Communication-Efficient Private Join and Compute over Distributed Input Sets

Yunqing Sun
Northwestern University
yunqing.sun@northwestern.edu

Xinran Cai*
Tsinghua University
caixr22@mails.tsinghua.edu.cn

Hanlin Liu✉†
Northwestern University
hanlin.liu@northwestern.edu

Xiao Wang
Northwestern University
wangxiao@northwestern.edu

Wei Dong
Nanyang Technological University
wei_dong@ntu.edu.sg

Abstract

Private Join and Compute (PJC) enables two parties to compute aggregates over matching records from their private datasets. In this work, we focus on the inner-product variant of PJC, which computes the inner product over matching records from their private datasets. It has important applications such as privacy-preserving ad conversion measurement. However, existing PJC protocols assume each party holds the entire dataset, which is often unrealistic in practice, where relevant datasets are distributed across multiple data owners. No existing PJC protocols directly support distributed input sets across multiple clients, while straightforward generic approaches introduce substantial overhead.

We propose an efficient approximate PJC protocol for distributed input sets while keeping the communication sublinear in the input size. Our protocol works in the semi-honest setting and uses two non-colluding servers that learn nothing beyond the final approximation. The core technical contribution is a novel adaptation of the Gödel Prize-winning AMS sketch redesigned for efficient evaluation under fully homomorphic encryption. Concretely, we show a new structured randomness that can be homomorphically generated from short seeds using just 3 levels of multiplication while maintaining the best plaintext accuracy bound. Based on our optimized implementation, clients can insert each input element into an encrypted sketch in 30 ms, which has a size of 250 KB, independent of input size. The servers can recover the final output within seconds, orders of magnitude faster than the generic method.

1 Introduction

Private Join and Compute (PJC) [LPR+21, WY21] is an important cryptographic primitive that enables two parties to perform data analytics over matching records without revealing their individual data. It was originally proposed by Google [Goo19] for privacy-preserving ad conversion measurement between advertisers and websites. A basic form of PJC is private set intersection sum (PSI-sum) [MRR20, IKN+20, CILO22], which reveals the sum of values associated with records in the intersection. A more expressive variant is PSI-inner product [LPR+21, WY21, CHI+23, KLS24, SJ23], which privately joins two datasets on a key and computes the inner product of the aligned values. Throughout this paper, we follow the established definition of [Goo19, LPR+21] and use PJC to denote the inner-product (and sum) variant.

All of the above PJC protocols consider two parties, each holding its entire dataset. A typical example from ad conversion measurement involves two parties: 1) an advertisement vendor (AV), which holds a dataset of (*customer_id*, *duration*) pairs, where *duration* denotes the amount of time a customer was exposed to an ad; 2) a transaction data provider (TDP), which holds a dataset of (*customer_id*, *spending*) pairs, where *spending* denotes the amount spent by the customer after viewing the ad. The desired computation can be expressed as the following SQL query:

*Work done while visiting Northwestern University.
†Corresponding author.

```
SELECT SUM(TDP.spending * AV.duration)
FROM AV
INNER JOIN TDP
    ON AV.cid = TDP.cid;
```

However, applying existing PJC protocols in this setting relies on a strong assumption: there exist two parties that actually hold all advertisement records and all transaction records, respectively. In practice, this assumption is often unrealistic. For example, after seeing an ad, a customer may purchase a product from Amazon, Target, or other retailers. In that case, the transaction data is partitioned across multiple mutually untrusted parties that are unwilling to share their customer records with one another. Similarly, advertisement records may also be split across multiple vendors or jurisdictions, where data sharing is restricted or prohibited.

It is therefore desirable to have a PJC protocol where the two input sets are distributed over many clients. However, extending existing PJC protocols to this setting is non-trivial, since they inherently assume that each party holds its entire input. There are two natural candidate solutions, but both fall short:

- **PJC Decomposition.** Since $\langle \sum_i \boldsymbol{x}_i, \sum_j \boldsymbol{y}_j \rangle = \sum_{i,j} \langle \boldsymbol{x}_i, \boldsymbol{y}_j \rangle$, we can decompose PJC over distributed input sets into a quadratic number of smaller PJC computations. Thus, one can support PJC over distributed input sets by each client running a PJC with every client for the other set. However, each pairwise PJC reveals its output, so this approach leaks every pairwise inner product to the servers; this is costly and leaks far more information than the target aggregate.
- **Server-based Generic 2PC.** Alternatively, we can let all clients secret-share their inputs to two non-colluding servers, who can then run a generic 2PC to aggregate and compute the final PJC values. The cost of 2PC would be quasi-linear to the sum of the sizes of all client inputs, which also leads to prohibitive cost.

1.1 Contributions

In this paper, we present a practical system for approximate PJC in which the two input sets can be contributed by many clients. Figure 1 illustrates the overall workflow of our protocol. We assume a cryptographic service provider (CSP) during the initialization and output phases; otherwise, only the aggregator server (Agg) needs to be online, and each client sends a single message to the aggregator.

1. **General framework for PJC over distributed sets.** We design a general framework for approximate PJC over distributed input sets. We observe that PJC is most useful for computing private aggregates and statistics, where the underlying data can naturally be viewed as samples from an underlying distribution and thus approximation is often acceptable. This motivates the use of the AMS sketch [AMS96], which enables approximate PJC in plaintext with very small communication. Building on this observation, we let each client use encrypted common randomness to homomorphically compute a local encrypted sketch and send a single message to Agg, which efficiently aggregates all client contributions. This design ensures that the randomness used by the sketch is never revealed, while keeping both client-server and server-server communication low. We detail this framework and protocol in Section 2.2 and Section 5.
2. **HE-friendly sketch construction.** A naive instantiation of the AMS sketch incurs prohibitive cost when homomorphically expanding the randomness required for sketch evaluation. To obtain a practical protocol, in Section 4.2 we develop a new AMS-style sketch construction that provably achieves the same level of accuracy while using only structured randomness that can be evaluated in merely *3 levels of homomorphic multiplication*. Our new sketch requires a significantly more intricate analysis, since the structured randomness we use is weaker than the weakest previously analyzed variant of the AMS sketch, which we formalize as a Δ-inadmissible distribution (Definition 2).
3. **Protecting the output using differential privacy for free.** Prior work has shown that the output of PJC may already be vulnerable to membership-inference attacks [FT25]. We therefore augment our framework with an efficient mechanism for releasing a differentially private output, incurring little additional protocol overhead while preserving output utility, which we develop in Section 6.

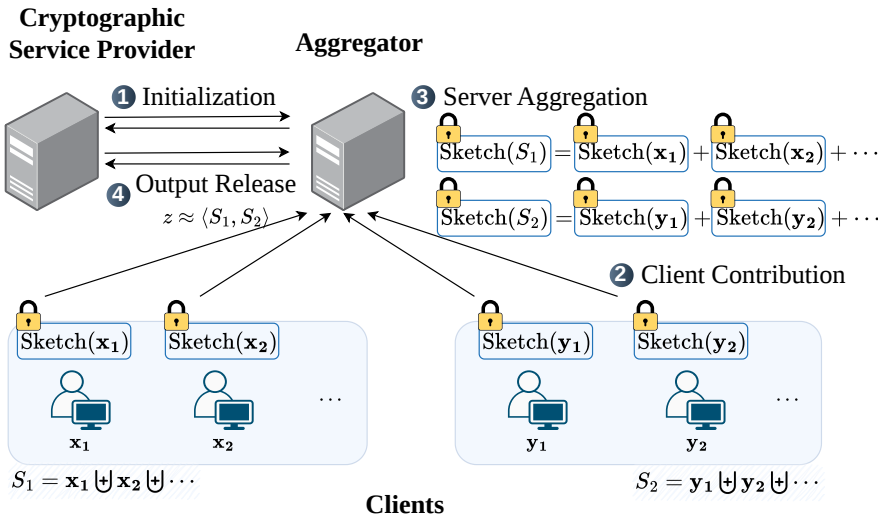


Figure 1: System model of Private Join and Compute over Distributed Input Sets. Two non-colluding servers, the Aggregator and the Cryptographic Service Provider (CSP), serve many clients. ① During initialization, the two servers jointly generate encrypted sketching randomness; the CSP can then go offline. ② Each client P_i locally computes an encrypted sketch of its input vector ($\boldsymbol{x}_i$ for set S_1, or $\boldsymbol{y}_i$ for set S_2) and sends it to the Aggregator in a single message. ③ The Aggregator homomorphically aggregates all client sketches into $[\![\text{Sketch}(S_1)]\!]$ and $[\![\text{Sketch}(S_2)]\!]$. ④ Finally, the two servers jointly compute the approximate inner product $z \approx \langle S_1, S_2 \rangle$ via threshold decryption and 2PC.

4. **Implementation and benchmark.** We implement and benchmark our protocol to evaluate its accuracy, client scalability, and server performance in Section 7. Compared with prior solutions, our protocol is well-suited to a wide range of deployment settings. In particular, the server cost is under 10 seconds regardless of the number of clients; the client's communication is non-interactive and only about 250 KB for any set size; the client's per-element computation is around 30 ms but is embarrassingly parallelizable.

Our framework is more general than existing PJC protocols and can also realize the standard two-party PJC functionality. In particular, suppose P_1 and P_2 hold datasets $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Then P_1 can act as both CSP and a client holding $\mathcal{S}_1$, while P_2 can act as both Agg and a client holding $\mathcal{S}_2$. This yields a two-party approximate PJC protocol with sublinear communication.

2 Technical Overview

2.1 Security Model

We start with our system setup and security model, also depicted in Figure 1. Our framework assumes two non-colluding servers: an Aggregator (Agg) and a Cryptographic Service Provider (CSP), serving w clients $P_1, \ldots, P_w$. We adopt the CSP+Aggregator architecture introduced by Nikolaenko et al. [NIW+13] for privacy-preserving matrix factorization: CSP only needs to be online to interact with Agg during the initialization phase and the finalization phase to obtain output; once setup completes, CSP can go offline and clients send a single message directly to Agg. We allow a semi-honest adversary to corrupt one of the servers (CSP or Agg) and a subset of the clients; the corrupted server learns nothing beyond what the ideal functionality reveals to it (in particular, when Agg is corrupted, it learns the final approximate output). Our proposed protocol securely instantiates the ideal functionality in Figure 2. We stress that this ideal functionality outputs a *randomized approximation* of the inner product, not its exact value: the output is $z \leftarrow F(\boldsymbol{v}_1, \boldsymbol{v}_2)$ for a randomized F carrying a provable accuracy guarantee (Section 4.3). Consequently, any output leakage inherent to PJC is with respect to this noisy estimate rather than the exact aggregate. We

Functionality $\mathcal{F}_{\mathsf{PJC}}$

Participants: Two servers Agg, CSP, and w parties $P_1, \ldots, P_w$.
Public function: A randomized function F that outputs an inner product approximation of the input vectors.

1. **Initialization.** Upon receiving ($\mathsf{Initialization}$) from Agg and CSP, initialize sets $\mathcal{S}_1 := \emptyset$ and $\mathcal{S}_2 := \emptyset$.
2. **Input.** Upon receiving from P_i input $(\mathsf{input}, p, \mathcal{S})$, where $p \in \{1, 2\}$ and $\mathcal{S}$ is a multiset over domain $[N]$, the functionality updates $\mathcal{S}_p := \mathcal{S}_p \biguplus \mathcal{S}$ as multi-set union. Send p to Agg.
3. **Output.** Upon receiving (Output) from Agg and CSP, the functionality computes a frequency vector $\boldsymbol{v}_1$ (resp. $\boldsymbol{v}_2$) of set $\mathcal{S}_1$ (resp. $\mathcal{S}_2$), computes and sends to Agg an approximation $z \leftarrow F(\boldsymbol{v}_1, \boldsymbol{v}_2)$.

Figure 2: The ideal functionality for private join and compute over distributed input sets.

further reduce this leakage by supporting differentially private outputs (Section 6), where Laplace noise is added to the encrypted sketches at negligible additional cost.

2.2 A New Framework for Distributed PJC

Motivating Example. Given a multi-set $\mathcal{X}$ over $[N]$, one can always encode it into a frequency vector $\boldsymbol{x} \in \mathbb{Z}^N$, where x_i denotes the multiplicity of element i. Viewing this multiplicity as the payload associated with element i, the inner product of two encoded multisets, $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i \in [N]} \boldsymbol{x}_i \boldsymbol{y}_i$ is exactly PJC: each common element contributes the product of its multiplicities in the two multisets, while an element held by only one party contributes zero. We start from a simple observation that this inner product can be achieved in plaintext with small communication using sketches. We take the AMS sketch [AMS96] as an example. The basic AMS sketch is defined as $X = \langle \boldsymbol{x}, \boldsymbol{r} \rangle$ with a uniformly sampled $\boldsymbol{r} \leftarrow \{-1, +1\}^N$. It is shown that X^2 serves as an unbiased estimator of $\|\boldsymbol{x}\|_2^2$, with variance at most $O(\|\boldsymbol{x}\|_2^4)$. To improve accuracy to $\mathcal{E}$, one can take the average of $k = O(1/\mathcal{E}^2)$ independent copies. Similarly, let $X = \langle \boldsymbol{x}, \boldsymbol{r} \rangle$ and $Y = \langle \boldsymbol{y}, \boldsymbol{r} \rangle$ be AMS sketches computed using the same randomness $\boldsymbol{r}$. Then the product XY is an unbiased estimator of the inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ and has variance bounded by $O(\|\boldsymbol{x}\|_2^2 \|\boldsymbol{y}\|_2^2)$.

AMS is a linear sketch in that computing the sketch X only involves linear operations over $\boldsymbol{x}$, given a fixed $\boldsymbol{r}$. Therefore, distributed PJC can be achieved with small communication in plaintext as follows. A server publishes a random vector $\boldsymbol{r}$ as described above. Each client with input vector $\boldsymbol{x}_i$ computes and sends $X_i = \langle \boldsymbol{x}_i, \boldsymbol{r} \rangle$ to the server. The server aggregates these values as $X = \sum_{i \in \mathcal{S}_1} X_i$, which corresponds to the sketch of the multiset union of all inputs, as $\sum_{i \in \mathcal{S}_1} X_i = \langle \sum_{i \in \mathcal{S}_1} \boldsymbol{x}_i, \boldsymbol{r} \rangle$. Similarly, for vector $\boldsymbol{y}_j$, the server obtains $Y = \sum_{j \in \mathcal{S}_2} Y_j$ from sketches $Y_j = \langle \boldsymbol{y}_j, \boldsymbol{r} \rangle$. Finally, the server can compute an approximation of the inner product $XY \approx \langle \sum_{i \in \mathcal{S}_1} \boldsymbol{x}_i, \sum_{j \in \mathcal{S}_2} \boldsymbol{y}_j \rangle$, optionally averaging over k independent repetitions to improve accuracy.

The use of the same randomness $\boldsymbol{r}$ is crucial for correctness, but is also a curse when we want to turn it into a privacy-preserving protocol. First of all, since $\boldsymbol{r}$ is public and all parties submit their sketch to the server, the server can easily perform a dictionary attack on each client's set and even recover it when it is small. One could possibly use two-party computation to securely perform the work of the server so that only the final output is revealed but even given just $\boldsymbol{r}$ and the final output, a dictionary attack can still be performed on the union of clients' sets.

Our Framework. In this paper, we decide to keep $\boldsymbol{r}$ private all the time so that none of the parties know the randomness and thus prevent the above attack. It roughly works as follows, ignoring some technical details.

1. **Initialization.** Two servers perform a threshold key generation protocol for a homomorphic encryption (HE) scheme so that at the end of the protocol, they have the public key and the evaluation key. Agg initializes two collections of encrypted sketches $\{[\![c_1^\ell]\!]\}^{\ell \in [k]}$ and $\{[\![c_2^\ell]\!]\}^{\ell \in [k]}$, where each entry is a fresh

encryption of 0, representing $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Two servers jointly generate encryptions of k random seeds $\{[\![\boldsymbol{s}^\ell]\!]\}^{\ell\in[k]}$, where the seeds are jointly generated and hidden from both servers, and publish them to all clients.

2. **Client Contribution.** For each client with input vector $\boldsymbol{x}$, the client homomorphically computes the ciphertext $[\![X^\ell]\!] = \langle G([\![\boldsymbol{s}^\ell]\!]), \boldsymbol{x}\rangle$, where $G : \{0,1\}^n \to \{-1,1\}^N$ is an expanding function, e.g., a pseudorandom generator (PRG). The client sends $(p, \{[\![X^\ell]\!]\}^{\ell\in[k]})$ to Agg to indicate that it will contribute to set $\mathcal{S}_p$, where $p \in \{1,2\}$.

3. **Server Aggregation.** Upon receiving $(p, \{[\![X^\ell]\!]\}^{\ell\in[k]})$ from a client, Agg homomorphically update $[\![c_p^\ell]\!] \leftarrow [\![c_p^\ell]\!] + [\![X^\ell]\!]$ for all $\ell \in [k]$, which effectively performs a multiset union.

4. **Output Release.** Agg and CSP perform threshold decryption on encrypted sketches $\left\{[\![c_1^\ell]\!], [\![c_2^\ell]\!]\right\}^{\ell\in[k]}$ to obtain additive secret shares of the corresponding plaintexts. They then invoke a generic $\mathcal{F}_{\mathsf{2PC}}$ functionality to securely compute $Z = \sum_{\ell\in[k]} c_1^\ell c_2^\ell$ and the final result is Z/k.

This protocol is highly efficient in communication: the communication between the two servers, the size of the published seeds, and the message from each client are all $O(k)$. The advantage in communication crucially relies on the use of homomorphic encryption to expand the seed. Next, we discuss the exact computational cost of this step under various instantiations.

2.3 Why Classic Constructions Fail

The above framework yields secure, sublinear-communication protocols, but the computational cost of expanding randomness can be high. We discuss some plausible choices. As a summary, we have the flexibility in choosing G, but the output should be $\{-1,+1\}^N$ for use in the next step. In practice, it suffices to generate outputs in $\{0,1\}^N$ and map them to $\{-1,+1\}^N$ via $\boldsymbol{b} \mapsto 1 - 2\boldsymbol{b}$. As a first optimization, if each output bit of $G(\boldsymbol{s})$ can be independently computed, one can compute

$$[\![X]\!] = \langle G([\![\boldsymbol{s}]\!]), \boldsymbol{x}\rangle = \sum_{i\in\mathcal{X}} G([\![\boldsymbol{s}]\!])[i],$$

where we avoid fully expanding to an N-length vector and instead compute only the required randomness. This property is satisfied by, e.g., local PRGs and the counter-mode instantiation from PRF. We reviewed classic ideas from both theory and cryptography, but we found that existing possible instantiations of G all suffer from large seed sizes and/or high computational cost for homomorphically expanding the seed.

Goldreich PRG. Homomorphic evaluation of PRGs has recently been explored in a different context [ZLY+26], using Goldreich-style random local PRGs [Gol00] instantiated with $\mathsf{XOR\text{-}MAJ}_{4,7}$. Such PRGs require a seed of length $\sqrt{N}$ to produce $O(N)$ bits; thus, this instantiation requires a seed size of $\Omega(\sqrt{N})$. It is also conjectured that reducing the seed length would require much deeper circuits.

PRG from primal LPN. Another plausible solution discussed in [ABBS25] was to use LPN-based PRGs. The output is defined as

$$G(\boldsymbol{s}, i_1, \ldots, i_\ell) = \mathbf{A}\cdot\boldsymbol{s} + \boldsymbol{e}\ ,$$

where $\boldsymbol{s}$ is a short random seed, $\mathbf{A}$ is a public sparse matrix over $\mathbb{F}_2$, and the regular noise vector $\boldsymbol{e} = (\mathsf{OneHot}(i_1), \ldots, \mathsf{OneHot}(i_\ell))$ is the concatenation of small vectors of length N/ℓ, each as the one-hot encoding of index i. The seed size of this LPN-based PRG is $|\boldsymbol{s}| + |i_1| + \cdots + |i_\ell|$, which is lower bounded by $\Omega(\sqrt{N})$ due to the security requirement [LWYY24] that $|\boldsymbol{s}|\ell > N$.

PRG from dual LPN. A related approach is to use a dual-LPN-based PRG, whose output is defined as

$$G(i_1, \ldots, i_\ell) = \mathbf{H}\cdot\boldsymbol{e}\ ,$$

where $\mathbf{H}$ is a uniformly random parity-check matrix over $\mathbb{F}_2$, and $\boldsymbol{e} = (\mathsf{OneHot}(i_1), \ldots, \mathsf{OneHot}(i_\ell))$. This construction has a small seed, but the main drawback is the prohibitively large circuit size/depth required to compute $\mathbf{H}\cdot\boldsymbol{e}$ given a dense public matrix $\mathbf{H}$, especially in the context of HE, where Free-XOR is not

available. In this setting, Boolean XOR gates must be implemented using arithmetic circuits, causing each XOR operation to incur the same cost as an AND gate.

4-wise independent randomness. It is well-known that the AMS sketch can keep the asymptotic accuracy even when $\boldsymbol{r}$ is 4-wise independent uniform, which means that any subset of 4 values is indistinguishable from uniform. The classical instantiation is to pick uniform values $a, b, c, d \in \mathbb{Z}_p$ for some prime $p \geq N$ and define $G : [N]^4 \to \{0,1\}^N$ such that

$$G(a,b,c,d)[i] = (a \cdot i^3 + b \cdot i^2 + c \cdot i + d) \bmod p \bmod 2.$$

However, computing $G(a,b,c,d)[i]$ can be slow in practice under HE, since the computation is over $\mathbb{Z}_p$ and requires a modular operation from $\mathbb{Z}_p$ to $\mathbb{Z}_2$ for each output bit, which is very expensive.

2.4 Warm-Up Solution: HE-Friendly AMS Sketch

As discussed above, we want a G that is friendly for HE evaluation and has sublinear seed size. Here, we first present a warm-up solution that generates 4-wise independent randomness but only requires **2** homomorphic multiplications for each element. We use this to show that an HE-aware design can significantly improve the state-of-the-art solution.

This warm-up solution has a seed size of $4\sqrt{N}$ bits consisting of three independent uniform Boolean vectors $\boldsymbol{c} \in \{0,1\}^{\sqrt{N}}$, $\boldsymbol{d} \in \{0,1\}^{\sqrt{N}}$, $\boldsymbol{e} \in \{0,1\}^{2\sqrt{N}}$, and we define the expansion as

$$G(\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e})[i \cdot \sqrt{N} + j] = \boldsymbol{c}[i] \oplus \boldsymbol{d}[j] \oplus \boldsymbol{e}[i+j],$$

for any $i, j \in [\sqrt{N}]$.

We now show that $G(\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e})$ is four-wise independent. Since every bit of the output can be viewed as XOR of some number of uniform and independent bits, we just need to show that for any four distinct indices (i_1, j_1), (i_2, j_2), (i_3, j_3), (i_4, j_4), no subset of outputs deterministically XORs to zero. That is,

$$\bigoplus_{k=1}^{\ell} G(\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e})[i_k \cdot \sqrt{N} + j_k] \equiv 0 \text{ for } \ell = 2, 3, 4 \ .$$

The two-output case is trivial. In the three-output case, three outputs of G involve 9 bits (possibly with repetitions), which cannot XOR to zero deterministically. For the XOR of four outputs to be identically zero, the first components (i_1, i_2, i_3, i_4) would need exactly two distinct values, each appearing twice, and similarly for the second components (j_1, j_2, j_3, j_4). This implies that the four relevant positions of $\boldsymbol{e}$, namely $\{i_k + j_k\}_{k \in [4]}$, must contain at least three distinct values and therefore cannot XOR to zero deterministically. Therefore, G is four-wise independent. We provide a complete proof in Appendix B.

This solution is faster than all instantiations in Section 2.3 as it only requires 2 homomorphic multiplications per element in the set. However, the seed size is $O(\sqrt{N})$, which is still undesirable. It is possible to generalize this approach to further reduce the seed size, but its concrete computational cost increases quickly; therefore, we take a different route in our final solution.

2.5 AMS Sketch with Imperfect Randomness

If the solution above is about designing a better randomness generation process that is also HE-friendly at the same time, our final solution takes a step further in co-designing between sketching algorithm and cryptography.

A Cheap and Unbiased Estimator. Our starting point is the LPN-based PRG solution:

$$G(\boldsymbol{s}, i_1, \ldots, i_\ell) = \mathbf{A} \cdot \boldsymbol{s} + \boldsymbol{e},$$

where $\boldsymbol{e} = \big(\mathsf{Unary}(i_1), \ldots, \mathsf{Unary}(i_\ell)\big)$. This solution is overkill, since the AMS sketch does not require full randomness. Thus, we try to explore ways to reduce its structure while maintaining the accuracy of the sketch. We do not like $\boldsymbol{e}$, which is expensive to expand in HE, and we do not like the fact that the seed size cannot be small.

k	100	200	400	1000	2500	10000
AMS + uniform r	0.055	0.041	0.028	0.017	0.011	0.006
AMS + our r	0.056	0.039	0.028	0.017	0.011	0.005

Table 1: **Approximation accuracy comparison under different k.** Accuracy is measured as median over 1000 executions.

To this end, we construct an expansion function as follows. We first sample a short private seed $\boldsymbol{s} \in \{0,1\}^n$ uniformly at random, and then independently sample a public matrix $\mathbf{A} \in \{0,1\}^{N \times n}$ such that each row has Hamming weight $\mathsf{HW}(\mathbf{A}_i) = d$ for some constant d, and $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$. The output random vector $\boldsymbol{r} \in \{-1,+1\}^N$, as required for the AMS sketch, is then obtained via

$$\boldsymbol{r} = (-1)^{\mathbf{A} \cdot \boldsymbol{s} \bmod 2} .$$

Note that this construction is computationally cheap as the number of homomorphic multiplications to compute any bit of $\mathbf{A} \cdot \boldsymbol{s}$ is linear in d. More precisely, since $\mathbf{A}$ is public, we can compute in HE

$$\boldsymbol{r}[i] = 1 - 2 \cdot \Big(\bigoplus_{j \in [n] \text{s.t.} \mathbf{A}_i[j]=1} \boldsymbol{s}[j] \Big).$$

Moreover, the solution is unbiased since we require that all rows in matrix $\mathbf{A}$ are different. Indeed, we have

$$\mathbb{E}[X^2] = \mathbb{E}\Big[\Big(\sum_i \boldsymbol{x}[i] \cdot r[i] \Big)^2 \Big] = \sum_i \boldsymbol{x}[i]^2 + \sum_{i \neq j} \boldsymbol{x}[i]\boldsymbol{x}[j] \mathbb{E}\Big[\boldsymbol{r}[i]\boldsymbol{r}[j] \Big].$$

In our case, $\boldsymbol{r} = (-1)^{\mathbf{A} \cdot \boldsymbol{s}}$ is not fully independent: although they are pair-wise independent, higher-order correlations remain.

$$\mathbb{E}\Big[\boldsymbol{r}[i]\boldsymbol{r}[j] \Big] = \mathbb{E}\Big[(-1)^{\mathbf{A}_i \cdot \boldsymbol{s}} \cdot (-1)^{\mathbf{A}_j \cdot \boldsymbol{s}} \Big] = \mathbb{E}\Big[(-1)^{(\mathbf{A}_i \oplus \mathbf{A}_j) \cdot \boldsymbol{s}} \Big].$$

Since $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$, then by the pairwise independence of $\boldsymbol{s}$, we have $\mathbb{E}\big[\boldsymbol{r}[i]\boldsymbol{r}[j]\big] = 0$. This means that X^2 is an unbiased estimator for $\|\boldsymbol{x}\|_2^2$.

Bounding Variance. Experimentally, we observed that the above construction works and achieves an accuracy comparable to that of the original AMS sketch! As shown in Table 1, the accuracy of this simplified structure is empirically identical to that achieved with uniform $\boldsymbol{r}$. To complete the analysis, it remains to establish a provable accuracy bound for this cryptography-inspired design.

The accuracy of the AMS sketch fundamentally follows from its tightly controlled variance. Concretely, since the first term of $X^2 = \sum_i \boldsymbol{x}[i]^2 + \sum_{i \neq j} \boldsymbol{x}[i]\boldsymbol{x}[j]\boldsymbol{r}[i]\boldsymbol{r}[j]$ is a constant, we have

$$\begin{aligned} \mathsf{Var}[X^2] &= \mathsf{Var}\Big[\sum_{i \neq j} \boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{x}[i]\boldsymbol{x}[j] \Big] = \mathbb{E}\Big[\Big(\sum_{i \neq j} \boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{x}[i]\boldsymbol{x}[j] \Big)^2 \Big] \\ &= \sum_{i \neq j, k \neq \ell} \mathbb{E}\Big[\boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{r}[k]\boldsymbol{r}[\ell] \Big] \boldsymbol{x}[i]\boldsymbol{x}[j]\boldsymbol{x}[k]\boldsymbol{x}[\ell]. \end{aligned}$$

Note that $\mathbb{E}\big[\boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{r}[k]\boldsymbol{r}[\ell]\big] = 0$ if $\mathbf{A}_i \oplus \mathbf{A}_j \oplus \mathbf{A}_k \oplus \mathbf{A}_\ell \neq \mathbf{0}$ and equals 1 otherwise; hence, all terms that vanish in expectation can be discarded. Consequently, we obtain

$$\mathsf{Var}[X^2] = \sum_{\boldsymbol{v} \neq \mathbf{0}} \sum_{\substack{\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v} \\ \mathbf{A}_k \oplus \mathbf{A}_\ell = \boldsymbol{v}}} \boldsymbol{x}[i]\boldsymbol{x}[j]\boldsymbol{x}[k]\boldsymbol{x}[\ell] = \sum_{\boldsymbol{v} \neq \mathbf{0}} \Big(\sum_{\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}} \boldsymbol{x}[i]\boldsymbol{x}[j] \Big)^2 .$$

Applying the Cauchy–Schwarz inequality yields

$$\mathsf{Var}[X^2] \leq \sum_{\boldsymbol{v} \neq \mathbf{0}} N(\boldsymbol{v}) \sum_{\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}} \boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2 ,$$

where $N(\boldsymbol{v}) \stackrel{\text{def}}{=} \#\{(i,j) \in [N]^2 \mid \mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}\}$ denotes the count of pairs corresponding to a fixed $\boldsymbol{v}$. This count is a rare-event variable with a small expected value $o(1)$, and exhibits sub-exponential tail behavior. By taking a union bound over all possible $\boldsymbol{v}$, we obtain that $\max_{\boldsymbol{v}\neq 0} N(\boldsymbol{v}) \leq \log\log(n)$ with overwhelming probability (detailed in Theorem 4). Hence,

$$\mathsf{Var}[X^2] \leq \log\log(n) \cdot \sum_{\boldsymbol{v}\neq \mathbf{0}} \sum_{\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}} \boldsymbol{x}[i]^2 \boldsymbol{x}[j]^2 \leq \log\log(n) \|\boldsymbol{x}\|_2^4 \ .$$

This establishes that the variance remains tightly controlled, which underlies the accuracy of the cryptography-inspired AMS sketch.

Moreover, the sparsity of $\mathbf{A}$ keeps the matrix-vector product highly efficient, as each output entry depends on only a few seed bits, requiring only a small number of modulo-2 additions per entry. This property is particularly advantageous for evaluation under HE. Consequently, this approach enables scalable generation of $\boldsymbol{r}$ without incurring the prohibitive overhead of fully independent random bits or of directly evaluating a PRG under FHE. For a realistic choice of $d = 8$, the per-element evaluation cost is 25.34 ms and is highly parallelizable, demonstrating high efficiency gains due to the introduced sparsity.

Connection to Subsampled-Hadamard Sketching. In hindsight, our construction turns out to have a clean interpretation in the language of randomness-efficient sketching. Recall a $2^n \times 2^n$ Hadamard matrix $\mathbf{H}$ has entries $\mathbf{H}[\boldsymbol{i}, \boldsymbol{j}] = (-1)^{\langle \boldsymbol{i}, \boldsymbol{j} \rangle}$, where $\boldsymbol{i}, \boldsymbol{j} \in \{0,1\}^n$. A line of work in compressed sensing [KW11, HR16] showed that when sketching a vector of dimension 2^n, one can build an AMS sketch by picking a random row index $\boldsymbol{s} \in \{0,1\}^n$ and using the entire row $\mathbf{H}_{\boldsymbol{s}} \in \{-1,+1\}^{2^n}$ as the sketch vector—known as *subsampled-Hadamard sketching*.

Our construction uses exactly the same Hadamard matrix and the same row-sampling step, but reads off only N entries of the chosen row at positions selected by a fixed sparse pattern, with $N \ll 2^n$. Concretely, fix N distinct points $\mathbf{A}_1, \ldots, \mathbf{A}_N \in \{0,1\}^n$, each of Hamming weight d, and set $\boldsymbol{r}[j] = \mathbf{H}[\boldsymbol{s}, \mathbf{A}_j]$ for $j = 1, \ldots, N$. The sketch is computed without materializing $\mathbf{H}$: each entry $\boldsymbol{r}[j] = (-1)^{\langle \boldsymbol{s}, \mathbf{A}_j \rangle}$ is the XOR of just d seed bits selected by the support of $\mathbf{A}_j$, which is what makes the sketch cheap under HE. We show that, with overwhelming probability, this structured-sparse pattern incurs at most a $\log\log n$ multiplicative overhead in sketch variance relative to standard AMS. Empirically, as shown in Figures 5b and 5c, the observed error closely matches that of standard AMS, suggesting that this overhead is negligible in practice. To our knowledge, this regime—where the row is sampled uniformly but the entries are restricted to a sparse subset of positions—has not been studied in the classical sketching literature, as it arises naturally only in cost models like HE evaluation where sparse linear maps are dramatically cheaper than dense ones.

Comparison to prior AMS derandomization. Braverman et al. [BCL+10] and related work [IM08] study *product domains* $[n]^k$ and show that AMS does not need a 4-wise independent hash over $[n]^k$: the product of k 4-wise independent hashes over $[n]$ suffices, even though this product is *not* itself 4-wise independent over $[n]^k$ (for $k \geq 2$). Their relaxation is thus along the *domain dimension*, and at $k = 1$ it reduces to standard 4-wise-independent AMS. Our setting is exactly this $k = 1$ domain, where we show that even there, 4-wise independence is not needed.

2.6 Supporting Differentially Private Output

Recall from Section 2.1 that the functionality outputs a randomized approximation of the dot product, not the exact value. To further reduce the impact of any output leakage, we show that our protocol can achieve differentially private output almost for free. The high-level idea is to add a small amount of Laplace noise to the aggregated encrypted sketches before computing the final estimate in 2PC. We show that, under modest assumptions on the input norms, this noise is small enough that it does not meaningfully degrade the output quality. Moreover, the noise can be added to the encrypted sketches outside of MPC, so the servers do not pay the cost of sampling DP noise inside a 2PC protocol. See Section 6 for details.

3 Preliminaries

3.1 Notations

$\mathsf{HW}(\boldsymbol{a})$ denotes the Hamming weight of the vector $\boldsymbol{a}$. We use $\boldsymbol{a}[i]$ to denote the i-th element of the vector $\boldsymbol{a}$. We use $\mathsf{supp}(\boldsymbol{a})$ to denote the support of $\boldsymbol{a}$, i.e., $\mathsf{supp}(\boldsymbol{a}) := \{\, i \mid \boldsymbol{a}[i] \neq 0 \,\}$. $\mathbf{1}_n$ denotes the all-ones vector of length n. We use $[\![a]\!]$ to denote the ciphertext from the encryption of a. For $a, b \in \{0,1\}$, homomorphic XOR of $[\![a]\!], [\![b]\!]$ is denoted as $[\![a \oplus b]\!] = [\![a]\!] + [\![b]\!] - 2 \cdot [\![a]\!] \cdot [\![b]\!]$. For $a \in \{0,1\}$, we define its signed encoding as $b = 2a - 1 \in \{-1, 1\}$. $[\![b]\!]$ is homomorphically evaluated as $2 \cdot [\![a]\!] - [\![1]\!]$. Each of these operations can be naturally extended to vectors by applying them element-wise.

3.2 Somewhat Homomorphic Encryption

Somewhat Homomorphic Encryption (SWHE) is a form of encryption that allows limited-depth computation over encrypted data without access to the secret key [Gen09]. A SWHE scheme is described by the following randomized algorithms:

- **Key Generation:** $(\mathsf{params}, \mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^L)$. The key generation algorithm takes as input a security parameter λ and a maximum multiplication depth L. It outputs public parameters params, a secret key sk for decryption, a public key pk for encryption, and an evaluation key ek for homomorphic evaluation. Here, params is an implicit input to the following algorithms, omitted for simplicity.
- **Encryption:** $[\![m]\!] \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$. The encryption algorithm takes as input a public key $\mathsf{pk} \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext $[\![m]\!] \in \mathcal{C}$.
- **Decryption:** $m \leftarrow \mathsf{Dec}(\mathsf{sk}, [\![m]\!])$. The decryption algorithm takes as input a secret key sk and a ciphertext $[\![m]\!]$, and outputs $m \in \mathcal{M}$.

The SWHE scheme needs to satisfy the following properties.

1. **Message homomorphism.** For any $(\mathsf{params}, \mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^L)$, any messages $m_i \in \mathcal{M}$ with its ciphertexts $[\![m_i]\!] \leftarrow \mathsf{Enc}(\mathsf{pk}, m_i)$ for each $i \in [\ell]$, where $\ell \geq 1$, the following holds with probability $1 - \mathsf{negl}(\lambda)$: for any polynomial-size circuit f of depth at most L,

$$\mathsf{Dec}\big(\mathsf{sk}, \widetilde{f}([\![m_1]\!], \ldots, [\![m_\ell]\!])\big) = f(m_1, \ldots, m_\ell) \ .$$

   Here, $\widetilde{f}$ denotes the homomorphic evaluation of the circuit f on ciphertexts, which can be performed in time $\mathsf{poly}(\lambda)$.

2. **Ciphertexts pseudorandom.** For any secret key sk kept private, the distribution of ciphertexts is computationally indistinguishable from the uniform distribution over the ciphertext space.

3.3 Data Stream Algorithms

3.3.1 AMS Sketch

The AMS sketch is a logarithmic-size sketch for estimating the second frequency moment of a stream. It is an optimized and specialized instance of the estimator for the kth frequency moment, as proposed by [AMS96].

Theorem 1 (The AMS Sketch [AMS96])**.** *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}^N$, $\boldsymbol{r} \in \{-1, +1\}^N$ be a random four-wise independent vector, $Z_x = \sum_{i \in [N]} \boldsymbol{r}[i]\boldsymbol{x}[i]$, $Z_y = \sum_{i \in [N]} \boldsymbol{r}[i]\boldsymbol{y}[i]$. Then $\mathbb{E}[Z_x \cdot Z_y] = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$ and $\mathsf{Var}[Z_x \cdot Z_y] = O(\|\boldsymbol{x}\|_2^2 \|\boldsymbol{y}\|_2^2)$.*

To bound the estimation variance, the median-of-means technique is applied, which maintains $O\Big(\frac{1}{\mathcal{E}^2} \ln \frac{1}{\delta}\Big)$ independent estimators to achieve an $(\mathcal{E}, \delta)$-approximation.

Theorem 2 (Median of Mean [NY83]). *Let $\{X_{i,j}\}_{i\in[t],j\in[k]}$ be i.i.d. random variables such that $\mathbb{E}[X_{i,j}] = Q$. Define* MoM *operator as*

$$\mathsf{MoM}(\{X_{i,j}\}_{i\in[t],j\in[k]}) \stackrel{def}{=} \mathsf{Median}_{i\in[t]}\Big(\frac{1}{k}\sum_{j\in[k]} X_{ij}\Big).$$

Let $t = O\big(\log\frac{1}{\delta}\big)$ and $k = O\Big(\frac{\mathsf{Var}[X]}{\mathcal{E}^2(\mathbb{E}[X])^2}\Big)$. Then

$$\Pr\left[\big|\mathsf{MoM}\big(\{X_{i,j}\}_{i\in[t],j\in[k]}\big) - Q\big| \geq \mathcal{E}Q\right] \leq \delta\ .$$

3.4 Differential Privacy

Unlike homomorphic encryption, which protects the privacy of data during computation, *differential privacy* [DMNS06] protects the privacy of individuals in the released output of a computation. Informally, it requires that the output distribution of an analysis does not change significantly when any single individual's record is added to or removed from the dataset. Consequently, the presence or absence of any one record is statistically hidden, providing a rigorous information-theoretic privacy guarantee. Here, two datasets D and D' are called *neighbors*, denoted by $D \sim D'$, if they differ by one record. In our PJC setting, each record changes exactly one entry of the vector $\boldsymbol{a}$ or $\boldsymbol{b}$ by one. In other words, deleting a record decreases one $a[i]$ or one $b[j]$ by one, while adding a record increases one such entry by one.

Theorem 3 (ϵ-differential privacy). *A randomized algorithm $\mathcal{M}$ with range $\mathcal{R}$ is ϵ-differential privacy (DP) if, for any neighboring datasets $D \sim D'$, and for all measurable subsets $S \subseteq \mathcal{R}$,*

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

A commonly used DP mechanism is the *Laplace mechanism*. Given a query $Q : \mathcal{D} \to \mathcal{R}$, its *global sensitivity* (GS_Q) is defined as

$$\mathrm{GS}_Q \stackrel{\text{def}}{=} \max_{D,\, D' :\, D\sim D'} |Q(D) - Q(D')|.$$

Then, the Laplace mechanism $\mathcal{M}(D) = Q(D)+\eta$ satisfies ϵ-DP, where η is drawn from a Laplace distribution with scale GS_Q/ϵ, i.e., $\eta \sim \mathrm{Lap}\big(\mathrm{GS}_Q/\epsilon\big)$. DP enjoys some properties, one of the most commonly used ones is *composition theory*:

Lemma 1 (Composition theory). *Let $\{\mathcal{M}_1, \ldots, \mathcal{M}_k\}$ be a series of ϵ-DP mechanisms. Then*

$$\mathcal{M}_1(D), \mathcal{M}_2(D), \ldots, \mathcal{M}_k(D)$$

is $k\epsilon$-DP.

3.5 Functionalities

We present our protocol in the standard simulation-based paradigm by instantiating $\mathcal{F}_{\mathsf{PJC}}$ in the $(\mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{ThSWHE}}, \mathcal{F}_{\mathsf{2PC}})$-hybrid model. We define the required ideal functionalities below, deferring some to Appendix A.

3.5.1 Functionality for private join and compute over distributed input sets

This functionality $\mathcal{F}_{\mathsf{PJC}}$ (see Figure 2) defines the ideal process for private join and compute over distributed input sets. It involves two servers, Agg and CSP, and w parties $P_1, \ldots, P_w$. Each party submits a multiset to the functionality, which adds it to the designated multiset union. Finally, the functionality computes the frequency vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ of the two multiset unions, applies a randomized function F to obtain an approximation $z \leftarrow F(\boldsymbol{v}_1, \boldsymbol{v}_2)$, and sends z to Agg.

Functionality $\mathcal{F}_{\mathsf{ThSWHE}}$

Public parameters: Security parameter λ, maximum multiplication depth L, message share space $\mathcal{M}$. Let KeyGen and Dec denote the key generation and decryption algorithms of an FHE scheme, respectively.

1. **KeyGen:** Upon receiving (init) from both servers, the functionality invokes $(\mathsf{params}, \mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^L)$, stores sk, and sends $(\mathsf{pk}, \mathsf{ek})$ to both parties.
2. **Decryption:** Upon receiving $(\llbracket m \rrbracket)$ from both parties, the functionality fetches sk, decrypts $m = \mathsf{Dec}(\mathsf{sk}, \llbracket m \rrbracket)$, samples uniform $m_1, m_2 \leftarrow \mathcal{M}$ under the constraint that $m_1 + m_2 = m$, and sends m_1 to one party and m_2 to the other party.

Figure 3: Two-party threshold SWHE ideal functionality.

3.5.2 Functionality for threshold somewhat homomorphic encryption

The functionality $\mathcal{F}_{\mathsf{ThSWHE}}$ (see Figure 3) models an ideal *threshold SWHE* system between two parties. Compared to a standard SWHE scheme, the main difference lies in the decryption procedure. In the threshold setting, neither party can independently decrypt a ciphertext $\llbracket m \rrbracket$. Instead, both parties jointly decrypt $\llbracket m \rrbracket$ such that each party obtains one additive share of m. We realize this using a distributed decryption procedure (e.g., [LWYY24]).

4 AMS Sketch with Inadmissible Distribution

4.1 Δ-Inadmissible Distribution

Definition 1 (Inadmissible Set)**.** *Let $\mathcal{D}$ be a distribution over $\{-1,1\}^N$. A set of distinct index pairs $\mathcal{I} \subseteq [N]^2$ is called* inadmissible *if every $(i,j) \in \mathcal{I}$ satisfies $i \neq j$, and for all $(i,j),(k,\ell) \in \mathcal{I}$,*

$$\mathbb{E}_{\boldsymbol{r} \leftarrow \mathcal{D}}\big[\boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{r}[k]\boldsymbol{r}[\ell]\big] = 1 \ .$$

Definition 2 (Δ-Inadmissible Distribution)**.** *Let $\mathcal{D}$ be a distribution over $\{-1,1\}^N$. We say that $\mathcal{D}$ is* Δ-inadmissible *if it satisfies the following properties:*

1. **No pair-wise collision:** *For all $i \neq j$, $\mathbb{E}_{\boldsymbol{r} \leftarrow \mathcal{D}}\big[\boldsymbol{r}[i]\boldsymbol{r}[j]\big] = 0$.*
2. ***Bounded inadmissible set size:*** *Every inadmissible set has size at most Δ, i.e., for any inadmissible set $\mathcal{I}$ over distribution $\mathcal{D}$, we have $|\mathcal{I}| \leq \Delta$.*
3. ***Partition with zero cross-correlation:*** *There exists a partition $\{\mathcal{I}_1, \ldots, \mathcal{I}_t\}$ of $\{(i,j) \in [N]^2 : i \neq j\}$ such that each $\mathcal{I}_m$ is an inadmissible set, and distinct sets have zero cross-correlation, i.e., for any $a \neq b$, $(i,j) \in \mathcal{I}_a$, and $(k,\ell) \in \mathcal{I}_b$,*
$$\mathbb{E}_{\boldsymbol{r} \leftarrow \mathcal{D}}\big[\boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{r}[k]\boldsymbol{r}[\ell]\big] = 0 \ .$$

Fact 1 (Uniform Distribution is 2-Inadmissible)**.** *Let $\mathcal{D}$ be the uniform distribution over $\{-1,+1\}^N$. Then $\mathcal{D}$ is 2-inadmissible.*

Sketch of proof. For the uniform distribution:

1. Each inadmissible set contains at most two ordered pairs, namely (i,j) and (j,i) for some $i > j$. We denote such a set by $\mathcal{I}_{(i,j)}$. Hence, every inadmissible set has size at most two.
2. For any $i \neq j$, we have $\mathbb{E}_{\boldsymbol{r} \sim \mathcal{D}}\big[\boldsymbol{r}[i]\boldsymbol{r}[j]\big] = 0$, since $\boldsymbol{r}[i]$ and $\boldsymbol{r}[j]$ are independent and uniformly distributed over $\{-1,+1\}$.

3. For any distinct inadmissible set $\mathcal{I}_{(i,j)} \neq \mathcal{I}_{(k,\ell)}$ with $i > j$ and $k > \ell$, we have

$$\mathbb{E}_{\boldsymbol{r} \leftarrow \mathcal{D}}\big[\boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{r}[k]\boldsymbol{r}[\ell]\big] = 0$$

by independence. Thus, $\mathcal{D}$ satisfies the partition property with zero cross-correlation.

Hence, the uniform distribution satisfies all the conditions of a 2-inadmissible distribution. □

Theorem 4. *Given a binary matrix $\mathbf{A} \in \{0,1\}^{N \times n}$, we define distribution $\mathcal{D}_{\mathbf{A}}$ over $\{-1,+1\}^N$ as follows: 1) sample a binary vector $\boldsymbol{s} \in \{0,1\}^n$ uniformly at random and 2) output $\boldsymbol{r} = 2(\mathbf{A}\boldsymbol{s} \bmod 2) - \mathbf{1}$. Let $d > 1$ be a constant, $n \geq 4d$ and $N \leq n^{d/2}$. For an overwhelming fraction of possible choices of $\mathbf{A}$ with each row having Hamming weight d and $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$, we have distribution $\mathcal{D}_{\mathbf{A}}$ is $\log\log(n)$-inadmissible.*

We defer the full proof of Theorem 4 to Appendix C and provide a sketch below.

Proof sketch. We analyze the correlation structure induced by the distribution $\mathcal{D}_{\mathbf{A}}$. For any nonzero $\boldsymbol{v} \in \{0,1\}^n$, we define

$$\mathcal{S}_{\boldsymbol{v}} \stackrel{\text{def}}{=} \{(i,j) \in [N]^2 \mid \mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}\} .$$

Each $\mathcal{S}_{\boldsymbol{v}}$ forms an inadmissible set, since for any $(i,j) \in \mathcal{S}_{\boldsymbol{v}}$, the random variables $\boldsymbol{r}[i]$ and $\boldsymbol{r}[j]$ are correlated through the relation $\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}$. Thus, bounding $\Delta(\mathcal{D}_{\mathbf{A}})$ reduces to bounding $|\mathcal{S}_{\boldsymbol{v}}|$.

We first characterize the probability that two random weight-d vectors $\mathbf{A}_i, \mathbf{A}_j$ satisfy $\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}$. Let $t = \mathsf{HW}(\boldsymbol{v})$. A combinatorial counting argument shows that $p_{\boldsymbol{v}} = \Pr[\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}] = \frac{\binom{t}{t/2}\binom{n-t}{d-t/2}}{\binom{n}{d}^2 - \binom{n}{d}}$. We show that for constant d and $n \geq 4d$, this probability is maximized at $t = 2$, yielding $p_{\max} = O(n^{-(d+1)})$.

To bound $|\mathcal{S}_{\boldsymbol{v}}|$, define indicator variables $X_{(i,j)} = \mathbf{1}[\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}]$, so that $|\mathcal{S}_{\boldsymbol{v}}| = \sum_{i \neq j} X_{(i,j)}$. We analyze the tail probability $\Pr[|\mathcal{S}_{\boldsymbol{v}}| \geq k]$ via the factorial moment method, by bounding $\mathbb{E}[(|\mathcal{S}_{\boldsymbol{v}}|)_k]$.

We distinguish two cases depending on whether the indices appearing in the k pairs are all distinct. Note that for any pair (i,j) satisfying $\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}$, the reversed pair (j,i) also satisfies the same condition. Hence, each unordered pair $\{i,j\}$ corresponds to at most two ordered pairs. Without loss of generality, we may restrict attention to a set of $k/2$ unordered pairs, denoted by $\{i_1,j_1\},\ldots,\{i_{k/2},j_{k/2}\}$ with $i_t > j_t$ for all t. The *good case* corresponds to the situation where these $k/2$ pairs involve k distinct indices, i.e., no two pairs share a common index. The *bad case* corresponds to collisions among the indices. A key observation is that the bad case cannot occur: if two pairs share a vertex, the constraints imply that two distinct rows of $\mathbf{A}$ must be equal, contradicting the assumption that all rows are distinct. Hence, only the good case contributes.

In the good case, although rows are sampled without replacement, we can upper bound the joint probability via a sequential exposure argument, yielding a bound of $p_{\max}^{k/2}$. Since the number of such tuples is at most N^k, we obtain $\mathbb{E}[(|\mathcal{S}_{\boldsymbol{v}}|)_k] \leq N^k \cdot p_{\max}^{k/2}$. Using $N \leq n^{d/2}$, this implies $\mathbb{E}[(|\mathcal{S}_{\boldsymbol{v}}|)_k] \leq n^{-\Omega(k)}$.

Setting $k = \log\log n$ and applying Markov's inequality, we get

$$\Pr[|\mathcal{S}_{\boldsymbol{v}}| \geq \log\log n] \leq \Pr[(|\mathcal{S}_{\boldsymbol{v}}|)_k \geq k!] \leq \frac{\mathbb{E}[(|\mathcal{S}_{\boldsymbol{v}}|)_k]}{k!} \leq \mathsf{negl}(n) .$$

Finally, a union bound over all $\boldsymbol{v} \neq 0$ (using $\mathsf{HW}(\boldsymbol{v}) \leq 2d$) implies that $\Delta(\mathcal{D}_{\mathbf{A}}) \leq \log\log n$ with overwhelming probability. □

4.2 AMS Sketch

Given a Δ-inadmissible distribution $\mathcal{D}$ (see Theorem 4), we now show how such a distribution can be directly used to instantiate an AMS-type sketch for inner product estimation.

At a high level, the classical AMS analysis relies on 4-wise independence to ensure unbiasedness and small variance. In our setting, although 4-wise independence no longer holds, the Δ-inadmissibility property guarantees that correlations are confined within small inadmissible sets. This structure is sufficient to recover unbiasedness and bound the variance with a multiplicative loss that depends only on $\Delta(\mathcal{D})$.

Theorem 5 (AMS sketch)**.** *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}^N$, and let $\mathcal{D}$ be a Δ-inadmissible distribution over $\{-1,+1\}^N$. Define AMS estimator*

$$Z^2 \stackrel{def}{=} \Big(\sum_{i\in[N]} \boldsymbol{r}[i]\boldsymbol{x}[i]\Big)\cdot\Big(\sum_{j\in[N]} \boldsymbol{r}[j]\boldsymbol{y}[j]\Big) ,$$

where $\boldsymbol{r} \leftarrow \mathcal{D}$. Then, $\mathbb{E}\big[Z^2\big] = \langle \boldsymbol{x}, \boldsymbol{y}\rangle$, $\mathsf{Var}\big[Z^2\big] \le \Delta(\mathcal{D})\|\boldsymbol{x}\|_2^2\|\boldsymbol{y}\|_2^2$.

Proof. We first rewrite

$$\begin{aligned} Z^2 &\stackrel{\text{def}}{=} \Big(\sum_{i\in[N]} \boldsymbol{r}[i]\boldsymbol{x}[i]\Big)\cdot\Big(\sum_{j\in[N]} \boldsymbol{r}[j]\boldsymbol{y}[j]\Big) \\ &= \sum_{i\in[N]} \boldsymbol{x}[i]\boldsymbol{y}[i] + \sum_{i\neq j} \boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{x}[i]\boldsymbol{y}[j] , \end{aligned}$$

where the second equality follows from the fact that $\boldsymbol{r}[i]^2 = 1$ for all $i \in [N]$.

By definition of Δ-inadmissible distribution (see Definition 1 and Definition 2), we have $\mathbb{E}\big[\boldsymbol{r}[i]\boldsymbol{r}[j]\big] = 0$ for all $i \neq j$. Thus, we obtain

$$\mathbb{E}\big[Z^2\big] = \sum_{i\in[N]} \boldsymbol{x}[i]\boldsymbol{y}[i] + \sum_{i\neq j} \boldsymbol{x}[i]\boldsymbol{y}[j]\mathbb{E}\Big[\boldsymbol{r}[i]\boldsymbol{r}[j]\Big] = \langle \boldsymbol{x}, \boldsymbol{y}\rangle .$$

Thus, Z^2 is an unbiased estimation of $\langle \boldsymbol{x}, \boldsymbol{y}\rangle$.

Then, we bound the variance of the estimator. Since the first term of Z^2, namely $\sum_{i\in[N]} \boldsymbol{x}[i]\boldsymbol{y}[i]$, is a constant, we have

$$\begin{aligned} \mathsf{Var}\big[Z^2\big] =& \mathsf{Var}\Big[\sum_{i\neq j} \boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{x}[i]\boldsymbol{y}[j]\Big] = \mathbb{E}\Big[\Big(\sum_{i\neq j} \boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{x}[i]\boldsymbol{y}[j]\Big)^2\Big] \\ &= \sum_{i\neq j,k\neq l} \boldsymbol{x}[i]\boldsymbol{y}[j]\boldsymbol{x}[k]\boldsymbol{y}[l]\mathbb{E}\Big[\boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{r}[k]\boldsymbol{r}[l]\Big] , \end{aligned}$$

where $\mathbb{E}\Big[\sum_{i\neq j} \boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{x}[i]\boldsymbol{y}[j]\Big] = 0$. There exists a partition of $\{(i,j) \in [N]^2 : i \neq j\}$ into inadmissible sets $\mathcal{I}_1 \dot\cup \cdots \dot\cup \mathcal{I}_t$. Moreover, they have zero cross-correlation, i.e.,

$$\mathbb{E}\Big[\boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{r}[k]\boldsymbol{r}[l]\Big] = \begin{cases} 1, & (i,j),(k,l) \in \mathcal{I}_m \\ 0, & (i,j) \in \mathcal{I}_m, (k,l) \in \mathcal{I}_n, m \neq n \end{cases} .$$

Thus, in $\mathsf{Var}[Z^2]$ all cross-terms between different $\mathcal{I}_m$'s are vanished and we have

$$\begin{aligned} &\mathsf{Var}\big[Z^2\big] \\ \le& \sum_m \sum_{(i,j),(k,l)\in\mathcal{I}_m} \boldsymbol{x}[i]\boldsymbol{y}[j]\boldsymbol{x}[k]\boldsymbol{y}[l] = \sum_m \Big(\sum_{(i,j)\in\mathcal{I}_m} \boldsymbol{x}[i]\boldsymbol{y}[j]\Big)^2 . \end{aligned}$$

Applying Cauchy-Schwarz and Theorem 4 gives us

$$\begin{aligned} &\mathsf{Var}\big[Z^2\big] \\ \le& \sum_m |\mathcal{I}_m| \sum_{(i,j)\in\mathcal{I}_m} \boldsymbol{x}[i]^2\boldsymbol{y}[j]^2 \le \Delta(\mathcal{D}) \sum_m \sum_{(i,j)\in\mathcal{I}_m} \boldsymbol{x}[i]^2\boldsymbol{y}[j]^2 \\ \le& \Delta(\mathcal{D}) \sum_{i,j\in[N]} \boldsymbol{x}[i]^2\boldsymbol{y}[j]^2 \le \Delta(\mathcal{D})\|\boldsymbol{x}\|_2^2\|\boldsymbol{y}\|_2^2 . \end{aligned}$$

□

Intuitively, the AMS estimator only depends on the coordinates where $\boldsymbol{a}$ and $\boldsymbol{b}$ are nonzero. Let $\mathsf{supp} \stackrel{\text{def}}{=} \mathsf{supp}(\boldsymbol{a}) \cup \mathsf{supp}(\boldsymbol{b})$ denote this set of effective indices. Therefore, it suffices for the randomness vector $\boldsymbol{r}$ to satisfy the inadmissibility condition restricted to these effective indices. Motivated by this observation, we relax the condition $N \le n^{d/2}$ to $|\mathsf{supp}| \le n^{d/2}$. Indeed, the AMS estimator $Z^2 \stackrel{\text{def}}{=} \Big(\sum_{i\in[N]} \boldsymbol{r}[i]\boldsymbol{x}[i]\Big) \cdot \Big(\sum_{j\in[N]} \boldsymbol{r}[j]\boldsymbol{y}[j]\Big)$ only depends on coordinates in supp, and all terms $\boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{x}[i]\boldsymbol{y}[j]$, which are nonzero only when $i, j \in \mathsf{supp}$. Thus, the analysis only involves pairs $(i, j) \in \mathsf{supp} \times \mathsf{supp}$. It follows that it suffices for $\mathcal{D}$ to be Δ-inadmissible over supp, i.e., restricted to pairs of indices in supp. Consequently, as long as $|\mathsf{supp}| \le n^{d/2}$, the same inadmissibility guarantee applies, and the variance bound follows.

Having bounded the variance, we can design a final sketch from the above basic sketch by a median-of-means improvement. By Theorem 2, this will blow up the space usage by a factor of $\frac{\mathsf{Var}[Z^2]}{\mathbb{E}[Z^2]^2} O(\frac{1}{\mathcal{E}^2}\log\frac{1}{\delta}) = O(\Delta(\mathcal{D})\frac{1}{\mathcal{E}^2}\log\frac{1}{\delta})$ in order to give an $(\mathcal{E}, \delta)$-estimate.

4.3 Inner Product Approximation

The above analysis shows that the AMS estimator, when instantiated with a Δ-inadmissible distribution $\mathcal{D}$, provides an unbiased estimate of the inner product with controlled variance. We encapsulate this estimator as a primitive $\mathsf{InnProdApprox}_{(k,t)}$, which takes two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}^N$ and a distribution $\mathcal{D}$ over $\{-1, +1\}^N$, and outputs a value z approximating $\langle \boldsymbol{a}, \boldsymbol{b}\rangle$. This instantiation is given in Algorithm 1. The sketch randomness $\boldsymbol{r}$ is sampled from $\mathcal{D}$. The number of trials required to achieve an $(\mathcal{E}, \delta)$-estimation is $k = O\Big(\Delta(\mathcal{D}) \cdot \frac{1}{\mathcal{E}^2}\Big)$ and $t = O\big(\log\frac{1}{\delta}\big)$. Then, with probability at least $1 - \delta$, the output satisfies $\big|z - \langle \boldsymbol{a}, \boldsymbol{b}\rangle\big| \le \mathcal{E} \cdot \big(\|\boldsymbol{a}\|_2 \|\boldsymbol{b}\|_2\big)$.

Algorithm 1 Inner Product Approximation via AMS with Inadmissible Distribution: $\mathsf{InnProdApprox}_{(k,t)}(\mathcal{D}, \boldsymbol{a}, \boldsymbol{b})$

Input: $\boldsymbol{a} \in \mathbb{Z}^N, \boldsymbol{b} \in \mathbb{Z}^N$; an inadmissible distribution $\mathcal{D}$.
Output: Inner product approximation z.

for $\ell \in [k]$ and $h \in [t]$ **do**
  $\boldsymbol{r} \leftarrow \mathcal{D}$
  $u_a := \langle \boldsymbol{a}, \boldsymbol{r}\rangle$
  $u_b := \langle \boldsymbol{b}, \boldsymbol{r}\rangle$
  $z_{\ell,h} := u_a \cdot u_b$
end for
$z := \mathsf{MoM}(\{z_{\ell,h}\}_{\ell\in[k],h\in[t]})$
return z

5 PJC Protocol for Distributed Input Sets

We propose Algorithm 1 for private join and compute. The Δ-inadmissible distribution $\mathcal{D}$ is instantiated via FHE, enabling the encrypted generation of the sketching vector. The sketching procedure is then evaluated homomorphically, and the resulting sketches are combined using a 2PC protocol with a median-of-means aggregator. This yields an accurate approximation of the inner product while preserving input privacy.

Our framework naturally extends to PJC over distributed input sets. We encode the inputs as vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}^N$, where each coordinate aggregates the payloads associated with the corresponding key. We then estimate the inner product via $\langle \boldsymbol{a}, \boldsymbol{b}\rangle \approx \langle \boldsymbol{a}, \boldsymbol{r}\rangle \cdot \langle \boldsymbol{b}, \boldsymbol{r}\rangle$, where $\boldsymbol{r} \leftarrow \mathcal{D}$. This abstraction further generalizes seamlessly to the distributed setting. Given collections of parties $\mathcal{S}_1$ and $\mathcal{S}_2$, we aggregate their encoded inputs and compute

$$\Big\langle \sum_{i\in\mathcal{S}_1} \boldsymbol{a}^{(i)}, \sum_{j\in\mathcal{S}_2} \boldsymbol{b}^{(j)} \Big\rangle \approx \Big(\sum_{i\in\mathcal{S}_1} \langle \boldsymbol{a}^{(i)}, \boldsymbol{r}\rangle\Big) \cdot \Big(\sum_{j\in\mathcal{S}_2} \langle \boldsymbol{b}^{(j)}, \boldsymbol{r}\rangle\Big) .$$

Therefore, PJC over distributed inputs is obtained directly, without any modification to the underlying protocol.

Protocol Π_{PJC}

Participants: Two servers Agg, CSP and w Parties $P_1, \ldots, P_w$. **Parameters:** Let $k, t, n, N, d \in \mathbb{N}$. Let $H : \{0,1\}^* \to \{0,1\}^{N \times n}$ be a random oracle that, on any input, outputs a Boolean matrix such that each row has Hamming weight exactly d and all rows are pairwise distinct. **Private sets:** Each party P_i holds a private input set $\mathcal{X}_i = \{(x_j^i, v_j^i)\}_{j \in [m]}$, where $x_j^i \in [N]$ and $v_j^i \in \mathbb{Z}$.

Initialization:

1. Two servers invoke $\mathcal{F}_{\mathsf{Rand}}$ and $\mathcal{F}_{\mathsf{ThSWHE}}$ to obtain $seed \in \{0,1\}^\lambda$ and keys $(\mathsf{ek}, \mathsf{pk})$, and then publish them.
2. For each $\ell \in [tk]$, Agg samples $\boldsymbol{s}_1^\ell \in \{0,1\}^n$, encrypts it as $[\![\boldsymbol{s}_1^\ell]\!]$, and sends it to CSP. Agg also initializes two encrypted sketches $[\![c_1^\ell]\!]$ and $[\![c_2^\ell]\!]$ as fresh encryptions of zero.
3. CSP samples $\boldsymbol{s}_2^\ell \in \{0,1\}^n$, homomorphically computes $[\![\boldsymbol{s}^\ell]\!] = [\![\boldsymbol{s}_1^\ell]\!] \oplus \boldsymbol{s}_2^\ell$, then rerandomizes $[\![\boldsymbol{s}^\ell]\!]$ by homomorphically adding an encryption of zero, and finally publishes the rerandomized ciphertext $[\![\boldsymbol{s}^\ell]\!]$.

Union operation over the set of P on $(\mathsf{input}, p, \mathcal{S})$:

1. P computes $\mathbf{A} \leftarrow H(seed)$.
2. For $\ell \in [tk]$, given P's private set $\mathcal{X} = \{(a_i, b_i)\}_{i \in [m]}$, P homomorphically computes its encrypted sketching result: $[\![z^\ell]\!] = \sum_{i \in [m]} b_i [\![\boldsymbol{r}^\ell[a_i]]\!]$, where $[\![\boldsymbol{r}^\ell[i]]\!]$ is obtained via homomorphic evaluation as $[\![\boldsymbol{r}^\ell[i]]\!] = 2(\mathbf{A}_i [\![\boldsymbol{s}^\ell]\!] \bmod 2) - 1$. Then, P rerandomizes each ciphertext $[\![z^\ell]\!]$ by homomorphically using a fresh encryption of zero $[\![0]\!]$, and sends the resulting ciphertexts to Agg.
3. For $\ell \in [tk]$, Agg updates $[\![c_p^\ell]\!] \leftarrow [\![c_p^\ell]\!] + [\![z^\ell + 0]\!]$, where $p \in \{1, 2\}$.

Join Computation via 2PC on Servers:

1. For each $\ell \in [tk]$, Agg and CSP interact with $\mathcal{F}_{\mathrm{ThSWHE}}$ to jointly decrypt $[\![c_1^\ell]\!]$ and $[\![c_2^\ell]\!]$, such that Agg obtains (d_1^ℓ, d_2^ℓ) and CSP obtains $(c_1^\ell - d_1^\ell, c_2^\ell - d_2^\ell)$.
2. Two servers call $\mathcal{F}_{\mathrm{2PC}}$ on inputs (d_1^ℓ, d_2^ℓ) and $(c_1^\ell - d_1^\ell, c_2^\ell - d_2^\ell)$ to compute $z = \mathsf{MoM}\big(\{c_1^\ell \cdot c_2^\ell\}^{\ell \in [tk]}\big)$ and send z to Agg.

Figure 4: Private join and compute over distributed input sets, secure against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{ThSWHE}}, \mathcal{F}_{\mathsf{2PC}})$-hybrid model.

We present Π_{PJC} (Figure 4) to realize $\mathcal{F}_{\mathsf{PJC}}$ for private join and compute over distributed input sets. The protocol involves two servers, Agg and CSP, and proceeds in the FHE + 2PC hybrid model described above. The sketching vector $\boldsymbol{r}$ remains hidden throughout, and all computations are performed over encrypted sketches, with only the final approximate result revealed. The detailed protocol is given below.

Initialization: Servers Agg and CSP initialize $\mathcal{F}_{\mathrm{ThSWHE}}$ to obtain $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek})$, and invoke $\mathcal{F}_{\mathrm{Rand}}$ to sample a public seed $seed \in \{0,1\}^\lambda$. The public parameters $(\mathsf{pk}, \mathsf{ek}, seed)$ are published.

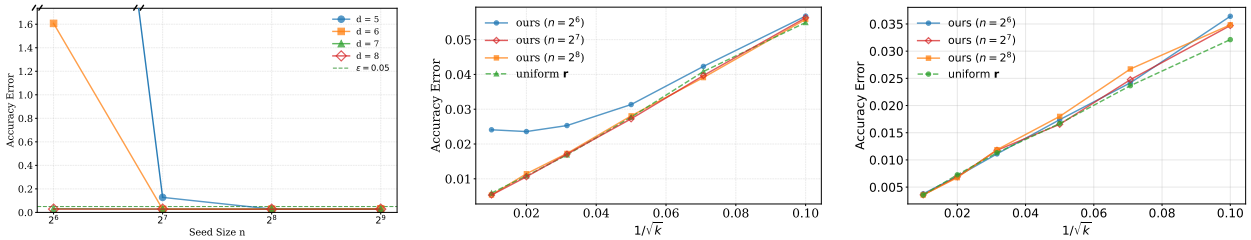
Both servers jointly generate an encrypted seed $\{[\![\boldsymbol{s}^\ell]\!]\}^{\ell \in [tk]}$ as follows. Agg samples a random vector $\boldsymbol{s}_1^\ell$ and sends its encryption $[\![\boldsymbol{s}_1^\ell]\!]$ to CSP. CSP samples $\boldsymbol{s}_2^\ell$, homomorphically computes $[\![\boldsymbol{s}^\ell]\!] = [\![\boldsymbol{s}_1^\ell]\!] \oplus \boldsymbol{s}_2^\ell$, and rerandomizes the result by adding a fresh encryption of zero. The resulting $[\![\boldsymbol{s}^\ell]\!]$ is then published.

Server Agg initializes $[\![c_1^\ell]\!]$ and $[\![c_2^\ell]\!]$ as fresh encryptions of zero for all $\ell \in [tk]$.

Union operation over the set of P on $(\mathsf{input}, p, \mathcal{S})$: Let P hold a private set $\mathcal{X} = \{(a_j, b_j)\}_{j \in [m]}$, and derive $\mathbf{A} \leftarrow H(seed)$. For each $\ell \in [tk]$, P homomorphically computes an encrypted sketch $[\![z^\ell]\!] = \sum_{j \in [m]} b_j \cdot [\![\boldsymbol{r}^\ell[a_j]]\!]$, where $[\![\boldsymbol{r}^\ell[i]]\!] = 2(\mathbf{A}_i [\![\boldsymbol{s}^\ell]\!] \bmod 2) - 1$. Each $[\![z^\ell]\!]$ is rerandomized and sent to Agg, who updates $[\![c_p^\ell]\!] \leftarrow [\![c_p^\ell]\!] + [\![z^\ell]\!]$.

Join Computation via 2PC on Servers: The servers jointly decrypt $\{[\![c_1^\ell]\!], [\![c_2^\ell]\!]\}^{\ell \in [tk]}$ via $\mathcal{F}_{\mathrm{ThSWHE}}$ to obtain secret shares, and invoke $\mathcal{F}_{\mathsf{2PC}}$ to compute $\mathsf{MoM}\big(\{c_1^\ell \cdot c_2^\ell\}^{\ell \in [tk]}\big)$.

In Theorem 6, we show that Π_{PJC} (Figure 4) securely realizes $\mathcal{F}_{\mathsf{PJC}}$ (Figure 2) against semi-honest adversaries.



(a) Accuracy error across (n, d) combinations

(b) Ours vs. standard AMS on uniform data

(c) Ours vs. standard AMS on real-world data

Figure 5: Accuracy Benchmark of Algorithm 1. (a) Empirical error across (n, d) combinations with $k = 400$, $t = 11$, $N = 2^{24}$, and $\mathsf{supp}(\boldsymbol{a}) \cup \mathsf{supp}(\boldsymbol{b}) = 1.5 \cdot 2^{18}$. (b,c) Empirical errors of Algorithm 1 against the standard AMS sketch on uniform and real-world data, with $d = 7$, $n \in \{2^6, 2^7, 2^8\}$; $\mathsf{supp}(\boldsymbol{a}) \cup \mathsf{supp}(\boldsymbol{b}) = 1.5 \cdot 2^{18}$ for uniform data and $\mathsf{supp}(\boldsymbol{a}) \cup \mathsf{supp}(\boldsymbol{b}) = 102{,}660$ for real-world data.

Theorem 6. *Let H be a random oracle, and assume that the SWHE scheme satisfies ciphertext pseudorandomness. Then, Protocol Π_{PJC} (Figure 4) securely realizes the functionality for private join and compute over distributed input sets in the presence of semi-honest adversaries in the $(\mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{ThSWHE}}, \mathcal{F}_{\mathsf{2PC}})$-hybrid model.*

The proof of Theorem 6 is deferred to Appendix D. Since fresh $\boldsymbol{s}^\ell$ are sampled per invocation while the public matrix $\mathbf{A}$ is derived from a fixed seed via a random oracle, the security analysis composes naturally across multiple protocol invocations on disjoint inputs.

6 Obtaining Differential Privacy for Free

So far, we have presented our PJC protocol based on homomorphic encryption, which achieves an error of $\mathcal{E} \cdot \big(\|\boldsymbol{a}\|_2^2 \|\boldsymbol{b}\|_2^2\big)$. As noted in Section 2.1, the functionality outputs a randomized approximation of the inner product, not the exact value. To further protect this released output, in this section we augment the protocol with differential privacy, so that the final estimate provably hides the contribution of any single record. We exploit the structure of our protocol to combine the two seamlessly, at minimal utility and efficiency loss. Notably, this adaptation is essentially "free of charge" when $\mathcal{E}$ is not too small, in the sense that achieving ε-DP incurs only an additional error term

$$O\left(\frac{\log \frac{1}{\delta}\sqrt{\Delta(\mathcal{D})}}{\varepsilon \mathcal{E}}\big(\|\boldsymbol{a}\|_2 + \|\boldsymbol{b}\|_2\big) + \frac{\log^2 \frac{1}{\delta}\,\Delta(\mathcal{D})^{1.5}}{\varepsilon^2 \mathcal{E}^3}\right).$$

Moreover, this additional term is asymptotically no larger than the original error of Algorithm 1 whenever $\mathcal{E} = \Omega\big(1/\min(\|\boldsymbol{a}\|_2, \|\boldsymbol{b}\|_2)\big)$ considering that ε is typically treated as a constant.

We now describe how to modify Algorithm 1 so that the output z satisfies ε-DP. The main idea is to privatize each $z_{\ell,h}$. Rather than adding noise directly to $z_{\ell,h}$, we apply the Laplace mechanism (see Section 3.4) separately to u_a and u_b. We work in the record-level DP model, where neighboring datasets differ in the addition or removal of a single (key, value) record. Since each such record affects at most one entry of $\boldsymbol{a}$ or $\boldsymbol{b}$ by one, the global sensitivity of both u_a and u_b is 1. Therefore, adding Laplace noise with scale $1/\varepsilon$ to each of u_a and u_b ensures that each $z_{\ell,h}$ satisfies ε-DP. Since there are $k \times t$ number of $z_{\ell,h}$, we then apply the basic composition theorem and allocate a privacy budget of $\varepsilon/(kt)$ to each one, so that the entire collection $\{z_{\ell,h}\}_{\ell \in [k],\, h \in [t]}$ together satisfies ε-DP. Finally, because z is obtained as a post-processing of $\{z_{\ell,h}\}_{\ell \in [k],\, h \in [t]}$, it also satisfies ε-DP. In summary, the modified algorithm adds Laplace noise with scale kt/ε to each of u_a and u_b. Importantly, these additional operations can be implemented without increasing the size of MPC, by each server adding a Laplace noise to their shares before feeding them to MPC. Below, we further show that this modification will only introduce a minor extra additive error.

Theorem 7. *For any $\varepsilon > 0$ and any $0 < \mathcal{E} < 1$ let $t = O\big(\log \frac{1}{\delta}\big)$ and $k = O\Big(\Delta(\mathcal{D}) \cdot \frac{1}{\mathcal{E}^2}\Big)$, then, with probability at least $1 - \delta$, DP version Algorithm 1 returns an $\tilde{z}$ such that*

$$\begin{aligned}&\big|\tilde{z} - \langle \boldsymbol{a}, \boldsymbol{b}\rangle\big| \\ \leq &\mathcal{E}\|\boldsymbol{a}\|_2\|\boldsymbol{b}\|_2 + O\Big(\frac{\log\frac{1}{\delta}\sqrt{\Delta(\mathcal{D})}}{\varepsilon\mathcal{E}}\big(\|\boldsymbol{a}\|_2 + \|\boldsymbol{b}\|_2\big) + \frac{\log^2\frac{1}{\delta}\Delta(\mathcal{D})^{1.5}}{\varepsilon^2\mathcal{E}^3}\Big).\end{aligned}$$

Proof. By following the same analysis as in Theorem 5, we have $\mathbb{E}(\tilde{z}_{\ell,h}) = \langle \boldsymbol{a}, \boldsymbol{b}\rangle$ and

$$\mathsf{Var}(\tilde{z}_{\ell,h}) \leq \Delta(\mathcal{D})\,\|\boldsymbol{a}\|_2^2\,\|\boldsymbol{b}\|_2^2 + 2\Big(\frac{tk}{\varepsilon}\Big)^2\big(\|\boldsymbol{a}\|_2^2 + \|\boldsymbol{b}\|_2^2\big) + 4\Big(\frac{tk}{\varepsilon}\Big)^4.$$

Therefore, with probability at least $1 - \delta$, we have

$$\mathcal{E}\|\boldsymbol{a}\|_2\|\boldsymbol{b}\|_2 + \frac{2t\sqrt{k}}{\varepsilon}\big(\|\boldsymbol{a}\|_2 + \|\boldsymbol{b}\|_2\big) + \frac{4t^2k^{1.5}}{\varepsilon^2}.$$

Combining with $k = O\big(\Delta(\mathcal{D}) \cdot \frac{1}{\mathcal{E}^2}\big)$ and $t = O\big(\log\frac{1}{\delta}\big)$, we get our target claim. □

When $\mathcal{E}$ is at least on the order of $\Omega\big(1/\min(\|\boldsymbol{a}\|_2, \|\boldsymbol{b}\|_2)\big)$, the first term, also corresponding to the error of Algorithm 1, dominates the remaining two terms. Thus, differential privacy comes essentially for free. For simplicity we use basic composition; tighter accountants such as Rényi DP [Mir17] would shrink the per-mechanism noise scale by roughly a $\sqrt{kt}$ factor and improve the ε-regime over which DP comes essentially for free, but do not change the asymptotic form of Theorem 7.

7 Performance Evaluation

We implement our protocol end-to-end to provide an accurate benchmark of its efficiency[1]. Homomorphic encryption operations are implemented using the Microsoft SEAL library [SEA23], while two-party computation (2PC) operations are realized with the EMP toolkit [WMK16]. In SEAL, we set the plaintext modulus to 24 bits and the polynomial modulus degree to 8192, supporting final output up to 2^{13}. Larger ranges require higher SEAL parameters. For efficiency, we use packing to batch up to 8192 instances into a single ciphertext. All experiments are conducted on AWS `c6i.xlarge` instances using a single thread. We consider two network configurations: a LAN setting with 5 Gbps bandwidth, and a WAN setting with 250 Mbps bandwidth and 80 ms latency.

7.1 Benchmark Accuracy

We evaluate the accuracy of $\mathsf{InnProdApprox}_{(k,t)}(\mathcal{D}, \boldsymbol{a}, \boldsymbol{b})$ in Algorithm 1 under different parameter configurations and data distributions. We use binary vectors $\boldsymbol{a}, \boldsymbol{b} \in \{0,1\}^N$ with $N = 2^{24}$, $\mathsf{HW}(\boldsymbol{a}) = \mathsf{HW}(\boldsymbol{b}) \leq 2^{18}$, and $\langle \boldsymbol{a}, \boldsymbol{b}\rangle \leq 2^{17}$. The inadmissible distribution $\mathcal{D}$ is instantiated following Theorem 4 via a binary matrix $\mathbf{A} \in \{0,1\}^{N\times n}$ with row weight d. We examine how the empirical accuracy tracks our theoretical bound as k, t, n, and d vary. Each reported value is the median over 1000 independent trials.

Accuracy with respect to (n, d). We study how the empirical accuracy of Algorithm 1 depends on the parameters (n, d) that define $\mathcal{D}$. Recall that n controls the inadmissibility level $\Delta(\mathcal{D}) = \log\log n$, while d governs the feasibility condition $\mathsf{supp}(\boldsymbol{a}) \cup \mathsf{supp}(\boldsymbol{b}) \leq n^{d/2}$. We fix $k = 400, t = 11$, which yields a theoretical bound of $\mathcal{E} \approx 0.05$, and sweep $d \in \{5, 6, 7, 8\}$ and $n \in \{2^6, 2^7, 2^8, 2^9\}$, reporting the median accuracy error over 1000 trials. As shown in Figure 5a, the empirical median error falls below the theoretical bound $\mathcal{E} \approx 0.05$ precisely when $n^{d/2} \geq \mathsf{supp}(\boldsymbol{a}) \cup \mathsf{supp}(\boldsymbol{b})$, and degrades sharply once this bound is violated. This empirical result confirms that (n, d) need only be large enough to cover the effective inputs $\mathsf{supp}(\boldsymbol{a}) \cup \mathsf{supp}(\boldsymbol{b})$ rather than the whole dimension N, which is a great saving for sparse vectors.

Accuracy on uniform data. Recall that k controls the approximation error $\mathcal{E}$, while t controls the confidence δ. We fix $t = 11$ and vary $k \in \{100, 200, 400, 1000, 2500, 10000\}$ to evaluate the accuracy of

[1] https://github.com/YunqingSun0-0/DPJC.git

		$n = 128$	$n = 256$	$n = 512$	$n = 1024$
Comp (s)	LAN	1.83	3.28	6.22	12.12
	WAN	6.96	12.00	22.95	43.68
Comm (MB)		52.38	104.76	209.53	419.06
Publish (MB)		54.28	106.66	211.43	420.96

Table 2: Initialization costs for different n.

Algorithm 1, and compare with the standard AMS sketch as a baseline. We generate $\boldsymbol{a}$ and $\boldsymbol{b}$ uniformly at random with $\mathsf{HW}(\boldsymbol{a}) = \mathsf{HW}(\boldsymbol{b}) = 2^{18}$ and $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = 2^{17}$, so $\mathsf{supp}(\boldsymbol{a}) \cup \mathsf{supp}(\boldsymbol{b}) = 1.5 \cdot 2^{18}$. We instantiate three distributions $\mathcal{D}$ with $d = 7$ and $n \in \{2^6, 2^7, 2^8\}$. The standard AMS sketch samples $\boldsymbol{r} \in \{-1, +1\}^N$ uniformly and admits the theoretical error bound $1/\sqrt{k}$.

As shown in Figure 5b, the empirical accuracy of Algorithm 1 matches the standard AMS sketch for $d = 7$ and $n \in \{2^7, 2^8\}$ across all k. This indicates that although our theoretical error bound is looser, we achieve the same practical accuracy on uniform data. One caveat is that $\mathcal{D}$ is generated from a short seed of length n, so n must provide enough entropy to support inputs with large $\mathsf{supp}(\boldsymbol{a}) \cup \mathsf{supp}(\boldsymbol{b})$. This is reflected in the $(2^6, 7)$ configuration, where the error fails to decrease as k increases.

Accuracy on real-world data. To evaluate Algorithm 1 on realistic workloads, we repeat the above experiment using vectors derived from the UCI Bag-of-Words dataset [New08]. Unlike the uniform case where $\boldsymbol{a}, \boldsymbol{b}$ are binary, here $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}^N$ are count vectors. Specifically, we use a document with 10^8 total word occurrences over 102,660 unique words, and on each side we randomly sample 2^{18} words to form $\boldsymbol{a}$ and $\boldsymbol{b}$. We use the parameter sweep $t = 11$, $d = 7$, $k \in \{100, 200, 400, 1000, 2500, 10000\}$, and $n \in \{2^6, 2^7, 2^8\}$, comparing against the standard AMS sketch.

As shown in Figure 5c, the empirical accuracy of Algorithm 1 closely tracks that of the standard AMS sketch across all k and n, and all empirical errors lie below the theoretical bound $1/\sqrt{k}$ of the standard AMS sketch. The entropy limitation observed on uniform data at $n = 2^6$ does not appear here, because real-world document vectors have much smaller $\mathsf{supp}(\boldsymbol{a}) \cup \mathsf{supp}(\boldsymbol{b})$ (at most 102,660). Thus, Algorithm 1 remains accurate on realistic inputs.

7.2 Protocol Efficiency

Building on the accuracy results above, we now fix $t = 11$ and $k = 400$, which yields an error less than 0.05, and evaluate end-to-end protocol efficiency. The private join computation among distributed clients proceeds in three phases:

1. Server initialization: Before clients provide their private inputs, CSP and Agg jointly generate and publish the encrypted random vectors, a public *seed*, and the key material $(\mathsf{ek}, \mathsf{pk})$. This phase involves only server-to-server computation and a one-time broadcast to clients; no client input is required.

2. Client local sketching: Each client locally derives its encrypted sketch by expanding the encrypted randomness into the inadmissible distribution on its private elements and evaluating the sketch. The client sends a single sketch ciphertext to Agg.

3. Server aggregation and output: Agg homomorphically aggregates all client encrypted sketches into two ciphertexts, runs distributed decryption with CSP, executes 2PC to reconstruct the sketches, and computes the median-of-means estimate.

We evaluate per-phase cost under LAN and WAN settings below.

7.2.1 Server Initialization

In this phase, the communication overhead comes from CSP sending its encrypted random vectors to Agg, and the computational cost comes from Agg homomorphically combining them with its own random vectors. We assume the homomorphic-encryption keys are already generated, so this CSP-Agg communication consists solely of the encrypted random vectors. We vary the length $n \in \{128, 256, 512, 1024\}$ of the random vectors

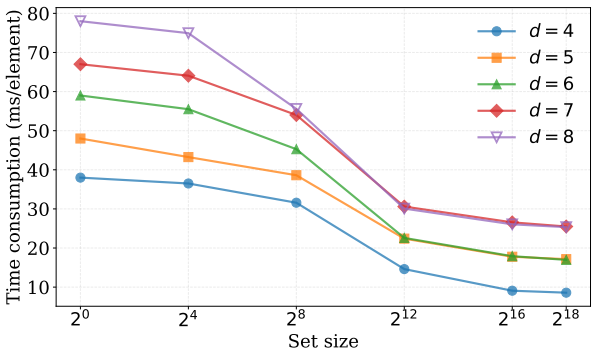


Figure 6: Amortized per-element cost for client local sketching. Tested under varying $d = \mathsf{HW}(\mathbf{A}_i)$ and set size m.

Setting	$w = 2{,}000$ clients (1000/side)			$w = 20$ clients (10/side)		
	$m = 2^8$	2^{12}	2^{16}	2^{16}	2^{20}	2^{24}
Ours						
Client comm	**0.25 M**	**0.25 M**	**0.25 M**	**0.25 M**	**0.25 M**	**0.25 M**
Client time	**14.2 s**	**2.05 m**	**28.3 m**	**28.3 m**	**7.36 h**	**4.92 d***
Server comm	**58.52 M**	**58.52 M**	**110.90 M**	**58.52 M**	**110.90 M**	**215.67 M**
Server time	**2.04 s**	**2.04 s**	**3.49 s**	**1.87 s**	**3.32 s**	**6.26 s**
KLS24 [KLS24]						
Client comm	910 M	2.15 G	20.2 G	202 M	3.07 G	51 G
Client time	3.55 m	5.20 m	1.25 h	44.9 s	13.5 m	4.81 h
Server				≈ 0		
CHIKT23 [CHI$^+$23]						
Client comm	67 M*	1.10 G	17.2 G	172 M	2.75 G	44 G*
Client time	1.67 m*	27.0 m	7.19 h	4.32 m	1.18 h	18.8 h*
Server				≈ 0		
PSTY19 [PSTY19]						
Client comm	1.84 G*	29.5 G*	472 G	4.72 G	75 G*	1.33 T*
Client time	8.8 s*	5.08 m*	2.37 h	1.42 m	8.92 m*	1.93 h*
Server				≈ 0		
Server-based 2PC						
Client				≈ 0		
Server comm	2.19 T	561 T	140 P	14.37 T	3.6 P	920 P
Server time	5.70 h	60.8 d	42.6 y	1.56 d	1.1 y	279 y

Table 3: PJC over distributed inputs. We report per-client and total server cost under a large-w setting and a small-w setting; each client holds m elements. Communication is measured in bytes, where M, G, T, and P denote 10^6, 10^9, 10^{12}, and 10^{15}, respectively. For the PJC-decomposition baselines [KLS24, CHI$^+$23, PSTY19], each client executes one pairwise PJC against every client on the opposite side; the server only sums the received pairwise inner products, with negligible cost (≈ 0). For the server-based generic 2PC baseline, client-side cost consists only of preparing and sending secret shares and is likewise negligible (≈ 0). Time values are in seconds (s), minutes (m), hours (h), days (d), or years (y) as appropriate. Values marked with * are extrapolated.

and evaluate both costs. As shown in Table 2, both grow linearly with n. Concretely, for $n = 1024$, initialization takes 12.12 s in LAN and 43.68 s in WAN, with 419.06 MB of communication between the two servers. Agg then publishes the encrypted randomness, a public seed, and key material, totaling 420.96 MB.

		Aggregate Client Encrypted Sketch	Threshold Decryption	2PC	Total
Comp. (s)	LAN	0.17	0.001	0.04	0.211
	WAN	0.17	0.006	1.50	1.676
Comm. (MB)		0	1.01	3.23	4.24

Table 4: Server aggregation and output costs. Broken down by step. The encrypted sketch aggregation includes 2000 clients.

7.2.2 Client Local Sketching

In this phase, all computations are performed locally by each client. The dominant cost is expanding the encrypted randomness at each private element. For each element, this expansion multiplies d ciphertexts together, where $d = \mathsf{HW}(\mathbf{A_i})$. We implement this as a balanced binary tree of homomorphic multiplications, reducing the multiplication depth to $\lceil \log d \rceil$. Recall the kt rounds are packed into a single ciphertext; total cost grows linearly with number of private elements.

We vary $d \in \{4, 5, 6, 7, 8\}$ and set size $m \in \{2^0, 2^4, 2^8, 2^{12}, 2^{16}, 2^{18}\}$; reporting the amortized per element cost in Figure 6. Two trends emerge. First, the amortized per element cost decreases as m grows. This is due to a caching optimization at the first tree level: pairwise products of encrypted-randomness indices are stored in a hash table and reused across elements, so larger m enables more cache hits. Second, the curves collapse into three tiers at large m. Once the cache absorbs nearly all first-level multiplications, the amortized cost is determined by the remaining uncached upper-tree multiplications: one for $d = 4$, two for $d = \{5, 6\}$, and three for $d = \{7, 8\}$. Values of d sharing the same uncached count therefore incur essentially the same cost. Concretely, for $d = 8$, the amortized cost is 30.08 ms at $m = 2^{12}$ and 25.34 ms at 2^{18}.

On the communication side, each client sends only its sketch ciphertext to Agg, incurring 0.25 MB per client regardless of m.

7.2.3 Server Aggregation and Output

In this phase, three steps dominate the cost: (1) aggregating encrypted sketches from multiple clients; (2) threshold decryption to obtain secret shares of the two sketches; (3) a 2PC protocol to reconstruct the sketches and compute the median-of-means estimate. Aggregation cost scales linearly with the number of clients (one HE addition per client); we report the time for 2000 clients. We estimate the distributed decryption cost following [ZLY+26], which takes 0.001 s in LAN and 0.006 s in WAN, with 1.01 MB communication. The 2PC cost depends on kt because, unlike earlier phases that pack all kt instances into a single ciphertext via SIMD, the 2PC operates on plaintext and handles each instance separately. For $kt = 4400$ in our setting, the 2PC takes 0.04 s in LAN and 1.50 s in WAN, with 3.23 MB communication.

7.2.4 Efficiency under Large Polynomial Modulus Degree.

Our default SEAL parameters support PJC with unit payloads, and random payloads when outputs remain below 2^{13} due to the limited noise budget. To support larger payloads, we test under polynomial modulus degree 16384, supporting output up to 2^{58} and batches up to $k \cdot t = 16384$ rounds for better accuracy. In our experiments, we use plaintext modulus 2^{24}. The test results are shown in Appendix E. Overall, the efficiency impact is limited: server initialization increases by 3×, client computation increases by 3× but remains fully parallelizable, and client–server communication grows from 0.25 MB to 0.71 MB. Server aggregation and output increases by 4× due to overflow protection for large payloads.

7.3 Comparison with Distributed-PJC Baselines

As discussed in Section 1, we compare our scheme against two natural baselines for extending PJC to the distributed setting: PJC decomposition from existing two-party PJC protocols [PSTY19, CHI+23, KLS24]; and server-based generic 2PC. We evaluate both under two settings: a large-w setting (small m) with w = 2,000 clients (e.g., small businesses with limited per-party data); a small-w setting (large m) with

$w = 20$ clients (e.g., enterprises each holding sizable datasets). In both, clients are evenly split between the two sides, and each holds m elements. Results are reported in Table 3.

- **PJC-decomposition baselines.** We first measure the per-execution cost of each two-party PJC scheme [PSTY19, CHI+23, KLS24]. As no public implementation is available for [CHI+23] or for the inner-product construction of [PSTY19], we adopt the measurements reported in [CHI+23] and extrapolate communication and LAN running time to other input sizes using their asymptotic complexities. For [KLS24] we run its implementation on a machine with 128 GB of RAM. Every client sequentially executes pairwise PJC against every client on the opposite side, so per-client communication and computation scale by 1,000× in the large-w setting and 10× in the small-w setting with respect to a single 2-party PJC. The server's role is to sum the received pairwise inner products, which is negligible.
- **Server-based generic 2PC baseline.** Each client secret-shares its input to two non-colluding servers running a generic 2PC protocol. The protocol jointly sorts all entries by key (with source tags), and for each matching key aggregates payloads within each source, multiplies them across sources, and sums the results. The oblivious sort dominates, scaling as $O(N \log^2 N)$ in the aggregate input size N. Due to memory limitations, we partition each side's aggregated input into chunks of 2^{13} and perform pairwise PJC. A single such PJC takes 21 s in LAN and 60 s in WAN with 2.3 GB communication; scaling to a per-side aggregate of size M increases cost by roughly $(M/2^{13})^2$ due to cross-chunk computation. Entries in Table 3 are extrapolated accordingly.

As shown in Table 3, our scheme lowers total communication in both settings. Client communication is a constant 0.25 MB, regardless of m. In small-w setting ($m = 2^{24}, w = 20$), total communication is 221 MB, roughly 4,000× less than [CHI+23] and over $10^9\times$ less than server-based 2PC. In the large-w setting ($m = 2^{16}$, $w = 2{,}000$), total communication is 611 MB, four orders of magnitude below the PJC-decomposition and eight below server-based 2PC. Computation gains follow the same trend in the large-w setting: in $w = 2{,}000 (m = 2^{16})$, each client finishes in 28.3 minutes, whereas each client in KLS24 takes 1.25 hours and the server-based 2PC server requires 42.6 years. In the small-w setting, however, our per-client time is higher than the PJC-decomposition baselines [KLS24, CHI+23, PSTY19]. At $w = 20, m = 2^{24}$, our single-threaded client time is 4.92 days. However, our entire client cost is local computation and hence embarrassingly parallelizable.

Optimized Multi-threaded Implementation. To further reduce the client local computation time in the small-w, large-m setting, we implement an optimized multi-threaded client. We partition the private elements across workers. To preserve the first-level pair cache optimization of the single-threaded implementation across workers, we first collect all unique first-level index pairs over the full client input and compute the corresponding ciphertext products in parallel. These products are stored in a shared read-only cache. We additionally cache the second level of the multiplication tree. At large m, this table saturates and its one-time construction cost is well amortized across elements. For $d = 8$, the resulting reuse reduces the dominant uncached tree work from three ciphertext multiplications per element to one, corresponding to roughly a further 3× reduction in the amortized HE multiplication cost compared with the original single-threaded path. We enabled Intel HEXL and evaluated the implementation on an AWS `c6i.16xlarge` instance using 32 threads. For $m = 2^8, 2^{12}, 2^{16}$, and 2^{20}, the optimized client finishes computation in 0.58 s, 4.25 s, 44.89 s, and 298.79 s, respectively. Notably, with parallelization and the roughly 3× additional reduction in amortized HE multiplication cost from second-level caching, the client run time at $m = 2^{20}$ drops from 7.36 hours in the original single-threaded implementation to under five minutes. We expect GPU acceleration to improve performance further in the small-w, large-m setting.

Acknowledgements

Wei Dong is supported by the RIE2025 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) (Award I2301E0026), administered by A*STAR, as well as supported by Alibaba Group and NTU Singapore through Alibaba-NTU Global e-Sustainability CorpLab (ANGEL).

References

[ABBS25] Amit Agarwal, Carsten Baum, Lennart Braun, and Peter Scholl. Low-bandwidth mixed arithmetic in VOLE-based ZK from low-degree PRGs. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 396–426. Springer, Cham, May 2025.

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *28th ACM STOC*, pages 20–29. ACM Press, May 1996.

[BCL+10] Vladimir Braverman, Kai-Min Chung, Zhenming Liu, Michael Mitzenmacher, and Rafail Ostrovsky. AMS without 4-wise independence on product domains. In *27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 5 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 119–130. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.

[CHI+23] Koji Chida, Koki Hamada, Atsunori Ichikawa, Masanobu Kii, and Junichi Tomida. Communication-efficient inner product private join and compute with cardinality. In Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik, editors, *ASIACCS 23*, pages 678–688. ACM Press, July 2023.

[CILO22] Wutichai Chongchitmate, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. PSI from ring-OLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 531–545. ACM Press, November 2022.

[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. Springer, Berlin, Heidelberg, March 2006.

[FT25] Francesca Falzon and Tianxin Tang. Learning from functionality outputs: Private join and compute in the real world. In Lujo Bauer and Giancarlo Pellegrino, editors, *USENIX Security 2025*, pages 7603–7622. USENIX Association, August 2025.

[Gen09] Craig Gentry. A fully homomorphic encryption scheme. In *Ph.D. thesis, Stanford University*, 2009.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. In *Elect. Colloq. on Computational Complexity (ECCC)*, volume 7, 2000.

[Goo19] Google. Helping organizations do more without collecting more data, June 2019.

[HR16] Ishay Haviv and Oded Regev. The restricted isometry property of subsampled fourier matrices. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 288–297. SIAM, 2016.

[HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 598–628. Springer, Cham, December 2017.

[IKN+20] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *2020 IEEE European Symposium on Security and Privacy*, pages 370–389. IEEE Computer Society Press, September 2020.

[IM08] Piotr Indyk and Andrew McGregor. Declaring independence via the sketching of sketches. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 737–745. SIAM, 2008.

[KLS24] Jiseung Kim, Hyung Tae Lee, and Yongha Son. Private set operations from circuit-based PSI. Cryptology ePrint Archive, Paper 2024/1560, 2024.

[KW11] Felix Krahmer and Rachel A. Ward. New and improved johnson-lindenstrauss embeddings via the restricted isometry property. *SIAM J. Math. Anal.*, 43(3):1269–1281, 2011.

[LPR+21] Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. Private join and compute from PIR with default. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 605–634. Springer, Cham, December 2021.

[LWYY24] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. The hardness of LPN over any integer ring and field for PCG applications. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 149–179. Springer, Cham, May 2024.

[Mir17] Ilya Mironov. Rényi differential privacy. In Boris Köpf and Steve Chong, editors, *CSF 2017 Computer Security Foundations Symposium*, pages 263–275. IEEE Computer Society Press, 2017.

[MRR20] Payman Mohassel, Peter Rindal, and Mike Rosulek. Fast database joins and PSI for secret shared data. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1271–1287. ACM Press, November 2020.

[New08] David Newman. Bag of Words. UCI Machine Learning Repository, 2008. DOI: https://doi.org/10.24432/C5ZG6P.

[NIW+13] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 801–812. ACM Press, November 2013.

[NY83] A. S. Nemirovsky and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, 1983.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Cham, May 2019.

[SEA23] Microsoft SEAL (release 4.1). `https://github.com/Microsoft/SEAL`, January 2023. Microsoft Research, Redmond, WA.

[SJ23] Yongha Son and Jinhyuck Jeong. PSI with computation or circuit-PSI for unbalanced sets from homomorphic encryption. In Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik, editors, *ASIACCS 23*, pages 342–356. ACM Press, July 2023.

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

[WY21] Yilei Wang and Ke Yi. Secure yannakakis: Join-aggregate queries over private data. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 1969–1981, New York, NY, USA, 2021. Association for Computing Machinery.

[YWZ20] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1627–1646. ACM Press, November 2020.

[ZLY+26] Wenhao Zhang, Hanlin Liu, Kang Yang, Wen jie Lu, Yu Yu, Xiao Wang, and Chenkai Weng. BitGC made (more) efficient. In *Advances in Cryptology - EUROCRYPT 2026*, Lecture Notes in Computer Science. Springer, 2026.

Open Science

We provide an repository at: `https://github.com/YunqingSun0-0/DPJC.git`. The repository is intended to support reproducibility of our results and to facilitate further research. It includes the following materials:

- Source code of the proposed protocol and all related components.
- Experimental scripts for reported results, including performance evaluation and accuracy evaluation.
- A README file that describes the overall code structure, provides step-by-step instructions for running the experiments.
- Configuration files specifying the cryptographic parameters and system settings used in the evaluation. Scripts for data preprocessing and synthetic dataset generation used in the experiments.

A Other Functionalities

A.1 Functionality for coin-tossing.

Our protocol would invoke the standard coin-tossing functionality $\mathcal{F}_{\mathsf{Rand}}$ shown in Figure 7, which can be securely computed in the random oracle model (see, e.g., [HSS17, YWZ20]).

Functionality $\mathcal{F}_{\mathsf{Rand}}$

Upon receiving (Rand, S) from both parties, where S is an efficiently sampleable set, sample $r \leftarrow S$, and then output r to all parties.

Figure 7: The ideal functionality for coin-tossing.

A.2 Functionality for 2-party Computation

This functionality $\mathcal{F}_{\mathsf{2PC}}$ (see Figure 8) defines the ideal execution of two-party computation (2PC). Two parties, P_1 and P_2, provide inputs x and y, respectively, and a trusted functionality computes $z = f(x, y)$. In the honest setting, both parties receive the output z. If one party is corrupted, the adversary controlling it receives z and may either send it to the honest party or output $\perp$.

Functionality $\mathcal{F}_{\mathsf{2PC}}$

Participants: Two parties P_1 and P_2.
Public parameters: A function $f(\cdot, \cdot)$.
Input Phase: Upon receiving input x from P_1 and y from P_2, the functionality records these values. If a party does not provide input, it uses a default value $\perp$.
Computation Phase: Once both inputs are received, compute $z \leftarrow f(x, y)$.
Output Phase: Sends (output, z) to P_1; P_2 receives no output.

Figure 8: The ideal functionality for two-party computation.

B Our 4-Wise Independent Construction

Theorem 8. *(Our 4-Wise Independent Construction) Let $m = n^2$, and let $\boldsymbol{c} \leftarrow \mathbb{F}_2^n$, $\boldsymbol{d} \leftarrow \mathbb{F}_2^n$, and $\boldsymbol{e} \leftarrow \mathbb{F}_2^{2n}$ be independent uniformly random bit vectors. Define $\boldsymbol{f} \in \mathbb{F}_2^m$, such that for all $i, j \in [n], k = (i-1)n+j \in [\ell]$, $\boldsymbol{f}_k = \boldsymbol{c}_i \oplus \boldsymbol{d}_j \oplus \boldsymbol{e}_{i+j}$. Then, $\boldsymbol{f}$ is four-wise independent over $\mathbb{F}_2^m$.*

Proof. Let $k_\ell = (i_\ell - 1)n + j_\ell$ for $\ell = 1, 2, 3, 4$ be four queried positions in $\boldsymbol{f}$. From the definition of $\boldsymbol{f}$, we have: $\boldsymbol{f}_{k_\ell} = \boldsymbol{c}_{i_\ell} \oplus \boldsymbol{d}_{j_\ell} \oplus \boldsymbol{e}_{i_\ell + j_\ell}$, for $\ell = 1, 2, 3, 4$. Define $\mathbf{F}_{k_\ell} \in \mathbb{F}^{4n}$ as a row vector with element one at indexes i_ℓ, $j_\ell + n$, and $i_\ell + j_\ell + 2n$, respectively. All other entries are zero. We have

$$\boldsymbol{f}_{k_\ell} = \mathbf{F}_{k_\ell} \cdot \begin{bmatrix} \boldsymbol{c} & \boldsymbol{d} & \boldsymbol{e} \end{bmatrix}^T \tag{1}$$

To establish 4-wise independence, for any four distinct indices $k_1, k_2, k_3, k_4 \in [\ell]$, the joint distribution of the corresponding outputs is uniform over $\mathbb{F}_2^4$. We observe that, if $\mathrm{rank}(\mathbf{F}_{k_1,k_2,k_3,k_4}) = 4$, $\{\boldsymbol{f}_{k_\ell}\}_{\ell \in [4]}$ is uniformly distributed over $\mathbb{F}_2^4$. We prove that, in our construction, $\mathrm{rank}(\mathbf{F}_{k_1,k_2,k_3,k_4})$ is always equal to 4. We distinguish the cases as follows:

1. If there are four i_ℓs, $\ell \in \{1, 2, 3, 4\}$ that composes $\mathbf{F}_{k_\ell}$ respectively, no matter how j_ℓ is selected in each $\mathbf{F}_{k_\ell}$, $\mathrm{rank}(\mathbf{F}_{k_1,k_2,k_3,k_4}) = 4$.

2. If there are three i_ℓs, $\ell \in \{1, 2, 3\}$, that composes $\mathbf{F}_{k_\ell}$, $\ell \in \{1, 2, 3, 4\}$, we assume i_3 appears twice in $\mathbf{F}_{k_3}$ and $\mathbf{F}_{k_4}$. According to our construction, index $k_\ell = (i_\ell - 1)n + j_\ell$. Thus, given $i_3 = i_4$, $\mathbf{F}_{k_3}$ and $\mathbf{F}_{k_4}$ are constructed by two different j_3 and j_4. We observe that, no matter how j_1 and j_2 are selected in $\mathbf{F}_{k_1}$ and $\mathbf{F}_{k_2}$, $\mathrm{rank}(\mathbf{F}_{k_1,k_2,k_3,k_4}) = 4$.

3. If there are two i_ℓs, $\ell \in \{1, 2\}$, that determines $\mathbf{F}_{k_\ell}$, $\ell \in \{1, 2, 3, 4\}$, we assume two cases: 1) i_1 appears once in $\mathbf{F}_{k_1}$ and i_2 appears three times in $\mathbf{F}_{k_\ell}$, $\ell \in \{2, 3, 4\}$. Same as above analysis, given $i_2 = i_3 = i_4$, $\mathbf{F}_{k_2}$, $\mathbf{F}_{k_3}$ and $\mathbf{F}_{k_4}$ are constructed by three different j_2, j_3 and j_4. We observe that, no matter how j_1 is selected in $\mathbf{F}_{k_1}$, $\mathrm{rank}(\mathbf{F}_{k_1,k_2,k_3,k_4}) = 4$. 2) i_1 appears twice in $\mathbf{F}_{k_1}$ and $\mathbf{F}_{k_3}$ and i_2 appears twice in $\mathbf{F}_{k_2}$ and $\mathbf{F}_{k_4}$. Same as above, given $i_1 = i_3$, $\mathbf{F}_{k_3}$ and $\mathbf{F}_{k_1}$ are constructed by two different j_1, j_3. Given $i_2 = i_4$, $\mathbf{F}_{k_4}$ and $\mathbf{F}_{k_2}$ are constructed by two different j_4, j_2. We observe that, no matter how j_4 and j_2 are selected, even $j_3 = j_4$, $j_1 = j_3$, $\mathrm{rank}(\mathbf{F}_{k_1,k_2,k_3,k_4}) = 4$.

4. If there are one i_ℓ for $\ell \in \{1\}$, that determines $\mathbf{F}_{k_\ell}$, $\ell \in \{1, 2, 3, 4\}$, same as above, four different j_ℓ for $\ell \in \{1, 2, 3, 4\}$ are needed to achieve four different entries k_ℓ, $\ell \in \{1, 2, 3, 4\}$. We have $\mathrm{rank}(\mathbf{F}_{k_1,k_2,k_3,k_4}) = 4$ in this case.

Thus, in our construction, for any distinct indices k_1, k_2, k_3, k_4 , $\mathrm{rank}(\mathbf{F}_{k_1,k_2,k_3,k_4})$ is always equal to 4. We conclude that for any distinct indices k_1, k_2, k_3, k_4 and any $a_1, a_2, a_3, a_4 \in \{0, 1\}$,

$$\Pr\left[\boldsymbol{f}_{k1} = a_1 \wedge \boldsymbol{f}_{k2} = a_2 \wedge \boldsymbol{f}_{k3} = a_3 \wedge \boldsymbol{f}_{k4} = a_4\right] = \frac{1}{16} \ .$$

□

C Proof of Theorem 4

Theorem 9 (Theorem 4, restated)**.** *Given a binary matrix $\mathbf{A} \in \{0,1\}^{N \times n}$, we define distribution $\mathcal{D}_{\mathbf{A}}$ over $\{-1, +1\}^N$ as follows: 1) sample a binary vector $\boldsymbol{s} \in \{0,1\}^n$ uniformly at random and 2) output $\boldsymbol{r} = 2(\mathbf{A}\boldsymbol{s} \bmod 2) - \mathbf{1}$. Let $d > 1$ is a constant, $n \geq 4d$ and $N \leq n^{d/2}$. For an overwhelming fraction of possible choices of $\mathbf{A}$ with each row having Hamming weight d and $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$, we have distribution $\mathcal{D}_{\mathbf{A}}$ is $\log\log(n)$-inadmissible.*

Proof. We begin by analyzing the properties under the condition that $\mathsf{HW}(\mathbf{A}_i) = \mathsf{HW}(\mathbf{A}_j) = d$ and $\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}$, where $t = \mathsf{HW}(\boldsymbol{v}) \neq 0$. More specifically, the following properties must hold:

1. Since both $\mathbf{A}_i$ and $\mathbf{A}_j$ have Hamming weight d, the Hamming weight of their XOR satisfies

$$t = \mathsf{HW}(\mathbf{A}_i \oplus \mathbf{A}_j) \leq \mathsf{HW}(\mathbf{A}_i) + \mathsf{HW}(\mathbf{A}_j) = 2d \ .$$

2. On the non-zero coordinates of $\boldsymbol{v}$, the vectors $\mathbf{A}_i$ and $\mathbf{A}_j$ must differ at every position. Consequently, among these t coordinates, each vector contains exactly $t/2$ ones, which contributes a factor of $\binom{t}{t/2}$ possible choices.

3. On the remaining $n-t$ coordinates, $\mathbf{A}_i$ and $\mathbf{A}_j$ must be identical and together contain exactly $d-t/2$ ones to preserve the Hamming weight constraint $\mathsf{HW}(\mathbf{A}_i) = \mathsf{HW}(\mathbf{A}_j) = d$, contributing a factor of $\binom{n-t}{d-t/2}$.

4. Consequently, $\mathsf{HW}(\boldsymbol{v})$ must be even, since $\mathbf{A}_i$ and $\mathbf{A}_j$ have the same Hamming weight, implying that the number of positions where the two vectors differ is even.

We now bound the correlation by analyzing the pairwise probability that two random weight-d vectors $\mathbf{A}_i$ and $\mathbf{A}_j$ satisfy $\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}$. The total sample space of $(\mathbf{A}_i, \mathbf{A}_j)$ has size $\binom{n}{d}^2 - \binom{n}{d}$ conditioned on $\mathsf{HW}(\mathbf{A}_i) = \mathsf{HW}(\mathbf{A}_j) = d$ and $\mathbf{A}_i \neq \mathbf{A}_j$. The number of valid pairs $(\mathbf{A}_i, \mathbf{A}_j)$ that $\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}$ is $\binom{t}{t/2}\binom{n-t}{d-t/2}$. Let

$$\mathcal{A} \stackrel{\text{def}}{=} \left\{ \boldsymbol{a} \in \{0,1\}^n \;\middle|\; \mathsf{HW}(\boldsymbol{a}) = d \right\} .$$

We define $p_{\boldsymbol{v}}$ as

$$p_{\boldsymbol{v}} \stackrel{\text{def}}{=} \Pr_{\mathbf{A}_i, \mathbf{A}_j \leftarrow \mathcal{A}, \mathbf{A}_i \neq \mathbf{A}_j} \left[\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v} \right] = \frac{\binom{t}{t/2}\binom{n-t}{d-t/2}}{\binom{n}{d}^2 - \binom{n}{d}} .$$

We show that for constant $2d \geq t$ and $n \geq 4d$, and for any nonzero vector $\boldsymbol{v}$, the probability $p_{\boldsymbol{v}}$ is maximized when $t = 2$. That is,

$$p_{\max} \stackrel{\text{def}}{=} \max_{\boldsymbol{v} \in \mathbb{F}_2^n \setminus \{\mathbf{0}_n\}} p_{\boldsymbol{v}} = \frac{2\binom{n-2}{d-1}}{\binom{n}{d}^2 - \binom{n}{d}} = \frac{2d(n-d)}{n(n-1)\left(\binom{n}{d} - 1\right)} \leq \frac{2d^{d+1}}{n^{d+1}} .$$

Since the denominators of $p_{\boldsymbol{v}}$ do not depend on t, maximizing $p_{\boldsymbol{v}}$ is equivalent to maximizing $F(t) \stackrel{\text{def}}{=} \binom{t}{t/2}\binom{n-t}{d-t/2}$. We now show that for every even $t \geq 2$,

$$\begin{aligned} \frac{F(t+2)}{F(t)} &= \frac{\binom{t+2}{t/2+1}}{\binom{t}{t/2}} \cdot \frac{\binom{n-t-2}{d-t/2-1}}{\binom{n-t}{d-t/2}} = \frac{4(t+1)}{t+2} \cdot \frac{(d-t/2)(n-d-t/2)}{(n-t)(n-t-1)} \\ &\leq 4 \cdot \frac{d-t/2}{n-t-1} \leq 4 \cdot \frac{d-t/2}{4d-t-1} \leq 1 . \end{aligned}$$

For any non-zero $\boldsymbol{v} \in \{0,1\}^n$, we define

$$\mathcal{S}_{\boldsymbol{v}} \stackrel{\text{def}}{=} \left\{ (i,j) \in [N]^2 \mid \mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v} \right\}$$

Thus, we obtain a partition of $\left\{ (i,j) \in [N]^2 \mid i \neq j \right\}$ as

$$\left\{ (i,j) \in [N]^2 \mid i \neq j \right\} = \dot{\bigcup}_{\boldsymbol{v} \neq \mathbf{0}} \mathcal{S}_{\boldsymbol{v}} .$$

For nonzero $\boldsymbol{v}$, each $\mathcal{S}_{\boldsymbol{v}}$ is an inadmissible set. By definition of $\mathbf{A}$, for any $i \neq j$, we have $\mathbf{A}_i \neq \mathbf{A}_j$. Therefore, for any $i \neq j$, we have

$$\begin{aligned} \mathbb{E}_{\boldsymbol{r} \leftarrow \mathcal{D}} \left[\boldsymbol{r}[i]\boldsymbol{r}[j] \right] &= \mathbb{E}_{\boldsymbol{s} \leftarrow \{0,1\}^n} \left[(-1)^{\mathbf{A}_i \boldsymbol{s}} (-1)^{\mathbf{A}_j \boldsymbol{s}} \right] \\ &= \mathbb{E}_{\boldsymbol{s} \leftarrow \{0,1\}^n} \left[(-1)^{\langle \mathbf{A}_i \oplus \mathbf{A}_j, \boldsymbol{s} \rangle} \right] = 0 . \end{aligned}$$

Moreover, for any $(i,j), (k,l) \in \mathcal{S}_{\boldsymbol{v}}$,

$$\begin{aligned} \mathbb{E}_{\boldsymbol{r} \leftarrow \mathcal{D}} \left[\boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{r}[k]\boldsymbol{r}[l] \right] &= \mathbb{E}_{\boldsymbol{s} \leftarrow \{0,1\}^n} \left[(-1)^{(\mathbf{A}_i \oplus \mathbf{A}_j \oplus \mathbf{A}_k \oplus \mathbf{A}_l)\boldsymbol{s}} \right] \\ &= \mathbb{E}_{\boldsymbol{s} \leftarrow \{0,1\}^n} \left[(-1)^{\langle \boldsymbol{v} \oplus \boldsymbol{v}, \boldsymbol{s} \rangle} \right] = 1 . \end{aligned}$$

Finally, for any $\boldsymbol{u} \neq \boldsymbol{v}$ and any $(i,j) \in \mathcal{S}_{\boldsymbol{u}}$, $(k,l) \in \mathcal{S}_{\boldsymbol{v}}$,

$$\begin{aligned} \mathbb{E}_{\boldsymbol{r} \leftarrow \mathcal{D}} \left[\boldsymbol{r}[i]\boldsymbol{r}[j]\boldsymbol{r}[k]\boldsymbol{r}[l] \right] &= \mathbb{E}_{\boldsymbol{s} \leftarrow \{0,1\}^n} \left[(-1)^{(\mathbf{A}_i \oplus \mathbf{A}_j \oplus \mathbf{A}_k \oplus \mathbf{A}_l)\boldsymbol{s}} \right] \\ &= \mathbb{E}_{\boldsymbol{s} \leftarrow \{0,1\}^n} \left[(-1)^{\langle \boldsymbol{v} \oplus \boldsymbol{u}, \boldsymbol{s} \rangle} \right] = 0 . \end{aligned}$$

We then consider the tail probability $\Pr[|\mathcal{S}_{\boldsymbol{v}}| \geq k]$ for a fixed nonzero $\boldsymbol{v}$ and any positive integer k. Let $(x)_k \stackrel{\text{def}}{=} x(x-1)\cdots(x-k+1)$ denote the falling factorial. Define the indicator variable $X_{(i,j)}$ such that $X_{(i,j)} = 1$ if and only if $\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}$, and $X_{(i,j)} = 0$ otherwise. Then

$$|\mathcal{S}_{\boldsymbol{v}}| = \sum_{i \neq j} X_{(i,j)} \ .$$

If $|\mathcal{S}_{\boldsymbol{v}}| \geq k$, then the falling factorial $(|\mathcal{S}_{\boldsymbol{v}}|)_k \geq k!$. Therefore, by Markov's inequality,

$$\Pr\Big[|\mathcal{S}_{\boldsymbol{v}}| \geq k\Big] \ \leq \ \Pr\Big[(|\mathcal{S}_{\boldsymbol{v}}|)_k \geq k!\Big] \ \leq \ \frac{\mathbb{E}[(|\mathcal{S}_{\boldsymbol{v}}|)_k]}{k!} \ .$$

For any collection of indicator variables $\{X_a\}_a$, the combinatorial identity

$$\left(\sum_a X_a\right)_k = \sum_{(a_1,\ldots,a_k)\,\text{distinct}} X_{a_1}\cdots X_{a_k}$$

holds. This identity follows since $\sum_a X_a$ counts the number of indices such that $X_a = 1$. Consequently, the falling factorial $(\sum_a X_a)_k$ equals the number of ways to choose k ordered distinct indices from those satisfying $X_a = 1$. The right-hand side enumerates exactly these choices, because the product $X_{a_1}\cdots X_{a_k}$ equals 1 if and only if all selected indices are distinct and each satisfies $X_{a_r} = 1$; otherwise, the product is 0. Therefore, the two sides of the identity count the same quantity. Applying this identity with $a_r = (i_r, j_r)$ with $i_r \neq j_r$, we have

$$(|\mathcal{S}_{\boldsymbol{v}}|)_k = \sum_{(a_1,\ldots,a_k)\,\text{distinct}} X_{a_1}\cdots X_{a_k} \ .$$

Taking the expectation gives

$$\mathbb{E}\left[(|\mathcal{S}_{\boldsymbol{v}}|)_k\right] = \sum_{\big((i_1,j_1),\ldots,(i_k,j_k)\big)\,\text{distinct}} \Pr\left[\bigwedge_{r=1}^{k} (\mathbf{A}_{i_r} \oplus \mathbf{A}_{j_r} = \boldsymbol{v})\right] \ .$$

Let us fix k distinct pairs for the analysis that follows. Note that for any pair (i,j) satisfying $\mathbf{A}_i \oplus \mathbf{A}_j = \boldsymbol{v}$, the reversed pair (j,i) also satisfies the same condition. Hence, each unordered pair $\{i,j\}$ corresponds to at most two ordered pairs. Without loss of generality, we may restrict attention to a set of $k/2$ unordered pairs, denoted by $\{i_1,j_1\},\ldots,\{i_{k/2},j_{k/2}\}$ with $i_t > j_t$ for all t. Let set $\mathcal{I} \stackrel{\text{def}}{=} \{i_1,j_1,\cdots,i_{k/2},j_{k/2}\}$ and $m \stackrel{\text{def}}{=} |\mathcal{I}|$ that denotes the number of distinct indices involved. The subsequent analysis separates the contribution into two cases. We first analyze the good case, corresponding to $m = k$, where all indices are distinct. The bad case, where $m < k$ due to index collisions, will be ruled out later based on the structural property that for all $i \neq j$, we have $\mathbf{A}_i \neq \mathbf{A}_j$.

Good Case: When $m = 2k$, all indices in $(i_1,j_1,\cdots,i_{k/2},j_{k/2})$ are distinct. Since the rows of $\mathbf{A}$ are sampled without replacement (and hence are pairwise distinct), they are not independent. Nevertheless, we can upper-bound the joint probability using a sequential-exposure argument. Specifically,

$$\Pr\Big[\bigwedge_{r=1}^{k/2} (\mathbf{A}_{i_r} \oplus \mathbf{A}_{j_r} = \boldsymbol{v})\Big] = \prod_{r=1}^{k/2} \Pr\Big[\mathbf{A}_{i_r} \oplus \mathbf{A}_{j_r} = \boldsymbol{v} \ \Big|\ \{\mathbf{A}_{i_s} \oplus \mathbf{A}_{j_s} = \boldsymbol{v}\}_{s<r}\Big] \ .$$

For each r, the conditional probability is at most $p_{\max}$, since conditioning only restricts the set of available rows. Hence,

$$\Pr\Big[\bigwedge_{r=1}^{k/2} (\mathbf{A}_{i_r} \oplus \mathbf{A}_{j_r} = \boldsymbol{v})\Big] \leq p_{\max}^{k/2} \ .$$

The number of ordered tuples $(i_1,j_1,\cdots,i_{k/2},j_{k/2})$ is at most N^k. Therefore, the total contribution from all good tuples satisfies

$$\begin{aligned}\mathbb{E}\big[(|\mathcal{S}_{\boldsymbol{v}}|)_k \mid m = 2k\big] &\leq N^k \cdot p_{\max}^{k/2} \leq n^{d/2\cdot k} \cdot \Big(\frac{2d^{d+1}}{n^{d+1}}\Big)^{k/2} \\ &\leq 2^{k/2} \cdot d^{(d+1)k/2} \cdot n^{-k/2} \ .\end{aligned}$$

		$n = 128$	$n = 256$	$n = 512$	$n = 1024$
Comp. (s)	LAN	3.1	5.7	10.8	20.7
	WAN	16.9	31.8	60.6	118.7
Comm. (MB)		149.2	298.4	596.7	1193.4
Publish (MB)		155.5	304.7	603.0	1199.7

Server initialization

m	2^0	2^4	2^8	2^{12}	2^{16}
$d = 4$	100.8	95.4	86.3	41.2	27.5
$d = 5$	127.6	122.6	108.6	66.5	54
$d = 6$	161.6	155.7	127.6	65.9	53.8
$d = 7$	188.8	182.2	149.9	91.6	79.8
$d = 8$	223.6	211.4	160.8	90.7	80.1

Client sketching (ms/elem.)

		Aggregate Encrypted Sketch	Threshold Decryption	2PC	Total
Comp. (s)	LAN	0.83	0.04	0.1	0.97
	WAN	0.83	0.05	2.39	3.27
Comm. (MB)		0	2.6	12.9	15.5

Server aggregation and output

Table 5: Protocol efficiency under polynomial modulus degree 16384.

Bad Case. Suppose $m < k$, i.e., some indices appear in multiple pairs. We show that this case cannot occur under the assumption that $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$.

Construct an undirected graph G over vertex set $[N]$, where each constraint $\mathbf{A}_{i_r} \oplus \mathbf{A}_{j_r} = \boldsymbol{v}$ introduces an edge (i_r, j_r).

If some connected component of G contains at least two edges, then there exists a path of length at least 2, say (u, u_1) and (u_1, u_2). These imply

$$\mathbf{A}_{u_1} = \mathbf{A}_u \oplus \boldsymbol{v} \quad \text{and} \quad \mathbf{A}_{u_2} = \mathbf{A}_{u_1} \oplus \boldsymbol{v} = \mathbf{A}_u \ .$$

Since $u_2 \neq u$, this contradicts the assumption that $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$.

Therefore, every connected component contains at most one edge, which implies $m = k$. Hence, the bad case cannot occur.

Combining the good contribution and the bad contribution, we have

$$\begin{aligned}\mathbb{E}\Big[(|\mathcal{S}_{\boldsymbol{v}}|)_k\Big] &= \mathbb{E}\Big[(|\mathcal{S}_{\boldsymbol{v}}|)_k \;\Big|\; m = 2k\Big] + \mathbb{E}\Big[(|\mathcal{S}_{\boldsymbol{v}}|)_k \;\Big|\; m \leq 2k\Big] \\ &\leq 2^{k/2} \cdot d^{(d+1)k/2} \cdot n^{-k/2} \ .\end{aligned}$$

By the factorial moment method and Markov's inequality, setting $k = \log\log(n)$ yields $\frac{\mathbb{E}[(|\mathcal{S}_{\boldsymbol{v}}|)_k]}{k!}$ is negligible, and consequently

$$\begin{aligned}\Pr\Big[|\mathcal{S}_{\boldsymbol{v}}| \geq \log\log(n)\Big] &\leq \Pr\Big[\big(|\mathcal{S}_{\boldsymbol{v}}|\big)_{\log\log(n)} \geq (\log\log(n))!\Big] \\ &\leq \mathsf{negl}(n) \ .\end{aligned}$$

Note that $\mathsf{HW}(\mathbf{A}_i \oplus \mathbf{A}_j)$ is at most $2d$ and must be even. Hence, the number of nonzero vectors $\boldsymbol{v} \neq 0$ that can appear is

$$\sum_{1 \leq t \leq 2d,\ t \text{ is even}} \binom{n}{t} \leq n^{2d} \ .$$

Applying the union bound and using the fact that d is constant, we get

$$\begin{aligned}\Pr\Big[\Delta(\mathcal{D}_{\mathbf{A}}) \geq \log\log(n)\Big] &\leq \sum_{\boldsymbol{v} \neq \mathbf{0}} \Pr\Big[|\mathcal{S}_{\boldsymbol{v}}| \geq \log\log(n)\Big] \\ &\leq n^{2d} \cdot \mathsf{negl}(n) \leq \mathsf{negl}(n) \ .\end{aligned}$$

□

D Security Proof

Theorem 10 (Theorem 6, restated)**.** *Let H be a random oracle, and assume that the SWHE scheme satisfies ciphertext pseudorandomness. Then, Protocol Π_{PJC} (Figure 4) securely realizes the functionality for private join and compute over distributed input sets in the presence of semi-honest adversaries in the $(\mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{ThSWHE}}, \mathcal{F}_{\mathsf{2PC}})$-hybrid model.*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) adversary that corrupts at most one of the servers, namely Agg or CSP, possibly colluding with a subset of the parties. We construct a PPT simulator $\mathcal{S}$, which runs the PPT adversary $\mathcal{A}$ as a subroutine and emulates functionalities $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{ThSWHE}}$ and $\mathcal{F}_{\mathsf{2PC}}$. $\mathcal{S}$ simulates the random oracle H by responding to random strings while keeping the consistency of responses. Let $\mathcal{C} \subseteq \{P_i\}_{i\in[w]}$ be the corrupted parties. $\mathcal{S}$ proceeds as follows.

Initialization:

1. $\mathcal{S}$ emulates the $\mathcal{F}_{\mathsf{ThSWHE}}$ by executing $(\mathsf{params}', \mathsf{sk}', \mathsf{pk}', \mathsf{ek}') \leftarrow \mathsf{KeyGen}(1^\lambda, 1^L)$, and sending $(\mathsf{ek}', \mathsf{pk}')$ to $\mathcal{A}$. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Rand}}$ by sampling a random $seed \leftarrow \{0,1\}^\lambda$. $\mathcal{S}$ sends $seed$ to $\mathcal{A}$.
2. For each $\ell \in [tk]$, $\mathcal{S}$ samples $\boldsymbol{s}^\ell \in \{0,1\}^n$ and generates a fresh encryption $[\![0]\!]$. If the corrupted server is CSP, $\mathcal{S}$ sends $[\![0]\!]$ to $\mathcal{A}$; otherwise, it publishes $[\![0]\!]$.
3. For each $\ell \in [tk]$, $\mathcal{S}$ generates two counters, c_1^ℓ and c_2^ℓ, initialized to 0.

Union operation over the set of P on $(\mathsf{input}, p, \mathcal{S})$:

1. If $P \notin \mathcal{C}$ and Agg is corrupted, $\mathcal{S}$ sends a fresh $[\![0]\!]$ to $\mathcal{A}$.
2. $\mathcal{S}$ samples a matrix $\mathbf{A} \in \{0,1\}^{N\times n}$ uniformly at random subject to the constraint that each row has Hamming weight exactly d and all rows are pairwise distinct.
3. For $\ell \in [tk]$, $\mathcal{S}$ computes its sketching result:

$$z^\ell = \sum_{i\in[m]} b_i \boldsymbol{r}^\ell[\mathbf{A}_i] \ ,$$

   where $\boldsymbol{r}^\ell[i] = 2(\mathbf{A}_i \boldsymbol{s}^\ell \mod 2) - 1$.
4. For $\ell \in [tk]$, $\mathcal{S}$ updates $c_p^\ell \leftarrow c_p^\ell + z^\ell$, where $p \in \{1,2\}$.

Join Computation via 2PC on Servers:

1. For each $\ell \in [tk]$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{ThSWHE}}$'s Dec command by uniformly sampling $d_1^\ell, d_2^\ell \in \mathcal{M}$ and sending (d_1^ℓ, d_2^ℓ) to $\mathcal{A}$.
2. If the corrupted server is Agg, $\mathcal{S}$ emulates $\mathcal{F}_{\mathrm{2PC}}$ by sending $z = \mathsf{MoM}(\{c_1^\ell c_2^\ell\}^{\ell\in[tk]})$ to $\mathcal{A}$.

We use a sequence of hybrids to show that real-world and ideal-world executions are computationally indistinguishable.

Hybrid 0**.** This hybrid (denoted by $\mathcal{G}_0$) is the real-world execution.

Hybrid 1**.** This hybrid (denoted by $\mathcal{G}_1$) is the same as $\mathcal{G}_0$, except that emulating $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{ThSWHE}}$ and $\mathcal{F}_{\mathsf{2PC}}$, and simulating random oracle H honestly.

It is obvious that $\mathcal{G}_1$ is perfectly indistinguishable from $\mathcal{G}_0$. Hybrid $\mathcal{G}_1$ can obtain all secrets of $\mathcal{A}$ by emulating $\mathcal{F}_{\mathsf{ThSWHE}}$.

Hybrid 2**.** This hybrid (denoted by $\mathcal{G}_2$) is identical to $\mathcal{G}_1$, except that $z = \mathsf{MoM}(\{c_1^\ell c_2^\ell\}^{\ell\in[tk]})$ is computed in the plaintext.

We prove that $\mathcal{G}_2$ is statistically indistinguishable from $\mathcal{G}_1$ by the decryption correctness of the underlying SWHE scheme.

Hybrid 3**.** This hybrid (denoted by $\mathcal{G}_3$) is identical to $\mathcal{G}_2$, except that every ciphertext sent to $\mathcal{A}$ is replaced with a fresh encryption of 0. This corresponds to the ideal-world execution.

We prove that $\mathcal{G}_3$ is computationally indistinguishable from $\mathcal{G}_2$ via a reduction to the ciphertext pseudorandomness property of the underlying SWHE scheme.

In conclusion, the joint distribution of the outputs of $\mathcal{S}$ and honest participants in the ideal-world execution is computationally indistinguishable from that of the outputs of $\mathcal{A}$ and honest participants in the real-world execution, which completes the proof. □

E Performance under Large Polynomial Modulus Degree

We evaluate the performance under larger polynomial modulus degree 16384, where ciphertexts become larger and each homomorphic multiplication becomes more expensive. We report the results for plain modulus 2^{24} across the three protocol phases in Table 5. In the server initialization phase, when $n = 128$, the LAN runtime is 3.1s, with 149.2 MB of server communication and 155.5 MB of broadcast communication. The cost scales approximately linearly with n and is roughly $3\times$ higher than that under polynomial modulus degree 8192. For client local sketching, the amortized cost is 90.7 ms per element when $d = 8$, also about $3\times$ higher than the cost under degree 8192. The client-to-server communication increases to 0.71 MB. In the server aggregation and output phase, both homomorphic aggregation and threshold decryption become more expensive due to the increased ciphertext size. For encrypted sketch aggregation over 2000 clients, the aggregation time is 0.83 s, while LAN threshold decryption takes 0.04 s. The subsequent 2PC computation incurs additional overhead due to the use of a multi-limb representation to prevent overflow under large payloads, resulting in a $4\times$ cost increase, corresponding to 0.1 s of LAN time and 12.9 MB of communication.