T, g_1, g_2, g_T, e) \leftarrow \mathcal{A}(1^\lambda)$, given the security parameter λ, outputs a description of additive groups $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T$ of the same order q, distinguished generators $g_1 \in \mathcal{G}_1$, $g_2 \in \mathcal{G}_2$, and an efficiently computable and nontrivial bilinear map*

$$e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_T.$$

A distinguished generator in $\mathcal{G}_T$ can be computed as $g_T \stackrel{\text{def}}{=} e(g_1, g_2)$.

E Additional Details for Bandwidth Bounds

We derive upper bounds on the client-side bandwidth of our shuffle protocols. Our main result, restated in Theorem 30, expresses this bound in terms of properties of the to-compute Boolean circuit $\mathcal{C}$ and the number of parties n. To state the result, we recall the notion of *Circuit Partitioning*, which captures exactly the properties of $\mathcal{C}$ needed to describe the bound. A (γ, n, t)-Circuit Partitioning divides $\mathcal{C}$ into n disjoint subcircuits, each of size is at most a $(1+\gamma)/n$ fraction of the total, with at most t wires crossing between subcircuits. The formal definition is given below.

Definition 24 ((γ, n, t)-Circuit Partitioning)**.** *Let $\gamma > 0$ and $n, t \in \mathbb{N}$. A Boolean circuit $\mathcal{C}$ admits a (γ, n, t)-partition if there exists a collection of disjoint gates subsets $\{\mathcal{C}_1, \cdots, \mathcal{C}_n\}$ satisfying the following:*

- ***Partition:*** *$\bigcup_{i \in [n]} \mathcal{C}_i = \mathcal{C}$ and $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for all $i \neq j \in [n]$.*
- ***Balanced size:*** *Each part has at most $(1+\gamma)\frac{|\mathcal{C}|}{n}$ gates; that is, $|\mathcal{C}_i| \leq (1+\gamma)\frac{|\mathcal{C}|}{n}$ for all $i \in [n]$.*
- ***Bounded boundary:*** *The maximum number of wires leaving any part is at most t, i.e.,*

$$\max_{i \in [n]} \mu(\mathcal{C}_i) \leq t,$$

$\mu(\mathcal{C}_i)$ denotes the number of wires that cross the boundary between $\mathcal{C}_i$ and $\mathcal{C} \setminus \mathcal{C}_i$, i.e., wires that are inputs to one and outputs of the other.

Algorithm 5 Equality ARE Scheme

Parameters:

- $f_{\mathsf{eq}} : \{0,1\} \times \{0,1\} \to \{0,1\}, \quad f_{\mathsf{eq}}(b_0, b_1) = \begin{cases} 1 & \text{if } b_0 = b_1 \\ 0 & \text{otherwise,} \end{cases}$
- The scheme has $n = 2$ clients and each client holds a bit
- The pairing parameter-generator (Theorem 23) $\mathcal{A}$

$\widetilde{\mathsf{Setup}}(1^\lambda, 1^n, 1^1)$

1: Let $\lambda' = \max(\lambda, 2)$, set $\mathsf{pp} = (q, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T, g_1, g_2, g_T, e) \leftarrow \mathcal{A}(1^{\lambda'})$;
2: Let $\mathsf{emb} : \{0,1\}^{\ell_A} \to [1, \frac{q-1}{2}]$ be injective (e.g., $\mathsf{emb}(s) = \mathsf{bin}(s) + 1$, where $\mathsf{bin}(\cdot)$ is the binary-to-integer conversion function).

Local Encoding Function $\widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i)$

Input: Party index $i \in \{0,1\}$ and its input $x_i \in \{0,1\}$.
Output: $\hat{x}_i \in \mathcal{G}_1 \times \mathcal{G}_2 \times \mathcal{G}_T$, the encoding of input x_i.

1: Embed $\chi_i = \mathsf{emb}(x_i) \in \left[1, \frac{q-1}{2}\right]$
2: Choose at random $r_i \leftarrow \mathbb{Z}_q$
3: Send $\hat{x}_i = \left(r_i \cdot g_1,\ (-1)^i \chi_i r_i \cdot g_2,\ (-1)^i \chi_i r_i^2 \cdot g_T\right)$

Decoding Function $\widetilde{\mathsf{Dec}}(\mathsf{pp}, \hat{y})$

Input: The sum of the encodings. $\hat{y} \in \mathcal{G}_1 \times \mathcal{G}_2 \times \mathcal{G}_T$
Output: An element $y = f_{\mathsf{eq}}(x_0, x_1) \in \{0,1\}$

1: If $e(\hat{y}_1, \hat{y}_2)$ equals to $\hat{y}_3$, set $y \leftarrow 1$, otherwise set $y \leftarrow 0$.
2: Output y.

Algorithm 6 Rabin-OT ARE Scheme

Parameters:

- $f_{\mathsf{rot}} : \{0,1\} \times (\{0,1\} \times \{0,1\}^{\ell_A}) \to \{0,1\}^{\ell_A}, \quad f_{\mathsf{rot}}((b_0, s), b_1) = \begin{cases} s & \text{if } b_0 = b_1 \\ r & \text{otherwise,} \end{cases}$
- The scheme has $n = 2$ clients and player $i = 0$ holds (b_0, s) and player $i = 1$ holds selection bit b_1.
- The pairing parameter-generator (Theorem 23) $\mathcal{A}$

$\widetilde{\mathsf{Setup}}(1^\lambda, 1^n, 1^{\ell_A+1})$

1: Let $\lambda' = \max(\lambda, \ell_A + 1)$, set $\mathsf{pp} = (q, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T, g_1, g_2, g_T, e) \leftarrow \mathcal{A}(1^{\lambda'})$ (recall that $q > 2^{\ell_A+2}$);
2: Let $\mathsf{emb} : \{0,1\}^{\ell_A} \to [1, \frac{q-1}{2}]$ be injective (e.g., $\mathsf{emb}(s) = \mathsf{bin}(s) + 1$, where $\mathsf{bin}(\cdot)$ is the binary-to-integer conversion function).

Local Encoding Function $\widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i)$

Input: Party index $i \in \{0,1\}$; player i's input $x_0 = (b_0, s) \in \{0,1\} \times \{0,1\}$ if $i = 0$, and $x_1 = b_1 \in \{0,1\}^\ell$ if $i = 1$
Output: $\hat{x}_i \in \mathcal{G}_1 \times \mathcal{G}_2 \times \mathcal{G}_T$, the encoding of input x_i.

1: Embed $\chi_i = \mathsf{emb}(b_i) \in \left[1, \frac{q-1}{2}\right]$
2: Choose at random $r_i \leftarrow \mathbb{Z}_q$
3: **if** $i = 0$ **then**
4: Send $\hat{x}_i = \left(r_i \cdot g_1,\ \chi_i r_i \cdot g_2,\ \left(\chi_i r_i^2 + \mathsf{emb}(s)\right) \cdot g_T\right)$
5: **else**
6: Send $\hat{x}_i = \left(r_i \cdot g_1,\ (-\chi_i r_i) \cdot g_2,\ (-\chi_i r_i^2) \cdot g_T\right)$
7: **end if**

Decoding Function $\widetilde{\mathsf{Dec}}(\mathsf{pp}, \hat{y})$

Input: The sum of the encodings. $\hat{y} \in \mathcal{G}_1 \times \mathcal{G}_2 \times \mathcal{G}_T$
Output: An element $y = f_{\mathsf{rot}}(x_0, x_1) \in \{0,1\}^{\ell_A}$

1: Compute $y \leftarrow \hat{y}_3 - e(\hat{y}_1, \hat{y}_2)$.
2: Output y.

Algorithm 7 1-of-2 String OT ARE Scheme

Parameters:

- $f_{\mathsf{ot}} : \{0,1\} \times \{0,1\}^{2\ell_A} \to \{0,1\}^{\ell_A}, \quad f_{\mathsf{ot}}((s_0, s_1), b) = \begin{cases} s_0 & \text{if } b = 0 \\ s_1 & \text{otherwise,} \end{cases}$
- $\Pi^{\mathsf{rot}}_{\mathrm{ARE}} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$: An ARE scheme for f_{rot}.
- The scheme has $n = 2$ clients and player $i = 0$ holds a pair of messages (s_0, s_1) and player $i = 1$ holds selection bit b.
- The pairing parameter-generator (Theorem 23) $\mathcal{A}$

$\underline{\widetilde{\mathsf{Setup}}(1^\lambda, 1^n, 2^{2\ell_A})}$

1: Let $\lambda' = \max(\lambda, \ell_A + 1)$, set $\mathsf{pp} = (q, \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_T, g_1, g_2, g_T, e) \leftarrow \mathcal{A}(1^{\lambda'})$ (recall that $q > 2^{\ell_A+1}$);
2: Let $\mathsf{emb} : \{0,1\}^{\ell_A} \to [1, \frac{q-1}{2}]$ be injective (e.g., $\mathsf{emb}(s) = \mathsf{bin}(s) + 1$, where $\mathsf{bin}(\cdot)$ is the binary-to-integer conversion function).

$\underline{\textbf{Local Encoding Function } \widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i)}$

Input: Party index $i \in \{0,1\}$; player i's input $x_0 = (s_0, s_1) \in \{0,1\}^{\ell_A} \times \{0,1\}^{\ell_A}$ if $i = 0$, and $x_1 = b \in \{0,1\}$ if $i = 1$
Output: $\hat{x}_i \in (\mathcal{G}_1 \times \mathcal{G}_2 \times \mathcal{G}_T)^2$, the encoding of input x_i.

1: Choose at random $r_i \leftarrow \mathbb{Z}_q$
2: First message: $(b,\ s_b|0^\lambda)$
3: **if** $i = 0$ **then**
4: Choose at random $c \leftarrow \{0,1\}$
5: $\hat{x}_i^1 \leftarrow \Pi^{rot}_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i = 0, x_i = (c, s_c \| 0^\lambda))$
6: $\hat{x}_i^2 \leftarrow \Pi^{rot}_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i = 0, x_i = (1-c, s_{1-c} \| 0^\lambda))$
7: Send $\hat{x}_i = (\hat{x}_i^1, \hat{x}_2^1)$
8: **else**
9: $\hat{x}_i^1 \leftarrow \Pi^{rot}_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i = 1, x_i = b)$
10: $\hat{x}_i^2 \leftarrow \Pi^{rot}_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i = 1, x_i = b)$
11: Send $\hat{x}_i = (\hat{x}_i^1, \hat{x}_2^1)$
12: **end if**

$\underline{\textbf{Decoding Function } \widetilde{\mathsf{Dec}}\,(\mathsf{pp}, \hat{y})}$

Input: The sum of the encodings. $\hat{y} \in (\mathcal{G}_1 \times \mathcal{G}_2 \times \mathcal{G}_T)^2$
Output: An element $y = f_{\mathsf{ot}}(x_0, x_1) \in \{0,1\}^{\ell_A}$

1: Compute $y_1 \leftarrow \Pi^{rot}_{\mathrm{ARE}}.\widetilde{\mathsf{Dec}}(\mathsf{pp}, \hat{y}_1), y_2 \leftarrow \Pi^{rot}_{\mathrm{ARE}}.\widetilde{\mathsf{Dec}}(\mathsf{pp}, \hat{y}_2)$
2: Output whichever of y_0 or y_1 has at least λ trailing zeros.

Definition 25 (Balanced non-XOR weighted n-way partition)**.** *Let $\mathcal{C}$ be a Boolean circuit with gate set V and (fanout) dependency edges $E \subseteq V \times V$, where each edge $(u \to v) \in E$ represents a wire from the output of gate u to an input pin of gate v. Define the non-XOR weight function $w : V \to \{0, 1\}$ by*

$$w(v) = \mathbf{1}[v \text{ is a non-XOR gate}],$$

and let $W_{\text{tot}} = \sum_{v \in V} w(v)$. A partition $\pi : V \to [n]$ induces parts $V_i = \{v \in V : \pi(v) = i\}$. We say π is γ-balanced (w.r.t. w) if for all $i \in [n]$,

$$\sum_{v \in V_i} w(v) \;\le\; (1+\gamma)\frac{W_{\text{tot}}}{n}.$$

The pin-level cut of π is

$$\text{cut}_{\text{pin}}(\pi) = \big|\{(u \to v) \in E :\; \pi(u) \neq \pi(v)\}\big|.$$

Problem 2 (Balanced non-XOR min-cut partitioning)**.** *Given (V, E), weight function w as above, parameters $n \in \mathbb{N}$ and $\gamma > 0$, find a partition $\pi : V \to [n]$ that solves:*

$$\min_{\pi : V \to [n]} \quad \text{cut}_{\text{pin}}(\pi) = \big|\{(u \to v) \in E :\; \pi(u) \neq \pi(v)\}\big|$$
$$\text{s.t.} \quad \sum_{v \in V_i} w(v) \le (1+\gamma)\frac{W_{\text{tot}}}{n} \qquad \forall i \in [n],$$

where $V_i = \{v \in V : \pi(v) = i\}$ and $W_{\text{tot}} = \sum_{v \in V} w(v)$.

Problem 3 (Balanced non-XOR min-max outgoing boundary partitioning)**.** *Let $\mathcal{C}$ have gate set V and pin-level dependency edges $E \subseteq V \times V$. Let $w : V \to \{0, 1\}$ be the non-XOR indicator weight, $w(v) = \mathbf{1}[v \text{ is non-XOR}]$, and $W_{\text{tot}} = \sum_{v \in V} w(v)$. Given parameters $n \in \mathbb{N}$ and $\gamma > 0$, find a partition $\pi : V \to [n]$ minimizing the maximum outgoing boundary:*

$$\min_{\pi : V \to [n]} \quad \max_{i \in [n]} \; \mu_{\text{out}}(V_i)$$
$$\text{s.t.} \quad \sum_{v \in V_i} w(v) \le (1+\gamma)\frac{W_{\text{tot}}}{n} \qquad \forall i \in [n],$$

where $V_i = \{v \in V : \pi(v) = i\}$ and

$$\mu_{\text{out}}(V_i) = \big|\{(u \to v) \in E :\; u \in V_i,\; v \notin V_i\}\big|.$$

Theorem 26 states that without assumptions on the circuit $\mathcal{C}$, there always exists a circuit for which every (γ, n, t)-Circuit Partitioning must have at least $t = \Omega(|\mathcal{C}|/n)$ wires crossing the boundaries. One can see this by constructing a graph from the circuit and analyzing the connectivity properties of the resulting graph. The full proof is deferred to Appendix G.3.

Lemma 26. *Let $n, m \in \mathcal{N}$ be integers and $\mathbb{C}_{n,m}$ be the class of Boolean circuit with $O(n)$ input wires and size m. Then, there exists a universal constant $c > 0$ such that for every larger enough m and $n \in \{2, 3, \ldots, m\}$, there exists a circuit $\mathcal{C} \in \mathbb{C}_{n,m}$ with the following property: for every balanced n-partition $\mathcal{C} = \{\mathcal{C}_1, \cdots, \mathcal{C}_n\}$ (i.e., $m/(2n) \le |\mathcal{C}_i| \le 2m/n$ for all i),*

$$\max_{i \in [n]} \mu(\mathcal{C}_i) \ge c \cdot \frac{|\mathcal{C}|}{n},$$

where $\mu(\mathcal{C}_i)$ denotes the number of wires with exactly one endpoint in $\mathcal{C}_i$.

Theorem 27 shows that for any circuit $\mathcal{C}$, one can construct a $(\gamma, n, O(|\mathcal{C}|/n))$-Circuit Partitioning. Combined with Theorem 26, this establishes that $O(|\mathcal{C}|/n)$ is a tight upper bound on the number of wires crossing the boundaries when no assumptions are made on $\mathcal{C}$.

Lemma 27 (Proof is in Appendix G.3). *For every Boolean circuit $\mathcal{C}$ and every integer $n \in \mathbb{N}$, the circuit $\mathcal{C}$ admits a $(0, n, 3\lceil\frac{|\mathcal{C}|}{n}\rceil)$-partition.*

In many cases, the computation has a well-structured pattern of intermediate value dependencies. This structure allows the its circuit $\mathcal{C}$ to be partitioned into levels such that every output wire either connects to the same level or the next. We use the notion of *Leveled Boolean circuit* to capture this property and formally define it below.

Definition 28 (Leveled Boolean circuit). *A Boolean circuit $\mathcal{C}$ is* leveled *if its gates can be partitioned into levels $L_1, L_2, \ldots$ such that every output wire either connects to the same level L_i or the next level L_{i+1}. The* width t *of $\mathcal{C}$ is defined as*

$$t \stackrel{\text{def}}{=} \max_i \mu(L_i),$$

where $\mu(L_i)$ denotes the number of wires with exactly one endpoint in L_i.

Theorem 29 shows that for a leveled Boolean circuit with width t, one can construct a Circuit Partitioning with at most $2t$ wires crossing between subcircuits.

Lemma 29 (Proof is in Appendix G.3). *Let integer $n \in \mathcal{N}$ and $\mathcal{C}$ be a leveled Boolean circuit with levels $L_1, \ldots$. Then, the circuit $\mathcal{C}$ admits a $(\frac{gn}{|\mathcal{C}|}, n, 2t)$-partition, where t is the width of $\mathcal{C}$ and $g = \max_i |L_i|$.*

Theorem 30 states the upper bound on the client-side bandwidth of shuffle protocols compiled with our compiler Π_{oSHARE}. In the general case, with no assumption made on $\mathcal{C}$, the client bandwidth is $O(\frac{|\mathcal{C}|}{n})$. If $\mathcal{C}$ is leveled with width t, the client bandwidth improves to $O(t)$. Theorem 30 follows from the observation that a client's bandwidth is proportional to the number of wires crossing its assigned subcircuit. The result then follows by directly applying the results given in Theorem 26, Theorem 27 and Theorem 29.

Theorem 30 (Upper Bounds for Bandwidth in Shuffle Protocols). *Let f be a multi-input function that admits a Boolean circuit $\mathcal{C}$. The shuffle protocol Π_{oSHARE} for f has the following upper bounds on the client-side bandwidth:*

- *For an arbitrary Boolean circuit $\mathcal{C}$, the bandwidth for each client is at most*

$$O\left(\frac{|\mathcal{C}|}{n}\right).$$

  Moreover, this bound is tight in general without further restrictions on $\mathcal{C}$.
- *If $\mathcal{C}$ is a leveled Boolean circuit with width t, then the bandwidth for each client is at most $O(t)$.*

F Security Proof for Balanced ARE Scheme

F.1 Security Proof for Permute Xor-OT ARE Scheme

Proof of Theorem 5. The correctness of Algorithm 1 can be easily verified by following the protocol execution. In the decoding phase, the server computes $y_1 = (s'_0 \| s'_1) \oplus r$, the masked input from the receiver, and $y_2 = (s_b \| s_{1-b}) \oplus r$, the masked input from the sender. Setting $y \leftarrow y_1 \oplus y_2$, we obtain

$$y = (s_b \| s_{1-b}) \oplus (s'_0 \| s'_1) = (s_b \oplus s'_0, s_{1-b} \oplus s'_1),$$

which is exactly the desired output of the Permute Xor functionality, defined in Theorem 4.

The security of Algorithm 1 can also be easily shown. We first construct a PPT simulator Sim_1 as the following: it takes as input the $(s_b \oplus s'_0, s_{1-b} \oplus s'_1)$ and samples a random string

$\bar{r} \in \{0,1\}^{2\ell_A}$, and then it outputs a tuple of string $((s_b\|s_{1-b}) \oplus (s'_0\|s'_1) \oplus \bar{r}, \bar{r})$. One can check that $((s_b\|s_{1-b}) \oplus (s'_0\|s'_1) \oplus \bar{r}, \bar{r})$ and $((s'_0\|s'_1) \oplus r, (s_b\|s_{1-b}) \oplus r)$, where $r \leftarrow \{0,1\}^{2\ell_A}$ follow the same distribution: for every string $x_1, x_2 \in \{0,1\}^{2\ell_A}$,

$$
\begin{aligned}
& \Pr\left[(s'_0\|s'_1) \oplus r = x_1, (s_b\|s_{1-b}) \oplus r = x_2\right] \\
= \ & \Pr\left[r = x_1 \oplus (s'_0\|s'_1), r = x_2 \oplus (s_b\|s_{1-b})\right] \\
= \ & \begin{cases} 0 & x_1 \oplus x_2 \neq (s'_0\|s'_1) \oplus (s_b\|s_{1-b}) \\ \frac{1}{2^{2\ell_A}} & x_1 \oplus x_2 = (s'_0\|s'_1) \oplus (s_b\|s_{1-b}) \end{cases} \\
= \ & \Pr\left[(s_b\|s_{1-b}) \oplus (s'_0\|s'_1) \oplus \bar{r} = x_1, \bar{r} = x_2\right].
\end{aligned}
$$

Secondly, since Algorithm 1 adopts a correct and secure OT ARE scheme, there exists a PPT simulator Sim_{OT} such that, its output given $y_2 = (s_b\|s_{1-b}) \oplus r$ is computationally indistinguishable with $\hat{y}_2$. Finally, we construct a PPT simulator Sim, building upon Sim_1 and Sim_{OT}, which takes as input the $(s_b \oplus s'_0, s_{1-b} \oplus s'_1)$ and its output is computationally indistinguishable with what the server observes in the Permute Xor-OT ARE Scheme.

Sim first run Sim_1 and obtains the tuple $((s_b\|s_{1-b}) \oplus (s'_0\|s'_1) \oplus \bar{r}, \bar{r})$. Which it further uses to output $((s_b\|s_{1-b}) \oplus (s'_0\|s'_1) \oplus \bar{r}, \mathsf{Sim}_{OT}(\bar{r}))$. By the triangle inequality of computational indistinguishability, one observes that $((s_b\|s_{1-b}) \oplus (s'_0\|s'_1) \oplus \bar{r}, \mathsf{Sim}_{OT}(\bar{r}))$ is computationally indistinguishable with $(\hat{y}_1, \hat{y}_2)$, therefore we complete the proof. □

F.2 Security Proof for Balanced ARE Scheme

Proof of Theorem 6. We start by showing the correctness of the ARE scheme given in Theorem 6. If the server correctly gathers all the necessary inputs for evaluating the garbled circuit, then the correctness of the balanced ARE follows directly from the correctness of its underlying garbling scheme. We then argue that the server correctly collects the garbled keys for each input wire, garbled circuits, and the decoding sets. For the garbled circuits and the decoding sets, since those are sent by the clients directly, the correctness directly follows the correctness of the underlying garbling scheme. The server runs an OT ARE instance with two clients to obtain the corresponding garbled keys for each circuit's input bit. Thus, the correctness of the garbled input follows directly from the correctness of the OT ARE scheme.

To obtain the garbled key for each boundary wire, the server first evaluate the preceding garbled circuit to retrieve the old garbled key. It then runs a Permute Xor ARE instance with two clients to obtain the masked new garbled key. Finally, the server derives the new garbled key by removing the mask using the old garbled key. The correctness of the garbled key for each boundary wire follows from the correctness of the Permute Xor ARE scheme and the underlying garbling scheme. The garbled key for other wires directly follows the correctness of the underlying garbling scheme.

To show the security of the ARE scheme, we first present a simulator algorithm $\mathsf{Sim}_{\mathrm{ARE}}$ as described in Algorithm 8. Given the circuit $\mathcal{C}$ and its output $\mathcal{C}(x_1, \ldots, x_n)$ on client inputs $x_1, \ldots, x_n$, the simulator generates a simulated view of the server in the ARE scheme. The simulator $\mathsf{Sim}_{\mathrm{ARE}}$ first invokes the garbling scheme's simulator $\mathsf{Sim}^{\mathrm{garbling}}$ on input $\mathcal{C}, \mathcal{C}(x_1, \ldots, x_n)$ to get a simulated garbled circuit $\widehat{\mathcal{C}}$, garbled input $\tilde{x}$ and the decoding set dK. It then uses these to generate the two components of the ARE encoding, namely $\hat{y}, y'$. The encoding y' consists of each client's garbled subcircuit $\widehat{\mathcal{C}}_i$ and their part of decoding set dK_i. The encoding $\hat{y}$ consists of the ARE encodings of the garbled inputs for each external input wire, as well as the ARE encodings of the masked garbled keys for each boundary wire.

Generating the dneurecoding set part dK_i from the simulated decoding set is straightforward. Simulating the garbled subcircuits $\widehat{\mathcal{C}}_i$ follows a similar approach, with additional care of boundary

Algorithm 8 $\mathsf{Sim}_{\mathrm{ARE}}$

Parameters: Security parameter λ; number of client n; input size of client ℓ_A; and circuit partitioning $(\mathcal{C}_1, \cdots, \mathcal{C}_n)$ for circuit $\mathcal{C}$

Input: Circuit $\mathcal{C}$ and its output $\mathcal{C}(x_1, \cdots, x_n)$

Output: The sum of the ARE encodings: $\hat{y}$; and the encoding $y' = (x'_1, \cdots, x'_n)$

$(\widehat{\mathcal{C}}, \tilde{x}, dK) \leftarrow \mathsf{Sim}_{\mathrm{priv}}^{\mathrm{garbling}}(1^\lambda, \mathcal{C}, \mathcal{C}(x_1, \cdots, x_n))$, where $\mathsf{Sim}_{\mathrm{priv}}^{\mathrm{garbling}}$ is the simulator for garbling scheme described in Theorem 21.

Initialize the garbled circuit as an empty map $\widehat{\mathcal{C}}_i \leftarrow \emptyset$ for each circuit part i.

Initialize a map R that assigns each boundary wire w a pair of random labels: $R[w] \leftarrow (k_0, k_1)$

// Simulate the encoding $y' = (x'_1, \cdots, x'_n)$ (Refer to the wire naming in Section 3.)

For each gate $C \in \mathcal{C}_i$ whose output wires is non-boundary::

    Add the garbled table $T = \widehat{\mathcal{C}}[C]$ for C to $\widehat{\mathcal{C}}_i$: $\widehat{\mathcal{C}}_i[C] \leftarrow T$

For each gate $C \in \mathcal{C}_i$ whose output wire is boundary:

    Retrieve the input wire labels of C from $\widehat{\mathcal{C}}$, assign $R[w]$ as the output wire label of C, and generate the garbled table T using these input and output labels.

    Add the garbled table T for C to $\widehat{\mathcal{C}}_i$: $\widehat{\mathcal{C}}_i[C] \leftarrow T$

Partition $(dK_1, \cdots, dK_n) \leftarrow dK$ and set $x'_i \leftarrow (\widehat{\mathcal{C}}_i, dK_i)$ for each $i \in [n]$.

// Simulate the encoding $\hat{y}$

For each external input wire w of $\mathcal{C}$:

    $\hat{y}[\phi(w)] \leftarrow \mathsf{Sim}^{\mathsf{ot}}(1^\lambda, \tilde{x}_w)$, where $\tilde{x}_w$ is the garbled input corresponding to wire w, and $\mathsf{Sim}^{\mathsf{ot}}$ is the simulator defined by the security of the OT ARE scheme $\Pi_{\mathrm{ARE}}^{\mathsf{ot}}$.

For each boundary wire w of $\mathcal{C}$:

    $b \leftarrow \mathrm{LSB}\,(R[w][0])$

    $\hat{y}[\phi(w)] \leftarrow \mathsf{Sim}^{\mathsf{pxt}}\left(1^\lambda, (K_{w0} \oplus R[w][0], K_{w1} \oplus R[w][1])\right)$ if $b = 0$, else $\hat{y}[\phi(w)] \leftarrow \mathsf{Sim}^{\mathsf{pxt}}\left(1^\lambda, (K_{w1} \oplus R[w][1], K_{w0} \oplus R[w][0])\right)$, where (K_{w0}, K_{w1}) is the wire label used in $\widehat{\mathcal{C}}$ for the wire w, and $\mathsf{Sim}^{\mathsf{pxt}}$ is the simulator defined by the security of the Permute Xor ARE scheme $\Pi_{\mathrm{ARE}}^{\mathsf{pxt}}$.

Return $\hat{y}, y'$

wires. Each boundary wire appears as an output wire in the preceding subcircuit and as an input wire in the succeeding subcircuit. However, since the simulated garbled circuit provides only one pair of label per wire. So the simulator $\mathsf{Sim}_{\mathrm{ARE}}$ randomly generates new labels for each boundary output wire of the last-layer gates in the preceding subcircuits, and uses these labels to construct the corresponding garbled tables.

To generate the encoding $\hat{y}$, the simulator invokes the OT ARE simulator for each external input wire w, using the simulated garbled input $\tilde{x}_w$ to generate the corresponding ARE encoding. For each boundary wire, the simulator invokes the Permute Xor ARE simulator to generate the corresponding ARE encoding using the garbled label from both the preceding and the succeeding subcircuit.

Finally, we argue that $\mathsf{Sim}_{\mathrm{ARE}}$'s output are computationally indistinguishable with what the server observes in the balanced ARE Scheme if the underlying Garbling scheme, OT ARE scheme and the permute Xor ARE scheme are secure. The argument follows a standard hybrid proof, relying on a polynomial number times of the triangle inequality for computational indistinguishability. □

G Missing Proof for Shuffle DP Protocols

G.1 Privacy Proof for the Shuffle DP Compiler

Proof of Theorem 8. Let $\vec{x}$ and $\vec{x}'$ be an arbitrary fixed pair of neighboring inputs for randomized function f, where

$$\vec{x} = (x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n),$$
$$\vec{x}' = (x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_n).$$

Let $\mathsf{Adv} \stackrel{\text{def}}{=} \{\mathsf{Adv}_\lambda\}_{\lambda\in\mathbb{N}}$ denote an arbitrary fixed PPT distinguisher. By assumption that Π_{ARE} and Π_{Sum} are secure, there exists a PPT simulator Sim. For compactness, define

$$p_{\Pi_{\mathrm{oSHARE}}}(\vec{z}) \stackrel{\text{def}}{=} \Pr\left[\mathsf{Adv}(\Pi_{\mathrm{oSHARE}}(1^\lambda, \vec{z})) = 1\right],$$
$$p_{\mathsf{Sim}}(\vec{z}) \stackrel{\text{def}}{=} \Pr\left[\mathsf{Adv}(\mathsf{Sim}(1^\lambda, 1^n, 1^{\ell_A}, f(\vec{z}))) = 1\right].$$

Then security gives $p_{\Pi_{\mathrm{oSHARE}}}(\vec{x}) \le p_{\mathsf{Sim}}(\vec{x}) + \mathtt{negl}(\lambda)$.

Regarding $\mathsf{Adv} \circ \mathsf{Sim}$ as a composing distinguisher, since f is (ε, δ)-differentially private, we then have

$$\begin{aligned} p_{\Pi_{\mathrm{oSHARE}}}(\vec{x}) &\le p_{\mathsf{Sim}}(\vec{x}) + \mathtt{negl}(\lambda) \\ &\le e^\varepsilon p_{\mathsf{Sim}}(\vec{x}') + \delta + \mathtt{negl}(\lambda) \\ &\le e^\varepsilon \left(p_{\Pi_{\mathrm{oSHARE}}}(\vec{x}') + \mathtt{negl}(\lambda)\right) + \delta + \mathtt{negl}(\lambda) \\ &\le e^\varepsilon p_{\Pi_{\mathrm{oSHARE}}}(\vec{x}') + \delta + \mathtt{negl}(\lambda). \end{aligned}$$

Combing together, by definition, we show that the scheme Π_{oSHARE} is (ε, δ)-computationally differentially private. □

G.2 A Basic General Compiler from an Arbitrary ARE Scheme

Algorithm 9 describes a basic protocol for computing a given functionality under computational differential privacy (DP) in the shuffle model. At a high level, the idea is to transform a computational ARE protocol into a shuffle protocol using a summation protocol in the shuffle model, like the Split-and-Mix Summation Protocol proposed by [IKOS06]. Concretely, each client runs the ARE protocol locally and generates the corresponding ARE encodings. For the part that encodes the

Algorithm 9 Generic ARE-based Shuffle Protocol (Π_{gSHARE})

Parameters:

- $f : \left(\{0,1\}^{\ell}\right)^{n} \mapsto \{0,1\}^{*}$: An n-party (randomized) function where each party holds an ℓ bits input.
- $\Pi_{\mathrm{ARE}} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$: An ARE scheme for f.
- ℓ_A: Input length of the encoding function $\widetilde{\mathsf{Enc}}$.
- $\Pi_{\mathrm{Sum}} = (\mathsf{Enc}_{\mathrm{Sum}}, \mathsf{Dec}_{\mathrm{Sum}})$: The shuffle Sum protocol (shown in Algorithm 4).
- m: The number of output messages from the encoding function $\mathsf{Enc}_{\mathrm{Sum}}$.

1: Compute the setup $\mathsf{pp} \leftarrow \Pi_{\mathrm{ARE}}.\widetilde{\mathsf{Setup}}(1^{\lambda}, 1^{n}, 1^{\ell_A})$

Local Encoding Function $\mathsf{Enc}(i, x_i)$

Input: Index $i \in [n]$, an element $x_i \in \mathcal{G}$, represented by an ℓ-bits string.
Output: $(x_i', \hat{x}_i)$, the encoding of input x_i.

1: Compute $(x_i', \hat{z}_i) \leftarrow \Pi_{\mathrm{ARE}}.\widetilde{\mathsf{Enc}}(\mathsf{pp}, i, x_i)$
2: Compute $\hat{x}_i \leftarrow \Pi_{\mathrm{Sum}}.\mathsf{Enc}_{\mathrm{Sum}}(i, \hat{z}_i)$
3: Output $(x_i', \hat{x}_i)$

Decoding Function $\mathsf{Dec}(x_1', \cdots, x_n', \hat{y}_1, \cdots \hat{y}_{mn})$

Input: The shuffled encodings $\hat{y}_i^1, \ldots, \hat{y}_i^{nm}$ from n clients, along with the encoding x_i' from client i.
Output: An element $y \leftarrow f(x_1, \cdots, x_n)$

1: Compute $\hat{y} \leftarrow \mathsf{Dec}_{\mathrm{Sum}}(\hat{y}_1, \cdots \hat{y}_{mn})$
2: Compute $y \leftarrow \widetilde{\mathsf{Dec}}\,(\mathsf{pp}, x_1', \cdots, x_n', \hat{y})$
3: Output y.

---

garbled circuit of the functionality, the client sent it directly to the analyzer. For the remaining part, which include information about the input and the garbled keys, each client secret-shares them across multiple messages sent to the shuffler. The analyzer then collects the shuffled shares along with the components sent directly by the clients, reconstructs the aggregated ARE encodings. It then runs the ARE evaluator, and outputs the final result of the protocol.

G.3 Proofs for Bandwidth Bounds

Proof of Theorem 26. At a high level, we will construct a circuit $\mathcal{C} \in \mathbb{C}_{n,m}$ such that its number of boundary wires satisfying the lower bound given in the statement. Our first step is to construct a graph expander $G = (V, E)$, which will be used as a skeleton for this circuit $\mathcal{C}$. Let $\varepsilon \in (0, 1)$ be a small constant and $d \geq 3$. We know that there exists a connected d-regular expander family (e.g., Ramanujan graphs) such that, when the number of vertices is larger enough, the graph in it satisfies that

$$|\partial S| \geq \varepsilon d \cdot \min\{|S|, |V \setminus S|\} \quad \text{for all } S \subseteq V,$$

where ∂S is the number of boundary edges connecting the subgraph S and $|V \setminus S|$. We construct the graph $G = (V, E)$ as the one from this family with $N = \Theta(m)$ vertices.

We then construct the Boolean circuit $\mathcal{C}$ from G by replacing each vertex with a constant-size gadget consisting of boolean gates, and each edge with a wire between the corresponding gadgets. Attach $O(n)$ input wires arbitrarily to the circuit and slightly tune each gadget to make sure the total number of gates is m.

Let $\{\mathcal{C}_1, \cdots, \mathcal{C}_n\}$ denote an arbitrary fixed balanced partition for $\mathcal{C}$ and V_i be the set of vertices corresponding gates in $\mathcal{C}_i$. Recall B_i denotes the number of boundary wires that connect to $\mathcal{C}_i$. Expansion of G gives us, for every $i \in [n]$,

$$B_i \geq \varepsilon d \cdot \min\{|V_i|, N - |V_i|\}.$$

Summing up, we have

$$\sum_{i=1}^{n} B_i \geq \varepsilon d \cdot \sum_{i=1}^{n} \min\{|V_i|, N - |V_i|\}.$$

Since, if all $|V_i| \leq N/2$, we have $\sum_{i=1}^{n} \min\{|V_i|, N - |V_i|\} = N$; otherwise at most one part exceeds $N/2$, and then the remaining parts have total size at least $(n-1) \cdot m/(2n) \geq (1 - \frac{1}{n})\frac{N}{2}$, therefore we have

$$\sum_{i=1}^{n} \min\{|V_i|, N - |V_i|\} \geq (1 - \tfrac{1}{n})N,$$

$$\sum_{i=1}^{n} B_i \geq \varepsilon d(1 - \tfrac{1}{n})N \geq \frac{\varepsilon d}{2} N.$$

By averaging, we know that there exists some $i \in [n]$ satisfying that

$$B_i \geq \frac{\varepsilon d}{2} \frac{N}{n}.$$

Since $m = \Theta(N)$, so there exists a universal constant $C \geq 1$ with $N \geq m/C$.

Finally, setting $c \stackrel{\text{def}}{=} \varepsilon d/(2C)$ completes the proof. □

Proof of Theorem 27. Let $m = |\mathcal{C}|$. We start by constructing such a n-partition. Fix a topological ordering of the gates $C_1, \ldots, C_m$ of $\mathcal{C}$. Partition this list into n consecutive blocks and denote this

blocks $\mathcal{C}_1, \ldots, \mathcal{C}_n$. We require that the first r blocks have size $q+1$ and the remaining $n-r$ blocks have size q such that $m = nq + r$.

We claim that the partition $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ is a

$$\left(0, n, 3\left\lceil \frac{|\mathcal{C}|}{n} \right\rceil\right)$$

partition because of the following.

Partition. By construction, the $\mathcal{C}_i$ are disjoint and cover all m gates, so that $\bigcup_{i\in[n]} \mathcal{C}_i = \mathcal{C}$ and $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for $i \neq j$.

Balanced size. Each part has size

$$|\mathcal{C}_i| \in \{q, q+1\} \leq (1+\gamma)\lceil \frac{m}{n} \rceil, \quad \gamma = 0.$$

Bounded boundary. The number of boundary wires is at most the total number of wires on gates in $\mathcal{C}_i$. Since we have $\lceil \frac{m}{n} \rceil$ gates in $\mathcal{C}_i$ and each gates has at most two input wires and exactly one output wire, we have

$$t \leq 3\lceil \tfrac{|\mathcal{C}|}{n} \rceil.$$

□

Proof of Theorem 29. Let $m = |\mathcal{C}|$. We start by constructing such a n-partition. For the leveled Boolean circuit $\mathcal{C}$, denote its levels as $L_1, \ldots$. Sweep levels left-to-right and greedily group consecutive levels into n blocks $\mathcal{C}_1, \ldots, \mathcal{C}_n$ so that each block's size is as close as possible to m/n. We claim that the partition $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ is a $(\frac{gn}{|\mathcal{C}|}, n, t)$-partition because of the following

Partition. By construction, the $\mathcal{C}_i$ are disjoint and cover all m gates, so that $\bigcup_{i\in[n]} \mathcal{C}_i = \mathcal{C}$ and $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for $i \neq j$.

Balanced size. By construction, each block has size at most $\frac{m}{n} + g$, because the only overshoot comes from the *last whole level* added to the block, whose size is at most g. Hence $|\mathcal{C}_i| \leq \frac{m}{n} + g$, i.e., $\gamma \leq \frac{g\,n}{m}$.

Bounded boundary. Fix a part $\mathcal{C}_i$ that is the union of whole levels from L_a through L_b. The only wires with exactly one endpoint in $\mathcal{C}_i$ cross the two interfaces between L_{a-1} and L_a, and between L_b and L_{b+1}. We then conclude that the number of its is at most $2t$. □

H Additional Details for Instantiations and Evaluation

This appendix collects material omitted from Section 6: inherited utility guarantees, formal task definitions, benchmark-circuit details, implementation details, and the partitioning objectives used in the experiments.

H.1 Utility Comparison

Because our compiler evaluates the same randomized mechanism as the corresponding central-DP algorithm, the resulting shuffle-CDP protocols inherit the utility of the central mechanisms. Table 4 summarizes this inheritance and contrasts it with known information-theoretic shuffle-DP guarantees.

For Selection, the compiled shuffle-CDP protocol improves the $O_\varepsilon(\sqrt{d})$ sample complexity of the known shuffle-DP protocol [CSU+19a] to the central-model $O_\varepsilon(\log d)$ bound. For Distinct Elements, the compiled protocol attains $\alpha = O_{\varepsilon,\delta}(\log(1/\beta))$, matching the central mechanism, while the known shuffle-DP guarantee scales as $O_{\varepsilon,\delta}(\sqrt{\min(n, |\mathcal{X}|)})$ at fixed confidence. For Linear Contextual Bandits, the compiled protocol inherits the central $O_\varepsilon(\sqrt{T})$ regret guarantee.

Task	Protocol	Privacy	Utility Metric	Result
Selection	Shuffle [CSU+19a]	DP	Sample Complexity	$O_\varepsilon(\sqrt{d})$
	Ours	CDP		$O_\varepsilon(\log d)$
	Central	DP		$O_\varepsilon(\log d)$
Distinct Elements	Shuffle [CGKM21]	DP	(α, β)-accuracy	$\alpha = O_{\varepsilon,\delta}\left(\sqrt{\min(n, \lvert\mathcal{X}\rvert)}\right)$ $\beta = 0.99$
	Ours	CDP		$\alpha = O_{\varepsilon,\delta}(\log(1/\beta))$ $\beta \in (0,1)$
	Central	DP		$\alpha = O_{\varepsilon,\delta}(\log(1/\beta))$ $\beta \in (0,1)$
Linear Contextual Bandits	Shuffle [CZ22b]	DP	Cumulative Pseudo-regret	$O_\varepsilon(T^{3/5})$
	Ours	CDP		$O_\varepsilon(\sqrt{T})$
	Central [SS18]	DP		$O_\varepsilon(\sqrt{T})$

Table 4: Utility comparison of central DP, shuffle DP, and our shuffle CDP protocols. Our utility matches the central model because the compiler evaluates the central randomized mechanism.

H.2 Task Definitions and Central Mechanisms

Selection Problem [CSU+19a]. Each user holds a binary vector $x_i \in \{0,1\}^d$. The goal is to output an index $j \in [d]$ whose total frequency is within an additive $n/10$ of the maximum:

$$\sum_{i=1}^{n} x_{i,j} \geq \max_{j' \in [d]} \sum_{i=1}^{n} x_{i,j'} - \frac{n}{10}.$$

The central solution uses the exponential mechanism [MT07], which selects j with probability proportional to

$$\exp\left(\frac{\varepsilon}{2n} \sum_{i=1}^{n} x_{i,j}\right).$$

This gives an $(\varepsilon, 0)$-DP mechanism with sample complexity $n = O(\frac{1}{\varepsilon} \log d)$, which is essentially optimal [SU17]. We compile a data-oblivious implementation of this mechanism using our compiler Π_{oSHARE}.

Distinct Elements Problem [CGKM21]. Each user holds a value $x_i \in \mathcal{X}$, and the goal is to estimate the number of distinct elements in $\mathrm{D} = \{x_1, \ldots, x_n\}$:

$$c(\mathrm{D}) = \left|y \in \mathcal{X} : \exists i \in [n] \text{ such that } x_i = y\right|.$$

An algorithm solves the (α, β)-distinct-elements problem if it outputs $\hat{c}$ satisfying

$$\Pr\left[|\hat{c} - c(\mathrm{D})| \leq \alpha\right] \geq 1 - \beta.$$

In the central model, the distinct count query has global sensitivity 1, so adding Laplace or Gaussian noise with scale $O(1/\varepsilon)$ gives an $(\varepsilon, 0)$-DP mechanism with error $O(\frac{1}{\varepsilon} \log(1/\beta))$ with probability at least $1 - \beta$. We implement the query using an oblivious sorting circuit and compile it with Π_{oSHARE}.

Linear Contextual Bandits [CZ22b]. In each round $t \in [T]$, a user arrives with context c_t, the learner chooses an action $a_t \in \mathcal{X}$, and observes reward $y_t \in [0,1]$ with conditional mean

$$\mathbb{E}\left[y_t \mid c_t, a_t\right] = \langle \theta^*, \phi(c_t, a_t) \rangle,$$

where $\phi(c_t, a_t) \in \mathbb{R}^d$ and $\theta^* \in \mathbb{R}^d$. The cumulative pseudo-regret is

$$\mathsf{Reg}(T) \stackrel{\text{def}}{=} \sum_{t=1}^{T} \max_{a \in \mathcal{X}} \langle \theta^*, \phi(c_t, a) \rangle - \langle \theta^*, \phi(c_t, a_t) \rangle.$$

The central DP algorithm maintains sufficient statistics

$$u_t = \sum_{\tau \le t} \phi(c_\tau, a_\tau) y_\tau, \qquad V_t = \lambda I + \sum_{\tau \le t} \phi(c_\tau, a_\tau) \phi(c_\tau, a_\tau)^\top,$$

and releases them with a tree-based mechanism and Laplace noise. With high probability, the regret is $O_\varepsilon(\sqrt{T})$ [SS18]. We compile the data-oblivious circuit for the noised statistics using Π_{oSHARE}.

Stochastic Gradient Descent. In supervised learning, given $\mathrm{D} = \{x_1, \ldots, x_n\}$ and loss $\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\theta; x_i)$, SGD updates $\theta_{t+1} \leftarrow \theta_t - \eta_t g_t$. The DP variant clips gradients and adds noise as in Algorithm 10.

Algorithm 10 Differentially private SGD [ACG+16][Alg. 1]

Input: Samples $\{x_1, \ldots, x_n\}$, loss $\mathcal{L}(\theta) = \frac{1}{N} \sum_i \mathcal{L}(\theta, x_i)$, learning rate η_t, noise scale σ, expected group size L, gradient norm bound C.
Initialize θ_0 randomly.
for $t \in [T]$ **do**
    Sample minibatch L_t with probability $q = L/n$.
    for each $x_i \in L_t$ **do**
        $g_t(x_i) \leftarrow \nabla_\theta \mathcal{L}(\theta_t, x_i)$.
    end for
    $\bar{g}_t(x_i) \leftarrow g_t(x_i) / \max\left(1, \frac{\|g_t(x_i)\|_2}{C}\right)$.
    $\tilde{g}_t \leftarrow \frac{1}{L}\left(\sum_i \bar{g}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$.
    $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{g}_t$.
end for
Output θ_T.