Bifrost: A Much Simpler Secure Two-Party Data Join Protocol for Secure Data Analytics

Shuyu Chen
Laboratory for Data Security and Governance, Fudan University
23110240005@m.fudan.edu.cn

Mingxun Zhou
The Hong Kong University of Science and Technology
mingxunz@ust.hk

Haoyu Niu
Laboratory for Data Security and Governance, Fudan University
23212010019@m.fudan.edu.cn

Guopeng Lin
Laboratory for Data Security and Governance, Fudan University
17302010022@fudan.edu.cn

Weili Han*
Laboratory for Data Security and Governance, Fudan University
wlhan@fudan.edu.cn

ABSTRACT

Secure data join enables two parties with vertically distributed data to securely compute the joined table, allowing them to perform downstream Secure multi-party computation-based Data Analytics (SDA), such as analyzing statistical information or training machine learning models, based on the joined table. While Circuit-based Private Set Intersection (`CPSI`) can be used for secure data join, it inherently introduces redundant dummy rows in the joined table, which results in high overhead in the downstream SDA tasks. `iPrivJoin` addresses this issue but introduces significant communication overhead in the redundancy removal process, as it relies on the cryptographic primitive Oblivious Programmable Pseudorandom Function (OPPRF) and multiple rounds of oblivious shuffles.

In this paper, we propose a much simpler secure data join protocol, `Bifrost`, which outputs (the secret shares of) a redundancy-free joined table. The highlight of `Bifrost` lies in its simplicity: it builds upon two conceptually simple building blocks, an ECDH-PSI protocol and a two-party oblivious shuffle protocol. The lightweight protocol design allows `Bifrost` to avoid the need for OPPRF. We also proposed a simple optimization named *dual mapping* that reduces the rounds of oblivious shuffle needed from two to one. Experiments on various datasets up to 100 GB show that `Bifrost` achieves 2.54 ~ 22.32× speedup and reduces the communication by 84.15% ~ 88.97% compared to the state-of-the-art redundancy-free secure data join protocol `iPrivJoin`. In the two-step SDA pipeline (secure join and secure analytics) experiments, the redundancy-free property of `Bifrost` not only avoids the catastrophic error rate blowup in the downstream analytics caused by dummy rows introduced by `CPSI`, but also shows up to 2.80× speed-up and up to 73.15% communication reduction in the secure analytics process.

PVLDB Reference Format:
Shuyu Chen, Mingxun Zhou, Haoyu Niu, Guopeng Lin, and Weili Han. Bifrost: A Much Simpler Secure Two-Party Data Join Protocol for Secure Data Analytics. PVLDB, 19(6): 1156 - 1169, 2026.
doi:10.14778/3797919.3797925

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 6 ISSN 2150-8097.
doi:10.14778/3797919.3797925

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at: https://github.com/stellasuc/secdjoin.git

1 INTRODUCTION

Secure Multi-Party Computation (SMPC) enables multiple parties to jointly compute some functions over their data while keeping their data private. By leveraging SMPC, parties can perform various Secure Data Analytics (SDA) tasks, including performing statistical analysis on their distributed data or even training a machine learning model, which helps address privacy-related data compliance requirements, such as GDPR [62] and CCPA [7].

Vertically distributed data is a common setting in real-world SDA applications, appearing in many privacy-critical scenarios, including finance [14, 26], e-commerce [13, 63], and healthcare [4, 64]. That is, the parties' data tables overlap in the indexing IDs but have disjoint feature columns (see Figure 1). For example, consider a hospital that is cooperating with a genomic research company on training a cancer risk assessment model based on genetic data. The hospital holds the health record table of its patients, and the genomic research company holds the gene description table of its clients. Both tables are indexed by the individuals' unique IDs. They will need to first *align* the data by the IDs and build a training dataset that contains only the records of those individuals who appear in both original tables. In the traditional setting with no privacy constraint, the parties can run a classical distributed *join* operation [8, 31, 52] on the tables and obtain a joined table that cross-matches the records in the original tables based on IDs and concatenates the feature columns. They can then proceed to train their models based on the data samples in this joined table.

In our targeted privacy-constrained setting, a *secure join* protocol is required instead of a traditional distributed join algorithm, given that these algorithms inherently leak sensitive information about one party's data table to the other party. The secure join protocol ensures that the computation process of the join operation reveals only the absolutely necessary information to both parties (e.g., the size of the final joined table), and outputs the joined table in secret-shared form to both parties, so neither party learns the full, privacy-sensitive joined table.

Existing secure data join schemes either introduce significant computation and/or communication costs, or pass down nonideal

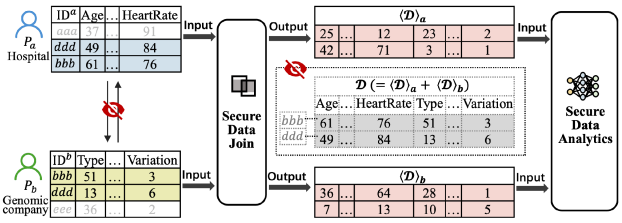


Figure 1: An example of a vertically distributed data setting. P_a and P_b each hold a table with three records. Both tables contain the intersection IDs "*bbb*" and "*ddd*".

overhead to the downstream SDA tasks. On the one hand, Circuit-based Private Set Intersection (CPSI) [50, 51] can be used for secure data join, but the protocols will inherently introduce many redundant dummy rows in the joined table (padded to the maximum length) other than the actual matched rows, which could be a huge waste in many practical scenarios. Liu *et al.* [39] showed that the redundant rows generated by CPSI may cause both downstream SDA tasks' time and communication overhead to increase by up to 3×, and proposed another state-of-the-art (SOTA) secure data join method named iPrivJoin. However, to remove the redundant rows, iPrivJoin introduces excessive communication overhead in the secure data join process, which stems from its reliance on Oblivious Programmable Pseudorandom Function (OPPRF) [51] for data encoding and two rounds of oblivious shuffle. Apart from that, iPrivJoin also requires 11 rounds of communication between the two parties, becoming another disadvantage for its practicality. Our experiments show that for the table size of 100 GB under WAN setting, iPrivJoin requires 53.87 hours in time cost and 773.95 GB in communication cost.

As a versatile building block, a secure join protocol can be applied to secure data mining [1, 37], SQL operators [5, 60], and data visualizations [3, 47] applications, in addition to the aforementioned SDA tasks. Given the importance of an efficient secure join protocol in these applications, we ask the following question:

> *How to construct a redundancy-free secure data join protocol with high computation efficiency and low communication cost?*

1.1 Our Contributions

In this paper, we propose Bifrost, a conceptually simple secure data join protocol with better performance compared to the baselines. We summarize our main contributions as follows:

- We propose a new secure two-party data join protocol named Bifrost that outputs (the secret shares of) the redundancy-free joined table. The highlight of Bifrost lies in its simplicity: Bifrost builds upon two conceptually simple building blocks, an Elliptic-Curve-Diffie-Hellman-based Private Set Intersection (ECDH-PSI) protocol [28, 43] and a two-party oblivious shuffle protocol [39]. The lightweight protocol design allows Bifrost to avoid many performance bottlenecks in iPrivJoin, including the need for Cuckoo hashing and OPPRF. We also propose a simple optimization named *dual mapping* that reduces the rounds of oblivious shuffle (as in iPrivJoin) needed from two to one.
- We provide a theoretical analysis of Bifrost's asymptotic performance and show that it outperforms the baselines in nearly all important metrics, including computation, communication, and round complexities. We also provide the rigorous security proof for Bifrost to meet the privacy requirement.
- We extensively evaluate the effectiveness of Bifrost in the pipeline of SDA tasks, including secure statistical analysis and secure training. First, we show that our redundancy-free design of Bifrost allows the downstream SDA tasks to achieve high-accuracy outputs, avoiding the catastrophic error rate blowup caused by CPSI-based solution [50, 51]. Moreover, the design helps the downstream SDA tasks achieve up to 2.80× improvement in running time with up to 73.09% less communication compared to pipelines using CPSI.
- We empirically evaluate Bifrost on various datasets up to 100 GB and compare its performance against the baselines. We show that, on real-world datasets, Bifrost achieves up to 15.2× and up to 10.36× speedups compared to iPrivJoin [39] and CPSI [50], respectively. In addition, compared to the redundancy-free secure data join baseline iPrivJoin, Bifrost achieves 2.54 ∼ 22.32× faster running time and reduces communication size by 84.15% ∼ 88.97%. Moreover, the advantages of Bifrost become more pronounced as the feature dimension increases. For instance, when the feature dimension varies from 100 to 6400, the running time improvement increases from 9.58× to 21.46×. Notably, the communication size of Bifrost is nearly equal to the size of the input data.

1.2 Design Goal and Technical Overview

Design Goal. We aim to design a secure two-party data join protocol, Bifrost. To clarify this goal, we first define the two-party data join operation [5, 49]. Consider two tables, Ta and Tb, held by two parties P_a and P_b, respectively. Each table consists of identifiers and corresponding feature columns. A two-party data join operation inputs Ta and Tb and outputs a joined table $\mathcal{D}$ by identifying the matched identifiers across both tables and concatenating their respective feature columns. This join operation can be expressed in SQL as follows:

```
SELECT Ta.* EXCEPT (identifiers), Tb.* EXCEPT (identifiers)
FROM Ta JOIN Tb
ON Ta.identifiers = Tb.identifiers
```

The *secure* two-party data join protocol, Bifrost, ensures that this join operation is performed securely. Here, "secure" means that Bifrost reveals only the size of the joined table during the computation process of the join operation, and outputs (secret) shares of the joined table to both parties (one share per party). Note that each joined table share is uniformly random, so it reveals nothing about the joined table value or the matched identifiers. Furthermore, these shares enable the two parties to directly perform downstream secure data analytics on the joined table shares using secure multi-party computation techniques.

Simplified Setting Description. To show how Bifrost achieves this design goal, we now provide a technical overview in a simplified setting. There are two tables consisting of identifiers and the corresponding feature columns, $[(ID_i^a, F_i^a)]_{i\in[n]}$ and $[(ID_i^b, F_i^b)]_{i\in[n]}$, held by party P_a and P_b, respectively. Both parties want to run some downstream task on the *joined table*, that is, $\{(F_i^a, F_i^b)\}_{i\in I}$ where

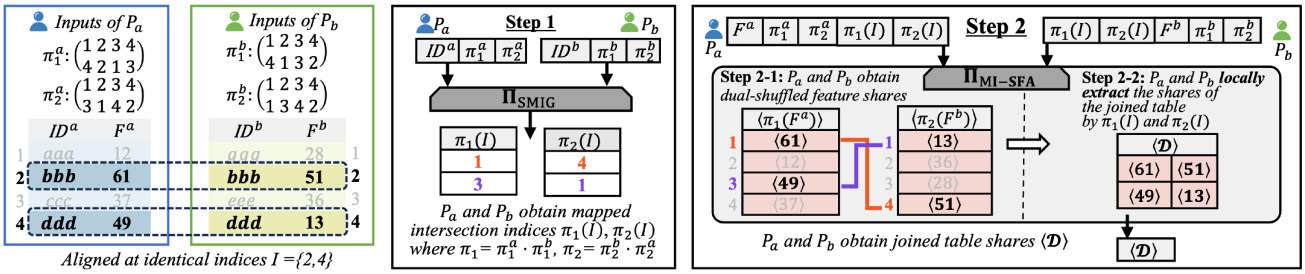


Figure 2: Simplified workflow of `Bifrost` with *Dual Mapping* optimization. The intersection IDs and their features are highlighted in bold. In the first step, both parties obtain mapped intersection indices $\pi_1(I)$ and $\pi_2(I)$. In the second step, both parties first obtain dual-shuffled feature shares $\langle \pi_1(F^a) \rangle$ and $\langle \pi_2(F^b) \rangle$ via one round of $\Pi_{\text{O-Shuffle}}$, then both parties locally extract the shares of the joined table $\mathcal{D}$ from $\langle \pi_1(F^a) \rangle$ and $\langle \pi_2(F^b) \rangle$ according to $\pi_1(I)$ and $\pi_2(I)$.

the list I includes all matched indices: $I = [i \mid ID_i^a = ID_i^b]$. We focus on designing an interactive protocol between the parties that outputs secret shares of the joined table to both parties (one share per party) without exposing the intersection index set I to either party.

For simplicity, we make the assumption here that the tables are "aligned", i.e., if an identifier appears in both tables, the corresponding rows must share the same relative location in the lists. This "aligned" simplicity allows us to focus on the challenge of how to identify I and extract the corresponding feature pairs $\{(F_i^a, F_i^b)\}_{i \in I}$ in secret-shared form without revealing I itself. The unaligned case, which is typically the standard definition of the secure join problem, can be accommodated through protocol-specific modifications.

Workflow of `Bifrost`. The two major steps are:

(1) *Secure mapped intersection generation (SMIG).* In this step, the two parties obliviously find all the matched indices I where $ID_i^a = ID_i^b$ for $i \in I$. For privacy, neither party can learn the indices I directly. Instead, our proposed protocol outputs a *mapped* version of the matched indices I, denoted as $\pi(I)$, where $\pi : [n] \mapsto [n]$ is a mapping (shuffling) unknown to both parties. We set π as the composition of two shuffles π^a, π^b, each only known to party P_a and P_b respectively. This ensures that both parties learn only the size of the intersection (i.e., row count of the joined table) and nothing else. To securely instantiate this step, we adapt an Elliptic-Curve-Diffie-Hellman-based Private Set Intersection (ECDH-PSI) protocol [28, 43] to our setting with several technical changes.

(2) *Secure feature alignment.* The second step is straightforward: the parties jointly run an oblivious shuffling protocol [39] to obtain the shares of $\pi(F^a)$ and $\pi(F^b)$[1]. Then, both parties can locally extract the shares of the joined table. That is, they collect the shares of $\{(\pi(F^a)_i, \pi(F^b)_i)\}_{i \in \pi(I)}$, which is exactly $\{(F_i^a, F_i^b)\}_{i \in I}$.

Example: the two parties hold $[(aaa, 12), (bbb, 61), (ccc, 37), (ddd, 49)]$ and $[(ggg, 28), (bbb, 51), (eee, 36), (ddd, 13)]$, respectively. The matched indices are $I = [2, 4]$, corresponding to the intersection identifers "bbb" and "ddd". Given a shuffle $\pi : (1 \to 3,\ 2 \to 1,\ 3 \to 4, 4 \to 2)$ unknown to both parties, the secure mapped intersection generation protocol outputs the mapped intersection indices $\pi(I) = \pi([2, 4]) = [1, 2]$. Then, both parties run the oblivious shuffle protocol [39] to shuffle their local feature columns according to π and obtain the shares of the two shuffled feature columns: $[61, 49, 12, 37]$ and $[51, 13, 28, 36]$. Now, given the mapped intersection indices $[1, 2]$, they simply collect the first and the second entries in the local shares, which are the shares of the concatenated intersection features: $[(61, 51), (49, 13)]$.

Optimization: Dual Mapping. A performance bottleneck of the aforementioned workflow is the secure shuffle protocol in the second step. The two-party oblivious shuffle protocol $\Pi_{\text{O-Shuffle}}$ [39] is designed for the following functionality: given the input of a data column (from one party) and a shuffle (from another party), the protocol outputs the shares of the shuffled data column to both parties. Recall that the shuffle π is a composition of π^a and π^b (first applying π^a, then applying π^b). When the parties run the oblivious shuffle protocol to shuffle the feature columns of F^a, P_a can first locally shuffle the feature columns according to π^a. Then, they can directly call $\Pi_{\text{O-Shuffle}}$ given the self-shuffled features $\pi^a(F^a)$ from P_a and the shuffle π^b from P_b to obtain the share of $\pi(F^a)$. However, when they need to shuffle the column of F^b, P_b cannot shuffle the column locally according to π^b first, because the shuffling operation is not commutative (i.e., $\pi = \pi^a \circ \pi^b$ may not be the same as $\pi^b \circ \pi^a$). A naive solution could be running the $\Pi_{\text{O-Shuffle}}$ twice, resulting in high communication overhead.

We propose a simple yet effective optimization called *Dual Mapping* to address this issue. That is, we now modify the secure mapped intersection generation (SMIG) protocol to output *two* mapped versions of the intersection indices I with two different mappings, π_1 and π_2. Importantly, we let $\pi_1 = \pi_1^a \circ \pi_1^b$ and $\pi_2 = \pi_2^b \circ \pi_2^a$ where π_1^a, π_2^a are only known to P_a, and π_1^b, π_2^b are only known to P_b. Then, in the oblivious shuffle protocol execution later, P_a's feature columns will be shuffled by π_1, while P_b's feature columns will be shuffled by π_2. A simplified workflow of `Bifrost` with the *Dual Mapping* optimization is shown in Figure 2. The advantage is that now both first layers of the shuffles can be done by the corresponding party locally, and we only need to execute the oblivious shuffle protocol once on both sides. Effectively, this optimization saves $O(m_b)$ online communication overhead compared to the naive solution, where m_b denotes the feature dimension for P_b.

[1]We abuse the notation here that $\pi(F^a)_{\pi(i)} = F_i^a$ for all $i \in [n]$.

Supporting the General Scenario. In practice, the tables from both parties are usually not aligned, i.e., the rows with the same identifier may not be in the same relative location in the two tables. The prior standard solutions to this issue, including `CPSI` [50] and `iPrivJoin` [39], use a combination of Cuckoo hashing [48] and simple hashing to place the data elements in the same relative location in the hash tables. This further complicates the protocol design and affects the overall performance.

Instead, our protocol is naturally capable of handling the general scenario without requiring any hashing technique involved, saving another $O(hm_b\kappa)$ in computation cost and $O(hm_b\kappa)$ in communication cost, where κ is the computational security parameter and h is the number of hashing functions. More specifically, our protocol simply outputs the mapped intersection index pairs $\mathsf{MIPairs} = \{(\pi_1(i), \pi_2(j)) \mid ID_i^a = ID_j^b\}$ in the first step, and proceeds as before. We demonstrate that this upgrade can be implemented with a minor modification to our aforementioned ECDH-PSI-based [43] secure mapped intersection generation protocol, incurring no additional overhead.

Comparison with the Prior State-of-the-art Scheme. Compared to the prior best scheme `iPrivJoin`, our proposed `Bifrost` achieves significantly better efficiency in terms of communication and computation overhead by fundamentally restructuring the workflow. Specifically, `iPrivJoin` operates in three steps: (1) private data encoding: the parties use Cuckoo hashing or simple hashing tables for data allocation and OPPRF [33] for data encoding, obtaining shares of (B, F^a, F^b) with $B_i = 0$ for intersection rows and a random value otherwise; (2) oblivious shuffle: the two parties employ two rounds of an oblivious shuffle protocol on shares of (B, F^a, F^b) to obtain shares of $\pi(B, F^a, F^b)$. Here, π is the composite of two shuffles π^a, π^b, each only known to P_a and P_b, respectively. (3) private data trimming: the two parties reconstruct $\pi(B)$ in plaintext and remove redundant rows from shares of $\pi(F^a, F^b)$ according to $\pi(B)$. The bottlenecks in `iPrivJoin` are twofold: Firstly, in the first step, all data from the two parties must be communicated, particularly through OPPRF, incurring both communication and computational complexities of $O(hn(\lambda + \log n + m_b\kappa) + nm_a\ell)$. Here λ and κ are security parameters, ℓ is the element bit length, and h is the number of hash functions. Secondly, in the second step, two rounds of $\Pi_{\text{O-Shuffle}}$ on encoded data are required due to the secret-shared nature of the encoded data, incurring both communication and computation complexities of $2n(m_a\ell + m_b\ell + \kappa)$.

In contrast, `Bifrost` builds upon two conceptually simple building blocks, an ECDH-PSI protocol [28, 43] and the oblivious shuffle protocol [39], eliminating Cuckoo hashing and OPPRF. Notably, ECDH-PSI overhead $O(n\sigma)$ (where σ is the ECC key bit-length), depends only on identifiers and not on feature dimensions. We further propose a simple optimization named *dual mapping* that reduces the need for two rounds of oblivious shuffle (as in `iPrivJoin`) to one round. As a result, the total communication and computation complexity of `Bifrost` are both $O(n\sigma + nm_a\ell + nm_b\ell)$, reducing overhead by at least 66.7% compared to `iPrivJoin`. As shown in Table 1, `Bifrost` exhibits substantially lower online communication overhead than `iPrivJoin`. Moreover, `Bifrost` reduces offline communication at least by half compared to `iPrivJoin`.

Table 1: Comparison of `Bifrost` and `iPrivJoin` [39] in online communication. Here, n is the row count of the input data; m_a and m_b are the feature dimensions for P_a and P_b, respectively; $m = m_a + m_b$; ℓ is the element bit length; σ is the ECC key bit length; h is the number of hash functions; λ and κ are the statistical security parameter and the computational security parameter, respectively.

Protocol	Communication Size (bits)		Round
`iPrivJoin` [39]	Step (1)	Step (2)+(3)	11
	$O(hn(\lambda + \log n + m_b\kappa) + nm_a\ell)$	$O(nm\ell + n\kappa)$	
`Bifrost`	$O(n\sigma + nm\ell)$		4

2 PRELIMINARIES

Notations. We use the notation $\{x_1, \ldots, x_t\}$ to denote an unordered set and the notation $[x_1, \ldots, x_t]$ to denote an ordered list.

ECDH-PSI. Private Set Intersection (PSI) protocols constructed using Elliptic Curve Cryptography (ECC) [32, 44] are commonly referred to as ECDH-PSI [28, 43]. ECC is typically preferred in PSI because it offers the same security level with significantly smaller key sizes compared to other cryptographic algorithms with similar functionality, such as RSA. Technically, ECC relies on the algebraic properties of elliptic curves over finite fields. An elliptic curve E over $\mathbb{Z}_q$ is defined by the equation $y^2 = x^3 + \gamma_1 x + \gamma_2 \pmod q$, where $\gamma_1, \gamma_2 \in \mathbb{F}_q$ satisfy the non-singularity condition $4\gamma_1^3 + 27\gamma_2^2 \not\equiv 0 \pmod q$. Scalar multiplication on this curve, denoted as kQ, represents the repeated addition of a point Q to itself k times. The security of ECC is based on the hardness of the elliptic curve discrete logarithm problem (ECDLP) [23]: given two points Q and $Q_2 = kQ$, it is computationally infeasible to recover k.

In a typical ECDH-PSI protocol, two parties P_a and P_b hold data $X = [x_i]_{i\in[n]}$ and $Y = [y_i]_{i\in[n]}$, respectively. They first map their input data to points on an elliptic curve using a cryptographic hash function $H(\cdot)$. Then, they compute the intersection set as follows:

(1) P_a and P_b each generates a private scalar key, denoted α and β, respectively.
(2) P_b computes $\beta Y = [\beta H(y)]_{y\in Y}$ and sends βY to P_a.
(3) P_a shuffles its data to obtain $\pi(X)$ and computes $\alpha\pi(X) = [\alpha H(x)]_{x\in\pi(X)}$. P_a receives βY and computes $\alpha\beta Y = [\alpha y]_{y\in\beta Y}$. P_a sends $\alpha\pi(X)$ and $\alpha\beta Y$ to P_b.
(4) Upon receiving $\alpha\pi(X)$ and $\alpha\beta Y$, P_b computes $\beta\alpha\pi(X) = [\beta x]_{x\in\alpha\pi(X)}$. Since scalar multiplication on elliptic curves is commutative (i.e., $\alpha\beta = \beta\alpha$), P_b finally obtains the intersection set by evaluating $\beta\alpha\pi(X) \cap \alpha\beta Y$. P_b can reveal their intersection elements to P_a if necessary.

Additive Secret Sharing. Additive secret sharing is a key technology in SMPC [19, 66]. In the two-party setting of additive secret sharing, an ℓ-bit value x is split into two secret-shared values (i.e., shares), denoted $\langle x\rangle_a$ and $\langle x\rangle_b$, where $\langle x\rangle_* \in \mathbb{Z}_{2^\ell}$ and each party P_* holds one share $\langle x\rangle_*$ for $* \in \{a, b\}$. The original value x can be revealed by summing all shares: $\langle x\rangle_a + \langle x\rangle_b \equiv x \pmod{2^\ell}$.

Oblivious Shuffle. The oblivious shuffle functionality $\mathcal{F}_{\text{O-Shuffle}}$ inputs a random permutation π from P_b and inputs a matrix X from P_a. It outputs a secret-shared and shuffled matrix $\langle\pi(X)\rangle$ while ensuring that the permutation π is unknown to P_a and X is unknown to P_b. In this paper, we employ the oblivious shuffle

protocol $\Pi_{\text{O-Shuffle}}$ (Protocol 1). In the online phase of Protocol 1, the communication size and round are $n \cdot m$ and 1, respectively, where $n \cdot m$ is the size of matrix X. Additionally, for the offline phase of $\Pi_{\text{O-Shuffle}}$, we employ the protocol in Algorithm 2 in [39].

Protocol 1: $\Pi_{\text{O-Shuffle}}$

Parameters: The size of matrix $n \times m$; The bit length of element ℓ.
Inputs: P_a holds a matrix $X \in \mathbb{Z}_{2^\ell}^{n \times m}$.
Outputs: Each party obtains $\langle \pi(X) \rangle$, where π is only known to P_b.
Offline: P_a generates a random matrix $R \in \mathbb{Z}_{2^\ell}^{n \times m}$. P_b samples a random permutation $\pi : [n] \mapsto [n]$. Then, two parties invoke an instance of $\mathcal{F}_{\text{O-Shuffle}}$ to obtain $\langle \pi(R) \rangle$.
Online:
1. P_a computes $\hat{X} = X - R$ and sends $\hat{X}$ to P_b.
2. P_a sets $\langle \pi(X) \rangle_a = \langle \pi(R) \rangle_a$.
3. P_b computes $\langle \pi(X) \rangle_b = \pi(X) + \langle \pi(R) \rangle_b$.

3 PROBLEM DESCRIPTION

We summarize notations we use in Table 2.

Table 2: Notations used in this paper.

Symbol	Description
P_a,P_b	Two parties involved in secure data join.
n	The row count of the input data.
m_a,m_b	The feature dimension of P_a and P_b, respectively.
m	The total feature dimension ($m = m_a + m_b$).
c	The row count of the joined table.
ID^a, ID^b	The identifers (IDs) of P_a and P_b, respectively.
F^a, F^b	The features of P_a and P_b, respectively.
$\mathcal{D}$	The redundancy-free joined table.
$\lvert ID^a \cap ID^b \rvert$	The size of the intersection identifiers.
$\mathbb{Z}_{2^\ell}$	The ring size used in our paper.
$\mathbb{Z}_{2^\ell}^{c \times m}$	All $c \times m$ matrices whose elements are from the ring $\mathbb{Z}_{2^\ell}$.
$\langle x \rangle_*$	The share (i.e., secret-shared value) of x for P_*.
X_i	The i-th element of X.
$X_{i,j}$	The j-th element of i-th row in X.
$\{x_1, \ldots, x_t\}$	An unordered set.
$[x_1, \ldots, x_t]$	An ordered list.
$[x]$	An ordered list $[1, 2, ..., x]$.

Problem Definition. To formally define the problem, we present the functionality $\mathcal{F}_{\text{2PC-DJoin}}$ of the secure two-party data join. Note that $\mathcal{F}_{\text{2PC-DJoin}}$ specifies how to realize a secure data join with a third trusted party (TTP), while our proposed `Bifrost` realizes it without such a TTP. As shown in Figure 3, $\mathcal{F}_{\text{2PC-DJoin}}$ inputs (ID^a, F^a) from P_a and inputs (ID^b, F^b) from P_b. Here, ID^* is an ordered list of n identifiers (IDs) and F^* is the corresponding features of size $n \times m_*$ for $* \in \{a, b\}$. $\mathcal{F}_{\text{2PC-DJoin}}$ outputs shares $\langle \mathcal{D} \rangle_a$ and $\langle \mathcal{D} \rangle_b$ of the joined table $\mathcal{D}$ to P_a and P_b, respectively. Specifically, the joined table $\mathcal{D}$ contains $c = \lvert ID^a \cap ID^b \rvert$ rows, where each row consists of m_a features from P_a and m_b features from P_b for an intersection ID. The joined table shares satisfy $\langle \mathcal{D} \rangle_a + \langle \mathcal{D} \rangle_b = \mathcal{D}$ and are sampled uniformly at random. Since each party only obtains a joined table share that is independent of $\mathcal{D}$, neither party learns the intersection IDs nor the feature values in $\mathcal{D}$. Consequently, after executing $\mathcal{F}_{\text{2PC-DJoin}}$, neither party learns any extra information beyond what is revealed by the size $c \times m$ of the joined table.

Functionality $\mathcal{F}_{\text{2PC-DJoin}}$

Parameters: The input data row count n. The bit length of element ℓ. P_a's and P_b's feature dimensions m_a and m_b. $m = m_a + m_b$.
Input: P_a inputs its data (ID^a, F^a). P_b inputs its data (ID^b, F^b).
Functionality:
1. Let intersection identifiers $ID^{a \cap b} = ID^a \cap ID^b$. Define the joined table row count as $c = \lvert ID^{a \cap b} \rvert$.
2. Let joined table $\mathcal{D} = \{[F^a_{j_a,1}, \ldots, F^a_{j_a,m_a}, F^b_{j_b,1}, \ldots, F^b_{j_b,m_b}]\}_{j \in [c]}$, where j_a and j_b are indices such that $ID^a_{j_a} = ID^b_{j_b} = ID^{a \cap b}_j$.
3. Sample two random tables $\langle \mathcal{D} \rangle_a \in \mathbb{Z}_{2^\ell}^{c \times m}$ and $\langle \mathcal{D} \rangle_b \in \mathbb{Z}_{2^\ell}^{c \times m}$ such that $\langle \mathcal{D} \rangle_a + \langle \mathcal{D} \rangle_b = \mathcal{D}$.
4. Return $\langle \mathcal{D} \rangle_a$ to P_a and $\langle \mathcal{D} \rangle_b$ to P_b.

Figure 3: Ideal functionality of Secure Two-Party Data Join.

Security Model. Following prior secure data join schemes [39, 56], in this paper, we consider semi-honest probabilistic polynomial time (PPT) adversaries. A semi-honest PPT adversary $\mathcal{A}$ can corrupt one of the parties P_a or P_b at the beginning of the protocol and aims to learn extra information from the protocol execution while still correctly executing the protocol. We follow the standard simulation-based definition of security for secure two-party computation [9, 46]. We give the formal security definition as follows:

Definition 1 (Semi-honest Model). *Let $view_C^\Pi(x, y)$ be the views (including input, random tape, and all received messages) of the corrupted party C during the execution of the protocol Π, x be the input of the corrupted party and y be the input of the honest party. Let $out(x, y)$ be the protocol's output of all parties and $\mathcal{F}(x, y)$ be the functionality's output. Π is said to securely compute a functionality $\mathcal{F}$ in the semi-honest model if for any Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$ there exists a simulator Sim_C such that for all inputs x and y,*

$$\{view_C^\Pi(x, y), out(x, y)\} \approx_c \{Sim_C(x, \mathcal{F}_C(x, y)), \mathcal{F}(x, y)\}.$$

where $\approx_c$ represents computational indistinguishability.

4 DESIGN

`Bifrost` is instantiated by Protocol 2, which realizes the secure two-party data join functionality $\mathcal{F}_{\text{2PC-DJoin}}$. For simplicity, we assume both parties hold input data with equal row counts (n). `Bifrost` can be extended to support parties holding data with different row counts without extra overhead, as detailed in the full version [12].

Setup, Offline and Online Phase. Similar to prior works [24, 30], operations in `Bifrost` are performed in three distinct phases: (1) offline phase: the two parties perform input-independent precomputations to generate correlated randomness; (2) setup phase: the two parties preprocess their input data, respectively; (3) online phase: the two parties use randomness and the preprocessed data to securely compute the output.

Example: Throughout this section we use a simple example where P_a holds $(ID^a, F^a) = [(aaa, 12), (bbb, 61), (ccc, 37), (ddd, 49)]$, and P_b holds $(ID^b, F^b) = [(ddd, 13), (bbb, 51), (ggg, 28), (eee, 36)]$. In the offline phase, P_a samples two permutations $\pi_1^a : (1 \to 4, 2 \to 2, 3 \to 1, 4 \to 3)$ and $\pi_2^a : (1 \to 3, 2 \to 1, 3 \to 4, 4 \to 2)$, and P_b samples two permutations $\pi_1^b : (1 \to 4, 2 \to 1, 3 \to 3, 4 \to 2)$ and $\pi_2^b : (1 \to 1, 2 \to 3, 3 \to 4, 4 \to 2)$.

Protocol 2: $\Pi_{\text{2PC-DJoin}}$

Parameters: The same as $\mathcal{F}_{2PC\text{-}DJoin}$.
Input: P_a inputs its data (ID^a, F^a). P_b inputs its data (ID^b, F^b).
Output: P_a and P_b obtain the joined table shares $\langle \mathcal{D} \rangle_a$ and $\langle \mathcal{D} \rangle_b$, respectively.

1. **[Secure Mapped Intersection Generation]**. P_a and P_b execute the secure mapped intersection generation protocol Π_{SMIG} (Protocol 3), where P_a inputs its identifiers ID^a, and P_b inputs its identifiers ID^b. After the execution, both parties obtain the mapped intersection index pairs MIPairs.
2. **[Secure Feature Alignment]**. P_a and P_b execute the mapped intersection-based secure feature alignment protocol $\Pi_{\text{MI-SFA}}$ (Protocol 4), where P_a inputs its feature columns F^a and mapped intersection pairs MIPairs, and P_b inputs its feature columns F^b and mapped intersection pairs MIPairs. After the execution, P_a and P_b obtain the joined table shares $\langle \mathcal{D} \rangle_a$ and $\langle \mathcal{D} \rangle_b$, respectively.

4.1 Secure Mapped Intersection Generation

In the secure mapped intersection generation step, the two parties execute our proposed secure mapped intersection generation (SMIG) protocol Π_{SMIG} (Protocol 3). Below, we introduce the definition and construction of this protocol.

4.1.1 Definition. As shown in Figure 4, we present a secure mapped intersection generation functionality $\mathcal{F}_{SMIG}$. $\mathcal{F}_{SMIG}$ takes as input identifiers ID^a from P_a and identifiers ID^b from P_b. It outputs to both parties only the mapped intersection index pairs, denoted by MIPairs. Specifically, MIPairs $= [\pi_1(i), \pi_2(j) \mid (i,j) \in \text{Pairs}_\uparrow]$, where $\text{Pairs}_\uparrow$ consists of all intersection index pairs (i,j) with $ID^a_i = ID^b_j$, ordered by i, and π_1, π_2 are permutation (shuffle) functions unknown to both parties. After the execution of $\mathcal{F}_{SMIG}$, neither party should learn any extra information beyond what is revealed by the row count c of the joined table.

Functionality $\mathcal{F}_{SMIG}$

Parameters: The input data row count n.
Input: P_a inputs its identifiers ID^a. P_b inputs its identifiers ID^b.
Functionality:

1. P_a samples two permutations $\pi^a_1, \pi^a_2 : [n] \mapsto [n]$. P_b samples two permutations $\pi^b_1, \pi^b_2 : [n] \mapsto [n]$. Let $\pi_1 = \pi^a_1 \circ \pi^b_1$ and $\pi_2 = \pi^b_2 \circ \pi^a_2$.
2. Let $\text{Pairs} = [(i,j) \mid ID^a_i == ID^b_j]$ and let $\text{Pairs}_\uparrow$ denote Pairs sorted in ascending order by the first element i of each pair.
3. Define the mapped intersection index pairs MIPairs $= [\pi_1(i), \pi_2(j)]_{(i,j)\in\text{Pairs}_\uparrow}$.
4. Return MIPairs to both P_a and P_b.

Figure 4: Ideal functionality of SMIG.

4.1.2 Construction. As shown in Protocol 3, the secure mapped intersection generation protocol Π_{SMIG} consists of an offline phase, a setup phase, and an online phase. The online phase of Π_{SMIG} consists of an ECC-based ID encrypting step and a mapped intersection generation step. Below, we introduce these steps in detail.

Offline Phase. Each party P_* $(* \in \{a,b\})$ samples two distinct random permutations $\pi^*_1, \pi^*_2 \in [n] \mapsto [n]$. Both parties keep their permutations for future use.

Setup Phase. Each party first locally shuffles its identifiers (IDs) and then encrypts the shuffled IDs using the ECC key. Specifically, (a) For P_a, it first shuffles its IDs with π^a_1 to obtain the shuffled IDs $\pi^a_1(ID^a)$. Next, P_a generates an ECC key α and uses α to encrypt the shuffled IDs, obtaining $\alpha H(\pi^a_1(ID^a))$, where $H(\cdot)$ maps each ID to an elliptic-curve point. (b) For P_b, it first shuffles its IDs with π^b_2 to obtain the shuffled IDs $\pi^b_2(ID^b)$. Next, P_b generates an ECC key β and uses β to encrypt the shuffled IDs, obtaining $\beta H(\pi^b_2(ID^b))$.

Step 1: ECC-based ID Encrypting. To enable P_a to later obtain the mapped intersection index pairs, the parties engage in an exchange of encrypted IDs using ECC, as follows: (1) P_a sends the encrypted and shuffled IDs $\alpha H(\pi^a_1(ID^a))$ to P_b. (2) P_b first shuffles the received data $\alpha H(\pi^a_1(ID^a))$ with π^b_1 to obtain dual-shuffled IDs $\alpha H(\pi_1(ID^a))$, where the composition permutation $\pi_1 = \pi^a_1 \circ \pi^b_1$. Next, using its private key β, P_b encrypts the dual-shuffled IDs to obtain the dual-encrypted and dual-shuffled IDs $\beta\alpha H(\pi_1(ID^a)) = [\beta(x)]_{x\in\alpha H(\pi_1(ID^a))}$. Finally, P_b sends $\beta\alpha H(\pi_1(ID^a))$ and its self-shuffled and encrypted IDs $\beta H(\pi^b_2(ID^b))$ to P_a. (3) P_a decrypts the received data $\beta\alpha H(\pi_1(ID^a))$ using its private key α, obtaining $\beta H(\pi_1(ID^a)) = [\alpha^{-1}(x)]_{x\in\beta\alpha H(\pi_1(ID^a))}$. Based on the hardness of the ECDLP, it is computationally infeasible for P_a to recover $H(\pi_1(ID^a))$ from $\beta H(\pi_1(ID^a))$ or recover $H(\pi^b_2(ID^b))$ from $\beta H(\pi^b_2(ID^b))$. At this point, P_a holds $\beta H(\pi_1(ID^a))$ and $\beta H(\pi^b_2(ID^b))$, both encrypted under P_b's private key β.

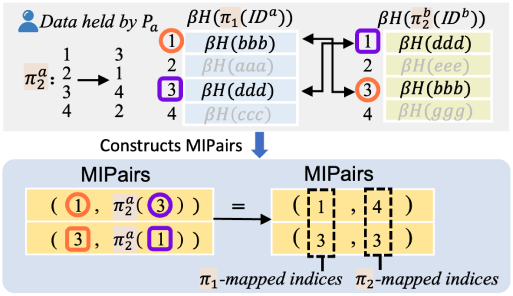


Figure 5: An illustration of constructing MIPairs by P_a in Step 2-(1) of Π_{SMIG} (Protocol 3), using the example data described at the beginning of Section 4.

Step 2: Mapped Intersection Generation. In this step, P_a first obtains the mapped intersection index pairs MIPairs from encrypted data $\beta H(\pi_1(ID^a))$ and $\beta H(\pi^b_2(ID^b))$, both of which are encrypted under P_b's private ECC key β. We provide a detailed explanation of P_a's procedure for constructing MIPairs in the next paragraph. Secondly, P_a outputs MIPairs and sends MIPairs to P_b.

P_a's procedure for constructing MIPairs. As shown in Figure 5, the core idea of this procedure relies on the observation that both $\beta H(\pi_1(ID^a))$ and $\beta H(\pi^b_2(ID^b))$ contain the same encrypted value $\beta H(id)$ for every intersection identifier id. Moreover, $\beta H(\pi_1(ID^a))$ is securely shuffled under the composition permutation π_1, i.e., $\pi^a_1 \circ \pi^b_1$, while $\beta H(\pi^b_2(ID^b))$ is shuffled under P_b's permutation π^b_2. To generate the mapped intersection index pairs MIPairs, P_a proceeds as follows. For each $i \in [n]$ in $\beta H(\pi_1(ID^a))$, if P_a can find some $j \in [n]$ such that $\beta H(\pi_1(ID^a))_i == \beta H(\pi^b_2(ID^b))_j$,

- Index i is exactly the π_1-mapped index of id in ID^a, since encrypted IDs $\beta H(\pi_1(ID^a))$ is π_1-shuffled.

Protocol 3: Π_{SMIG}

Parameters: The input data row count n. The ECC key bit length σ. A hash function $H(\cdot)$ used for hashing an element to a point on the elliptic curve.
Input: P_a inputs identifiers ID^a. P_b inputs identifiers ID^b.
Output: Both P_a and P_b obtain the mapped intersection index pairs MIPairs.
Offline: P_a samples two random permutations $\pi_1^a, \pi_2^a \in [n] \mapsto [n]$. P_b samples two random permutations $\pi_1^b, \pi_2^b \in [n] \mapsto [n]$.
Setup:
- **[Shuffling and Encrypting Inputs of P_a]**. P_a shuffles ID^a with π_1^a to obtain $\pi_1^a(ID^a)$. P_a generates an ECC private key α and computes $\alpha H(\pi_1^a(ID^a)) = [\alpha H(id)]_{id \in \pi_1^a(ID^a)}$.
- **[Shuffling and Encrypting Inputs of P_b]**. P_b shuffles ID^b with π_2^b to obtain $\pi_2^b(ID^b)$. P_b generates an ECC private key β and computes $\beta H(\pi_2^b(ID^b)) = [\beta H(id)]_{id \in \pi_2^b(ID^b)}$.

Online:

1. **[ECC-based ID Encrypting]**.
   (1) P_a sends $\alpha H(\pi_1^a(ID^a))$ to P_b.
   (2) P_b shuffles $\alpha H(\pi_1^a(ID^a))$ with π_1^b to obtain dual-shuffled IDs $\alpha H(\pi_1(ID^a))$, where the composition permutation $\pi_1 = \pi_1^a \circ \pi_1^b$. Then, using its private key β, P_b encrypts the dual-shuffled IDs to obtain the dual-encrypted IDs $\beta\alpha H(\pi_1(ID^a)) = [\beta(x)]_{x \in \alpha H(\pi_1(ID^a))}$. P_b sends this data $\beta\alpha H(\pi_1(ID^a))$ and its encrypted and shuffled IDs $\beta H(\pi_2^b(ID^b))$ to P_a.
   (3) P_a decrypts the received data $\beta\alpha H(\pi_1(ID^a))$ using its private key α, obtaining $\beta H(\pi_1(ID^a)) = [\alpha^{-1}(x)]_{x \in \beta\alpha H(\pi_1(ID^a))}$.
2. **[Mapped Intersection Generation]**.
   (1) For $i \in [n]$, if P_a can find some $j \in [n]$ such that $\beta H(\pi_1(ID^a))_i == \beta H(\pi_2^b(ID^b))_j$ (implying both are encrypted from the same intersection identifier id):
      - P_a maps j to $\pi_2^a(j)$. Let i' denote the index of id in ID^b, so that $j = \pi_2^b(i')$ and $\pi_2^a(j) = \pi_2^a(\pi_2^b(i')) = \pi_2(i')$.
      - P_a adds the pair $\left(i, \pi_2^a(j)\right)$ to MIPairs, where the first and second entries are the π_1-mapped index of id in ID^a and the π_2-mapped index of id in ID^b, respectively.
   (2) P_a outputs MIPairs (also sends MIPairs to P_b).

- P_a maps j to $\pi_2^a(j)$. Let i' denote the index of id in ID^b, so that $j = \pi_2^b(i')$ and $\pi_2^a(j) = \pi_2^a(\pi_2^b(i')) = \pi_2(i')$, which is the π_2-mapped index of id in ID^b.
- P_a then appends the pair $\left(i, \pi_2^a(j)\right)$ to MIPairs, where the first entry is the π_1-mapped index of id in ID^a and the second entry is the π_2-mapped index of id in ID^b.

Note that P_a learns no information about the actual values of the intersection IDs, since P_a only obtains encrypted and shuffled IDs.
Example: As shown in Figure 5, after step 1, P_a obtains $\beta H(\pi_1(ID^a))$ and $\beta H(\pi_2^b(ID^b))$. Then, in step 2-(1), P_a constructs the mapped intersection index pairs MIPairs $= [(1,4),(3,3)]$, which are then sent to P_b in step 2-(2).

4.2 Secure Feature Alignment

In the secure feature alignment step, based on the outputs of Π_{SMIG} (Protocol 3), P_a and P_b execute our proposed mapped intersection-based secure feature alignment protocol $\Pi_{\text{MI-SFA}}$ to securely obtain the joined table shares $\langle \mathcal{D} \rangle$.

As shown in Protocol 4, $\Pi_{\text{MI-SFA}}$ takes as input feature columns F^a from P_a and feature columns F^b from P_b. In addition, it takes as input the mapped intersection index pairs MIPairs from both parties. After execution, $\Pi_{\text{MI-SFA}}$ outputs to both parties joined table shares $\langle D \rangle$. $\Pi_{\text{MI-SFA}}$ consists of an offline phase and an online phase. In the offline phase, P_a and P_b generates random matrices required for the online execution of $\Pi_{\text{O-Shuffle}}$ (Protocol 1). The online phase consists of three steps. In steps 1 and 2, P_a and P_b securely dual-shuffle features from P_a and P_b, respectively. For dual-shuffling, each party performs the first-layer shuffle by locally using its permutation and performs the second-layer shuffle by executing $\Pi_{\text{O-Shuffle}}$ (Protocol 1). In step 3, both parties locally extract the shares of the joined table according to the mapped intersection index pairs MIPairs. Below, we introduce these steps in detail.

Offline Phase. Each party P_* ($* \in \{a, b\}$) generates a random matrix $R^* \in \mathbb{Z}_{2^\ell}^{n \times m_*}$. Using the permutations π_2^b and π_2^a generated in the offline phase of Π_{SMIG} (Protocol 3) by P_b and P_a, respectively, both parties involke the oblivious shuffle $\mathcal{F}_{\text{O-Shuffle}}$ to obtain the shares of shuffled random matrix, specifically $\langle \pi_2^b(R^a) \rangle$, $\langle \pi_2^a(R^b) \rangle$.

Step 1: Dual-Shuffling ($\pi_1 = \pi_1^a \circ \pi_1^b$) Features from P_a. Both parties securely obtain shares of P_a's dual-shuffled features simply as follows: (1) P_a locally shuffles its features F^a with its permutation π_1^a to obtain $\pi_1^a(F^a)$. (2) P_a and P_b jointly execute the online phase of $\Pi_{\text{O-Shuffle}}$ (Protocol 1), where P_a inputs shuffled features $\pi_1^a(F^a)$, random matrix R^a, shuffled random matrix share $\langle \pi_1^b(R^a) \rangle_a$ and receives dual-shuffled feature share $\langle \pi_1^b(\pi_1^a(F^a)) \rangle_a$, i.e., $\langle \pi_1(F^a) \rangle_a$, while P_b inputs its permutation π_1^b (generated in the offline phase of Protocol 3), shuffled random matrix share $\langle \pi_1^b(R^a) \rangle_b$ and receives dual-shuffled feature share $\langle \pi_1(F^a) \rangle_b$. Note that the only communication in the online phase of $\Pi_{\text{O-Shuffle}}$ (Protocol 1) is P_a sending its features masked with random matrix R^a to P_b.

Step 2: Dual-Shuffling ($\pi_2 = \pi_2^b \circ \pi_2^a$) Features from P_b. In a symmetrical step analogous to the first step, the parties now dual-shuffle the features F^b from P_b, with the roles of P_a and P_b exchanged. This process yields dual-shuffled shares $\langle \pi_2(F^b) \rangle_a$ and $\langle \pi_2(F^b) \rangle_b$ to P_a and P_b, respectively.

Step 3: Locally Extract Intersection Data by the Mapped Intersection. This step is straightforward. Each party holds the dual-shuffled feature shares $\langle \pi_1(F^a) \rangle$, $\langle \pi_2(F^b) \rangle$ and mapped intersection index pairs MIPairs, where each pair are the π_1-mapped index of intersection identifier id in ID^a and the π_2-mapped index of intersection identifier id in ID^b. Thus, each party locally extracts shares from $\langle \pi_1(F^a) \rangle$ whose indices are the first entry in each pair of MIPairs and extracts shares from $\langle \pi_2(F^b) \rangle$ whose indices are the second entry in each pair of MIPairs, forming its joined table share. That is, each party collects $\langle \mathcal{D} \rangle = \{\langle \pi_1(F^a)_i \rangle, \langle \pi_2(F^b)_j \rangle\}_{(i,j) \in \text{MIPairs}}$.
Example: Recall that the mapped intersection index pairs MIPairs $= [(1,4),(3,3)]$ are obtained in Protocol 3. After steps 1 and 2, P_a obtains dual-shuffled feature shares $\langle \pi_1(F^a) \rangle_a = \langle [61, 12, 49, 37] \rangle_a = [43, 8, 23, 9]$ and $\langle \pi_2(F^b) \rangle_a = \langle [36, 28, 13, 51] \rangle_a = [25, 15, 4, 16]$, while P_b obtains dual-shuffled feature shares $\langle \pi_1(F^a) \rangle_b = \langle [61, 12, 49, 37] \rangle_b = [18, 4, 26, 28]$ and $\langle \pi_2(F^b) \rangle_b = \langle [36, 28, 13, 51] \rangle_b = [11, 13, 9, 35]$. In step 3, using MIPairs, each party locally collects $\langle \mathcal{D} \rangle = \{\langle \pi_1(F^a)_i \rangle, \langle \pi_2(F^b)_j \rangle\}_{(i,j) \in \text{MIPairs}} = \langle [(61, 51), (49, 13)] \rangle$. That is, P_a obtains the joined table share $\langle \mathcal{D} \rangle_a = [(43, 16), (23, 4)]$, while P_b obtains the joined table share $\langle \mathcal{D} \rangle_b = [(18, 35), (26, 9)]$.

Protocol 4: $\Pi_{\text{MI-SFA}}$

Parameters: The input data row count n. P_a's feature dimension m_a. P_b's feature dimension m_b.

Input: P_a inputs feature columns F^a and the mapped intersection index pairs MIPairs. P_b inputs feature columns F^b and the mapped intersection index pairs MIPairs.

Output: P_a and P_b obtain the joined table shares $\langle \mathcal{D} \rangle_a$ and $\langle \mathcal{D} \rangle_b$, respectively, where $\mathcal{D}$ consists of the matched rows.

Offline:

- **[Shuffling Random Matrix of P_a].** P_a samples random matrix $R^a \in \mathbb{Z}_{2^\ell}^{n \times m_a}$. P_a and P_b invoke $\mathcal{F}_{\text{O-Shuffle}}$, where P_a inputs R^a and receives $\langle \pi_1^b(R^a) \rangle_a$, while P_b inputs permutation π_1^b, generated in the offline phase of Π_{SMIG} (Protocol 3), and receives $\langle \pi_1^b(R^a) \rangle_b$.
- **[Shuffling Random Matrix of P_b].** P_b samples random matrix $R^b \in \mathbb{Z}_{2^\ell}^{n \times m_b}$. P_a and P_b invoke $\mathcal{F}_{\text{O-Shuffle}}$, where P_a inputs permutation π_2^a, generated in the offline phase of Π_{SMIG} (Protocol 3), and receives $\langle \pi_2^a(R^b) \rangle_a$, while P_b inputs R^b and receives $\langle \pi_2^a(R^b) \rangle_b$.

Online:

1. **Dual-Shuffling ($\pi_1 = \pi_1^a \circ \pi_1^b$) Features from P_a:**
   (1) P_a shuffles its features F^a with π_1^a to obtain $\pi_1^a(F^a)$.
   (2) P_a and P_b execute online phase of $\Pi_{\text{O-Shuffle}}$ (Protocol 1), where P_a inputs $\pi_1^a(F^a)$, random matrix R^a, shuffled random matrix share $\langle \pi_1^b(R^a) \rangle_a$ and receives dual-shuffled feature share $\langle \pi_1(F^a) \rangle_a$, while P_b inputs permutation π_1^b, generated in the offline phase of Π_{SMIG} (Protocol 3), shuffled random matrix share $\langle \pi_1^b(R^a) \rangle_b$ and receives dual-shuffled feature share $\langle \pi_1(F^a) \rangle_b$.
2. **Dual-Shuffling ($\pi_2 = \pi_2^b \circ \pi_2^a$) Features from P_b:**
   (1) P_b shuffles its features F^b with π_2^b to obtain $\pi_2^b(F^b)$.
   (2) P_a and P_b execute online phase of $\Pi_{\text{O-Shuffle}}$ (Protocol 1), where P_b inputs $\pi_2^b(F^b)$, random matrix R^b, shuffled random matrix share $\langle \pi_2^a(R^b) \rangle_b$, and receives dual-shuffled feature share $\langle \pi_2(F^b) \rangle_b$, while P_a inputs permutation π_2^a, generated in the offline phase of Π_{SMIG} (Protocol 3), shuffled random matrix share $\langle \pi_2^a(R^b) \rangle_a$ and receives dual-shuffled feature share $\langle \pi_2(F^b) \rangle_a$.
3. **Locally Extract Intersection Data by the Mapped Intersection Indices**: Each party P_* ($* \in \{a, b\}$) extracts intersection data share from dual-shuffled feature shares according to MIPairs, yielding $\langle \mathcal{D} \rangle_* = \{\langle \pi_1(F^a)_i \rangle_* \| \langle \pi_2(F^b)_j \rangle_*\}_{(i,j) \in \text{MIPairs}}$.

5 SECURITY PROOF

Below, we prove the security of two protocols in `Bifrost`, namely Π_{SMIG} (Protocol 3) and $\Pi_{\text{MI-SFA}}$ (Protocol 4), against semi-honest adversaries in the two-party setting (the definition of the security model is shown in Section 2).

Theorem 1. *The SMIG protocol Π_{SMIG} (Protocol 3) securely realizes $\mathcal{F}_{SMIG}$ in Figure 4 against semi-honest adversary $\mathcal{A}$.*

Proof. We exhibit simulators Sim_a and Sim_b for simulating the view of the corrupt P_a and P_b, respectively, and prove that the simulated view is indistinguishable from the real one via standard hybrid arguments.

Case 1 (Corrupt P_a). The simulator Sim_a receives P_a's input ID^a and P_a's output MIPairs. Sim_a simulate the view of $\mathcal{A}$. The incoming messages to $\mathcal{A}$ is encrypted and shuffled IDs $\beta\alpha H(\pi_1(ID^a))$ and $\beta H(\pi_2^b(ID^b))$ received in Online step 1-(2). Sim_a generates the view for $\mathcal{A}$ as follows:

- Sim_a samples an ECC key β from $\mathbb{Z}_{2^\sigma}$, two random permutations $\pi_1^b, \pi_2^b : [n] \mapsto [n]$. In addition, Sim_a constructs an n-sized IDs ID^b by randomly sampling c IDs from P_a's IDs ID^a and the remaining $n - c$ IDs from the complement of ID^a, where c is the row count of the joined table, i.e., |MIPairs|. Moreover, Sim_a follows the real protocol to sample an ECC key α from $\mathbb{Z}_{2^\sigma}$, two random permutations $\pi_1^a, \pi_2^a : [n] \mapsto [n]$.
- Sim_a shuffles ID^a with $\pi_1 = \pi_1^a \circ \pi_1^b$ to obtain $\pi_1(ID^a)$. Sim_a computes $\beta\alpha H(\pi_1(ID^a)) = [\beta\alpha H(id)]_{id \in \pi_1(ID^a)}$.
- Sim_a shuffles ID^b with π_2^b to obtain $\pi_2^b(ID^b)$. Sim_a computes $\beta H(\pi_2^b(ID^b)) = [\beta H(id)]_{id \in \pi_2^b(ID^b)}$.
- Sim_a appends $\beta\alpha H(\pi_1(ID^a))$ and $\beta H(\pi_2^b(ID^b))$ to the view of $\mathcal{A}$.
- Sim_a follows the real protocol to construct output MIPairs from $\alpha H(\pi_1(ID^a))$ and $\alpha H(\pi_2^b(ID^b))$.
- Sim_a appends MIPairs to the view of $\mathcal{A}$.

We argue that the view output by Sim_a is indistinguishable from the real one. We first define three hybrid transcripts T_0, T_1, T_2, where T_0 is the real view of P_a, and T_2 is the output of Sim_a.

1. $Hybrid_0$. The first hybrid is the real interaction described in Protocol 3. Here, an honest party P_b uses real inputs and interacts with the corrupt party P_a. Let T_0 denote the real view of P_a.
2. $Hybrid_1$. Let T_1 be the same as T_0, except that $\beta\alpha H(\pi_1(ID^a))$ and $\beta H(\pi_2^b(ID^b))$ are replaced by the simulated values. By the security of ECC, the simulated $\beta\alpha H(\pi_1(ID^a))$ and $\beta H(\pi_2^b(ID^b))$ have the same distribution as they would in the real protocol. Hence, T_0 and T_1 are statistically indistinguishable.
3. $Hybrid_2$. Let T_2 be the same as T_1, except that MIPairs is replaced by the simulated values. For each pair in MIPairs, the first and second entries are exactly the π_1-mapped index of the intersection identifier id in ID^a and the π_2-mapped index of the same intersection identifier id in ID^b, In the real view, $\pi_1 = \pi_1^a \circ \pi_1^b$ and $\pi_2 = \pi_2^b \circ \pi_2^a$ are two random composition permutations, which are unknown to P_a, their specific mappings are essentially random. Thus MIPairs are uniformly random over $[n]$. The simulator generates random permutations π_1^b and π_2^b. Even though π_1^a and π_2^a are fixed, applying a uniformly random permutation π_1^b or π_2^b afterward means the final mapping π_1 and π_2 are uniformly random. Thus, the resulting distributions of MIPairs in both the real and simulated views are statistically identical. Therefore, the simulated view is indistinguishable from the real view. This hybrid is exactly the view output by the simulator.

Case 2 (Corrupt P_b). The simulator Sim_b receives P_b's input ID^b and P_b's output MIPairs. The incoming view of $\mathcal{A}$ consists of the encrypted and shuffled IDs $\alpha H(\pi_1^a(ID^a))$ received in Online step 1-(1) and the output MIPairs received in Online step 2-(2). Sim_b generates the view for $\mathcal{A}$ as follows:

- Sim_b samples an ECC key α from $\mathbb{Z}_{2^\sigma}$, two random permutations $\pi_1^a, \pi_2^a : [n] \mapsto [n]$. In addition, Sim_b constructs an n-sized IDs ID^a by randomly sampling c IDs from P_b's IDs ID^b and the remaining $n - c$ IDs from the complement of ID^b, where c is the row count of the joined table, i.e., |MIPairs|. Moreover, Sim_b follows the real protocol to sample an ECC key β from $\mathbb{Z}_{2^\sigma}$, two random permutations $\pi_1^b, \pi_2^b : [n] \mapsto [n]$.
- Sim_b shuffles ID^a with π_1^a to obtain $\pi_1^a(ID^a)$. Sim_b computes $\alpha H(\pi_1^a(ID^a)) = [\alpha H(id)]_{id \in \pi_1^a(ID^a)}$.
- Sim_b appends $\alpha H(\pi_1^a(ID^a))$ to the view of $\mathcal{A}$.

- $\mathsf{Sim_b}$ computes $\beta H(\pi_1(ID^a))$ and $\beta H(\pi_2^b(ID^b))$. Then, $\mathsf{Sim_b}$ involve the procedure for constructing MIPairs.
- $\mathsf{Sim_b}$ appends MIPairs to the view of $\mathcal{A}$.

We argue that the view output by $\mathsf{Sim_b}$ is indistinguishable from the real one. We first define three hybrid transcripts T_0,T_1, T_2, where T_0 is the real view of P_b, and T_2 is the output of $\mathsf{Sim_b}$.

1. $Hybrid_0$. The first hybrid is the real interaction described in Protocol 3. Here, an honest party P_a uses real inputs and interacts with the corrupt party P_b. Let T_0 denote the real view of P_b.
2. $Hybrid_1$. Let T_1 be the same as T_0, except that $\alpha H(\pi_1^a(ID^a))$ is replaced by the simulated values. By the security of ECC, the simulated $\alpha H(\pi_1^a(ID^a))$ has the same distribution as it would in the real protocol.
3. $Hybrid_2$. Let T_2 be the same as T_1, except that MIPairs is replaced by the simulated values. Similar to Case 1, in the real view, $\pi_1 = \pi_1^a \circ \pi_1^b$ and $\pi_2 = \pi_2^b \circ \pi_2^a$ are two random permutations, which are unknown to P_a, their specific mappings are essentially random. Thus, MIPairs are uniformly random over $[n]$. The simulator generates random permutations to replace them. Thus, the resulting distributions of MIPairs in both the real and simulated views are statistically identical. This hybrid is exactly the view output by the simulator. □

Theorem 2. *The MI-SFA protocol $\Pi_{\text{MI-SFA}}$ (Protocol 4) securely realizes $\mathcal{F}_{\text{MI-SFA}}$ against semi-honest adversary $\mathcal{A}$.*

Proof Sketch. We argue the security of $\Pi_{\text{MI-SFA}}$ by reducing it to the security of its sub-protocol, $\Pi_{\text{O-Shuffle}}$ (Protocol 1). By design, $\Pi_{\text{MI-SFA}}$ involves interaction only during execution of $\Pi_{\text{O-Shuffle}}$. All other steps are performed locally and hence reveal no additional information to the other party. Intuitively, $\Pi_{\text{O-Shuffle}}$ is secure because its only communication step involves sending masked input data $\hat{X} = X - R$, where R is a random matrix with entries sampled uniformly from $\mathbb{Z}_{2^\ell}$. This ensures that $\hat{X}$ is uniformly distributed and independent of X, making any two inputs statistically indistinguishable from their masked versions. Consequently, a simulator can output a uniformly random matrix with distribution identical to the real execution. As the security of $\Pi_{\text{O-Shuffle}}$ is formally proven in [39], the security of Protocol 4 follows. □

6 EVALUATION

Our evaluation aims to answer the following questions:

- Is it beneficial to use the redundancy-free secure data join protocol, `Bifrost`, instead of `CPSI` [50], in the two-step SDA pipeline (secure data join and secure analytics) (Section 6.2)?
- How does `Bifrost` perform relative to the SOTA redundancy-free secure data join protocol, `iPrivJoin` [39] (Section 6.3)?

6.1 Implementation and Experiment Settings

Implementation. We implement `Bifrost` using C++ based on the ECC library libsodium[2]. We represent all input data in fixed-point format over the ring $\mathbb{Z}_{2^{64}}$, as in the SDA frameworks [16, 45]. For ECC, we use Curve25519 [6], which provides 128-bit security with a 256-bit key length. The curve equation of Curve25519 is $y^2 = (x^3 + 486662x^2 + x) \mod (2^{255} - 19)$.

[2]https://github.com/jedisct1/libsodium

Experiment Environment. We perform all experiments on a single Linux server equipped with 2× 10-core 2.4 GHz Intel Xeon Silver 4210R and 1 TB RAM. Each party is simulated by a separate process with one thread. We apply the tc tool[3] to simulate a WAN setting with a bandwidth of 100 Mbps and 40ms round-trip time, following prior works [29, 39].

Baseline. We consider the following two baselines:

- `iPrivJoin` [39]: To the best of our knowledge, `iPrivJoin` is the only existing protocol that provides the same functionality, i.e., a redundancy-free secure two-party data join protocol, as `Bifrost`. Since `iPrivJoin` is not open-sourced, we re-implemented it, replacing its OPPRF libraries with faster ones [50, 51].
- `CPSI` [50]: `CPSI` [50] is the SOTA circuit-based PSI protocol, which can be used for secure data join. `CPSI` introduces many redundant dummy rows in the joined table (padded to the maximum length), other than the actual matched rows. We reproduce `CPSI`'s results using its open-source code[4].

For OPPRF, we set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$, consistent with the two baselines. For Cuckoo hashing, we refer to the setting of `CPSI` [50], with $\omega = 1.27$ and 3 hashing functions.

Table 3: Detailed information of datasets.

Dataset	Row Count (n)	Feature Dimension (m)
Education Career Success	5000	19
Breast Cancer Gene	1904	693
Skin Cancer	10000	785
a9a	32561	123
ARCENE	900	10000
MNIST	60000	784
Tiny ImageNet	100000	4096
CelebA	200000	5000

Datasets. Table 3 summarizes all real-world datasets used in our evaluation. To answer the first question (Section 6.2), we use four real-world datasets: Education Career Success [54], Breast Cancer Gene [2], Skin Cancer [61], and a9a [11], to evaluate on secure data join and SDA tasks. To answer the second question (Section 6.3), we first evaluate `Bifrost` on eight datasets: the above four plus ARCENE [22], MNIST [36], Tiny ImageNet [35], and the large-scale CelebA [40], following prior SDA studies [45, 59]. These datasets vary in row counts (from 900 to 200000) and feature dimensions (from 19 to 10000). For each dataset, we perform vertical partitioning, where the two parties hold disjoint feature columns but share the same row count as the original dataset. We set the intersection rate to $\rho = 80\%$, consistent with `iPrivJoin` [39]. Specifically, we randomly select 80% overlapping samples and fill the remaining 20% of non-overlapping rows with randomly generated samples. As the intersection rate (ρ) has a negligible impact on the performance of the baselines and `Bifrost`, setting $\rho = 80\%$ provides a fair comparison. In addition, we configure that P_a holds the data with a greater number of feature columns, as the baselines are more efficient in this setup compared to the reverse configuration (see Section 6.3.3 for details). Furthermore, we generate synthetic datasets with sizes up to 100 GB to evaluate efficiency under varying parameters.

[3]https://man7.org/linux/man-pages/man8/tc.8.html
[4]https://github.com/Visa-Research/volepsi.git

Table 4: End-to-end online comparison of Bifrost, iPrivJoin [39] and CPSI [50] on two types of SDA tasks: secure statistical analysis and secure training (the term "secure" is omitted in the table). The costs of the SDA tasks for Bifrost and iPrivJoin are identical, as both produce the same inputs for SDA tasks. The efficiency of secure training is evaluated in one epoch.

Steps	Education Career Sucess						Breast Cancer Gene					
	Time (s)			Communication (MB)			Time (s)			Communication (MB)		
	Bifrost	iPrivJoin [39]	CPSI [50]	Bifrost	iPrivJoin [39]	CPSI [50]	Bifrost	iPrivJoin [39]	CPSI [50]	Bifrost	iPrivJoin [39]	CPSI [50]
Secure Data Join	1.17	2.94	3.61	1.21	7.96	7.36	1.46	19.37	11.83	10.25	67.44	61.53
Statistical Analyze	3.04		8.50	0.50		1.86	11.19		21.50	103.22		186.63
Total	4.21	5.98	12.11	1.71	8.46	9.22	12.65	30.56	33.33	113.47	170.66	248.16
Steps	Skin Cancer						a9a					
	Time (s)			Communication (MB)			Time (s)			Communication (MB)		
	Bifrost	iPrivJoin [39]	CPSI [50]	Bifrost	iPrivJoin [39]	CPSI [50]	Bifrost	iPrivJoin [39]	CPSI [50]	Bifrost	iPrivJoin [39]	CPSI [50]
Secure Data Join	7.11	108.07	73.63	60.87	385.51	319.05	8.24	61.88	33.62	33.98	247.85	228.79
Training	2371.44		3735.75	683.14		1073.44	7181.97		11365.90	290.28		459.31
Total	2378.55	2479.51	3809.38	744.01	1068.65	1392.49	7190.21	7243.85	11399.52	324.26	538.13	688.10

6.2 Two-step SDA Pipeline Evaluation

We compare CPSI [50], iPrivJoin [39], and Bifrost in a two-step SDA pipeline: secure data join and SDA task execution. The SDA task leverages the outputs of secure data join as its input data.

SDA tasks. We evaluate two types of SDA tasks: secure statistical analysis and secure training. (1) For secure statistical analysis, we examine the following two scenarios:

- Chi-square test: an education department and a tax department securely compute a chi-square test value on education data and income data [17]. To simulate this scenario, the feature columns in the Education Career Success dataset [54] are vertically partitioned into education-related and income-related features, which are held separately by the two parties.
- Pearson correlation: a hospital and a genomic company securely compute the Pearson correlation between disease and gene data [38]. To simulate this scenario, the feature columns in the Breast Cancer Gene dataset [2] are vertically partitioned into disease-related features and gene-related features, which are held separately by the two parties.

We design and implement secure Chi-square test and secure Pearson correlation protocols using the SMPC primitives in MP-SPDZ [29]. (2) For secure training, we examine two scenarios: One party holds users' image data while the other holds their labels; two parties hold disjoint subsets of users' features. In both scenarios, the parties aim to securely train a model on the joined table shares. We use the Skin Cancer [61] and a9a [11] datasets to simulate the above two scenarios, respectively. We employ MP-SPDZ [29], a widely used open-source SMPC framework, to securely train logistic regression models over 10 epochs.

In terms of efficiency, Table 4 shows that Bifrost significantly accelerates the secure data join process, achieving up to 10.36× faster running time and up to 6.73× reduction in communication size compared to CPSI, and up to 15.20× faster running time and up to 7.29× reduction in communication size compared to iPrivJoin. The inferior performance of iPrivJoin relative to CPSI stems from its reliance on multiple rounds of oblivious shuffling to remove redundant rows present in CPSI. An important observation is that, during the downstream SDA process, CPSI incurs an increase of $1.58 \sim 2.80\times$ in running time and $1.57 \sim 3.72\times$ in communication size. The inefficiency of the SDA process caused by CPSI arises from the redundant padded outputs in the join results. Specifically, CPSI outputs a joined table of row count $N = \omega \cdot n$ (with $\omega = 1.27$ in our experiments), where $\rho \cdot n$ rows are intersection data (intersection rate ρ). For $\rho = 80\%$, the redundant fraction of CPSI's outputs is $\big((\omega-\rho)\cdot n\big)/N \approx 37.01\%$. In terms of correctness, Table 5 shows that CPSI leads to catastrophic error rate blowup in downstream secure statistical analysis and degrades the accuracy of secure training, with the Chi-square test statistic deviating by up to 13858.55%. Due to the limited performance and accuracy of CPSI, we exclude it from subsequent secure data join evaluations.

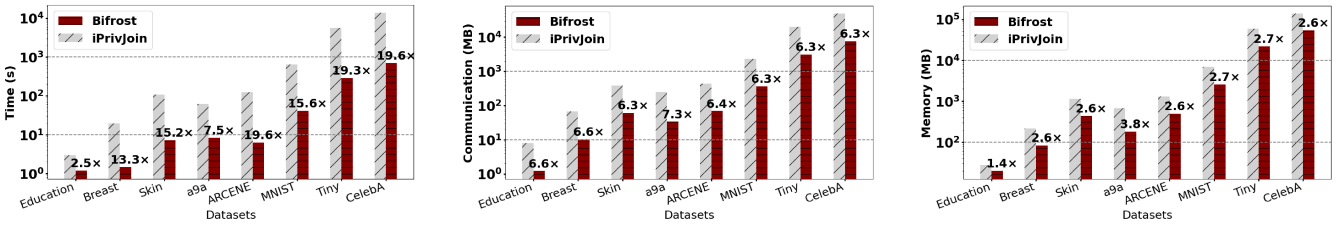
Table 5: Correctness of SDA tasks: Bifrost vs. CPSI [50] compared to performing data analytics on plaintext. ('Education' = Education Career Success, 'Breast' = Breast Cancer Gene).

SDA tasks	Stats. Percent Error		Training Accuracy Bias	
Dataset	Education	Breast	Skin Cancer	a9a
Bifrost	0.00%	0.08%	−0.27%	−0.42%
CPSI [50]	13858.55%	77.20%	−0.36%	−0.50%

6.3 Secure Data Join Evaluation

6.3.1 Evaluation on Real-World Datasets across Three Phases. We compare Bifrost with iPrivJoin [39] across three phases (offline, setup, and online) using eight real-world datasets summarized in Table 3. Table 6 and Figure 6 highlight the key results, summarized as follows:

- In the online phase, Bifrost significantly outperforms iPrivJoin by $2.54 \sim 19.64\times$ in running time. Moreover, Bifrost outperforms iPrivJoin by $1.40 \sim 3.75\times$ in memory cost. The performance gain of Bifrost mainly stems from a reduction of $84.15\% \sim 86.29\%$ in communication size compared to iPrivJoin. Specifically, the inefficiency of iPrivJoin stems from using cryptographic primitive OPPRF and multiple rounds of oblivious shuffle for redundant data removal. In contrast, Bifrost requires a single round of oblivious shuffle while eliminating OPPRF. In addition, Bifrost requires only 4 communication rounds, fewer than the 11 rounds required by iPrivJoin.
- Considering both the setup and online phases, Bifrost achieves an average speedup of 17.59× over iPrivJoin. Despite incurring high computational overhead in the setup phase due to ECC operations, Bifrost maintains superior overall performance.



(a) Time (s) (b) Communication (MB) (c) Memory (MB)

Figure 6: Online Comparison of `Bifrost` vs. `iPrivJoin` [39] on eight real-world datasets.

In addition, considering all three phases, `Bifrost` achieves an average speedup of 2.57× in running time and a 2.59× reduction in communication size. This improvement is primarily attributed to `Bifrost`'s offline phase, which requires less than half of the communication size of `iPrivJoin`.

- The performance advantage of `Bifrost` scales with dataset size: on the largest dataset, CelebA (15 GB), `iPrivJoin` requires 227.20 minutes to complete the secure data join, whereas `Bifrost` completes the secure data join in only 11.57 minutes, yielding a 19.64× speedup in online running time, thereby highlighting its superior efficiency and scalability.

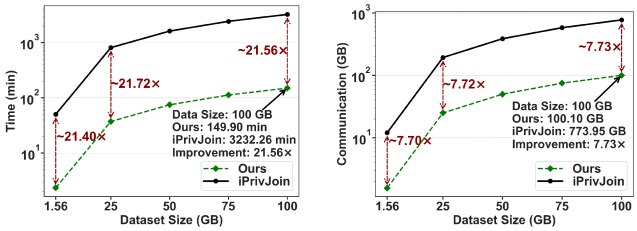
Table 6: Average comparison of `Bifrost` vs. `iPrivJoin` [39] across three phases on eight real-world datasets.

Protocol	Offline		Setup	Online	
	Time (s)	Comm. (MB)	Time (s)	Time (s)	Comm. (MB)
`Bifrost`	**11754.62** (2.39×)	**88307.52** (2.53×)	12.28	**130.86** (19.24×)	**1415.78** (6.34×)
`iPrivJoin` [39]	28059.99	223704.32	0.22	2518.01	8983.04

In the following evaluation across varying parameters, we focus exclusively on the online-phase performance using datasets up to 100 GB. Evaluation across varying parameters for smaller-scale datasets, the offline phase, and unbalanced settings is presented in the full version [12].

6.3.2 Evaluation across Different Row Counts. We conduct experiments on datasets of row count n from 2^{14} to 2^{20} with a fixed feature dimension ($m_a = m_b = 6400$), which corresponds to dataset sizes scaling from 1.56 GB up to 100 GB. As shown in Figure 7, `Bifrost` significantly outperforms `iPrivJoin` by 21.40 ∼ 21.72× in running time and 7.70 ∼ 7.73× in communication size. Furthermore, the runtime advantage of `Bifrost` remains steady at 21.59×, and its communication size reduction remains at about 87.05%. This is because both `Bifrost` and `iPrivJoin` have communication and computation costs that grow linearly in n. Therefore, the relative efficiency advantage remains stable for fixed feature dimensions.

6.3.3 Evaluation across Different Feature Dimensions. We evaluate the effect of feature dimensionality under two settings: (1) We vary the feature dimensions $m_a = m_b$ from 100 to 6400 while fixing the dataset size at $n = 2^{20}$. (2) We vary one party's feature dimension m_* from 100 to 6400, where $* \in \{a, b\}$, while fixing the other party's feature dimension to 100 and the dataset size to $n = 2^{20}$. The corresponding results are presented in Figure 8(a) and Figure 8(b), respectively. We summarize as follows:



(a) Time (min) (b) Communication (GB)

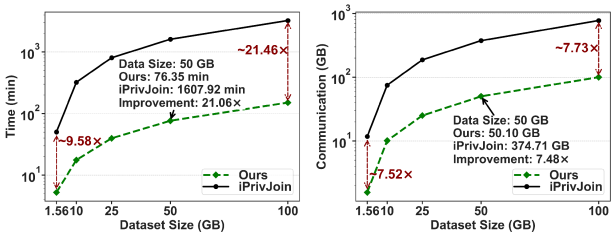
Figure 7: Comparison of `Bifrost` vs. `iPrivJoin` [39] on varying row counts when fixing feature dimension.

- `Bifrost` outperforms `iPrivJoin` by 9.58 ∼ 22.32× in running time and 7.52 ∼ 9.07× in communication size. In addition, the performance improvement of `Bifrost` becomes more pronounced as the feature dimension increases. For instance, as $m_a = m_b$ increases from 100 to 6400, the running time improvement of `Bifrost` over `iPrivJoin` increases from 9.58× to 21.46×. This trend occurs because the computational and communication costs of `iPrivJoin` grow more rapidly with the feature dimension than those of `Bifrost`. Therefore, `Bifrost` exhibits better efficiency when the feature dimension is large.
- `Bifrost` achieves better performance compared to `iPrivJoin`, up to 22.32× when increasing only m_b and up to 19.52× when increasing only m_a. The gap widens as m_b grows because `iPrivJoin` must share P_b's features with P_a via OPPRF, whereas `Bifrost` avoids this bottleneck. Thus `Bifrost` scales more efficiently when one party holds higher-dimensional features.

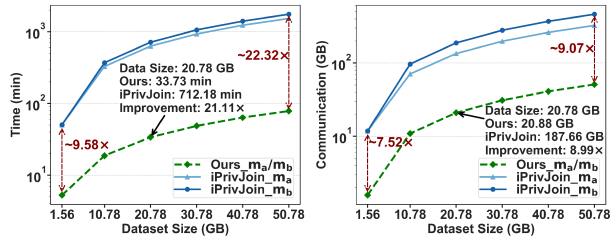
7 DISCUSSION

Supporting Join on Non-Key Attributes. For non-key attributes, there are two cases. In the first case, the attribute is unique within each table. The parties can treat this attribute as the identifiers and directly apply `Bifrost` to perform secure joins. In the second case, the attribute is non-unique, which means multiple records may share the same attribute value. Handling non-unique attributes requires additional steps beyond the scope of this work, and we leave this as future work.

Strengthening the Threat Model. `Bifrost`, which is secure under the semi-honest model, can be strengthened to provide security under a stronger threat model that accounts for specific malicious protocol deviations. We outline the strengthening path of `Bifrost` (comprising Π_{SMIG} and $\Pi_{\text{MI-SFA}}$) toward this stronger threat model



(a) Varying $m_a = m_b$ from 100 to 6400.



(b) Varying m_a (or m_b) from 100 to 6400 when fixing m_b (or m_a). iPrivJoin_m$_a$ and iPrivJoin_m$_b$ denote the performance of `iPrivJoin` when varying m_a and m_b, respectively; Ours_m$_a$/m$_b$ denotes the performance of `Bifrost` under either variation and is plotted as a single curve because m_a and m_b affect `Bifrost` identically.

Figure 8: Comparison of `Bifrost` vs. `iPrivJoin` [39] on varying feature dimensions when fixing row count.

as follows: (i) For Π_{SMIG} (Protocol 3), we can integrate a commitment scheme and Non-Interactive Zero-Knowledge proofs (NIZKs) to ensure that ECC-based ID encrypting is executed correctly, preventing malicious parties from using invalid keys; (ii) For $\Pi_{\text{MI-SFA}}$ (Protocol 4), we can replace its underlying semi-honest oblivious shuffle protocol ($\Pi_{\text{O-Shuffle}}$) with a malicious-secure shuffle protocol [57] to ensure that each party correctly shuffles the other party's features.

8 RELATED WORK

Circuit-based PSI. The initial `CPSI` protocol is introduced by Huang *et al.* [27], which allows two parties, each with input X and Y, to output the secret-shared $X \cap Y$ without revealing any other information. Since the outputs are secret-shared, they can be directly used in subsequent MPC protocols, such as secure model training. In this way, `CPSI` can be employed to perform secure data join. Subsequent research on `CPSI` primarily follows two directions: those based on OPPRF [10, 21, 33, 48, 50, 51] and others based on private set membership [15, 41]. A number of works [24, 42, 55] also focus on designing efficient `CPSI` protocols for unbalanced settings. Despite these innovations, `CPSI` inherently introduces many redundant dummy rows (i.e., secret-shared zeros) in the joined table (padded to the maximum length), other than the actual matched rows. These redundant rows result in significant overhead to downstream secure data analytics tasks in many practical scenarios.

Balanced Secure Data Join Scheme. To address the above limitation in `CPSI`, Liu *et al.* propose a redundancy-free secure two-party data join protocol, `iPrivJoin` [39]. To remove the redundant rows, `iPrivJoin` introduces excessive communication overhead in the secure data join process, which stems from its reliance on OPPRF [51] and multiple rounds of oblivious shuffle.

Unbalanced Secure Data Join Scheme. Real-world scenarios may involve unbalanced datasets [56], where one party's dataset row count is significantly smaller than the other's. Song *et al.* propose the SOTA secure *unbalanced* data join framework `Suda` [56]. They leverage polynomial-based operations to efficiently output (the secret shares of) the redundancy-free joined table. However, it scales inefficiently when directly applied to balanced settings, which are common and foundational settings.

Others. (1) Secure data join schemes [20, 25, 53] for vertical federated learning rely on a Trusted Third Party (TTP), which may impose a strong trust assumption in practical scenarios. (2) Private Set Union (PSU) scheme [58, 65] enables secure data join without a TTP but pads non-overlapping parts with synthetic data, resulting in redundant outputs. (3) Enclave-based scheme [18, 34, 67] executes joins inside trusted hardware execution environments, which can reduce cryptographic overheads but inherit enclave trust.

We summarize representative secure data join schemes in Table 7.

Table 7: High-level comparison of various secure data join schemes with simplified communication complexity. Here, n is the row count of the input data; m_a and m_b are the feature dimensions for P_a and P_b, respectively; $m = m_a + m_b$; ℓ and ℓ' are the bit-lengths of the element and the encrypted element, respectively; λ and κ are the statistical security parameter and the computational security parameter, respectively.

Scheme	Communication complexity	Limitation
Ours	$O(nm\ell)$	-
`iPrivJoin` [39]	$O(nm_b\kappa + nm\ell)$	High communication overhead
`CPSI` [50]	$O(nm_b\kappa + nm\ell)$	Redundant rows
`Suda` [56]	$O(nm\ell')$	Designed for unbalanced setting
FL-Join [25]	$O(nm\ell')$	Relies on a trusted third party
PSU [65]	$O((n + n\log n)m\ell)$	Redundant synthetic rows
Enclave-Join [34]	–	Relies on enclave trust

9 CONCLUSION

In this paper, we propose an efficient and simple secure two-party data join protocol, referred to as `Bifrost`. `Bifrost` outputs (the secret shares of) the redundancy-free joined table. The highlight of `Bifrost` lies in its simplicity: `Bifrost` builds upon two conceptually simple building blocks, an ECDH-PSI protocol and an oblivious shuffle protocol. The lightweight protocol design allows `Bifrost` to avoid many performance bottlenecks in the SOTA redundancy-free secure two-party data join protocol, `iPrivJoin`, including the need for Cuckoo hashing and OPPRF. We also propose an optimization named *dual mapping* that reduces the rounds of oblivious shuffle needed from two to one. Compared to the SOTA redundancy-free protocol `iPrivJoin` [39], `Bifrost` achieves 4.41 $\sim$ 12.96× faster running time and reduces communication size by 64.32% $\sim$ 80.07%. Moreover, experiments on two-step SDA (secure join and secure analytics) show that `Bifrost`'s redundancy-free design avoids error blowup caused by redundant padded outputs (unlike `CPSI`), while delivering up to 2.80× faster secure analytics with up to 73.15% communication reduction compared to `CPSI`.

ACKNOWLEDGMENTS

This work was supported by the National Cryptologic Science Fund of China (Number 2025NCSF01010) and the Natural Science Foundation of China (Number 92370120).

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 2000. Privacy-preserving data mining. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data (SIGMOD '00). 439–450.

[2] Raghad Alharbi. 2022. Breast Cancer Gene Expression Profiles (METABRIC). https://www.kaggle.com/datasets/raghadalharbi/breast-cancer-gene-expression-profiles-metabric/data

[3] D Avraam, R Wilson, O Butters, T Burton, C Nicolaides, E Jones, A Boyd, and P Burton. 2021. Privacy preserving data visualizations. EPJ Data Science 10, 1 (2021), 2–2.

[4] Ahmad Taher Azar and Shereen M El-Metwally. 2013. Decision tree classifiers for automated medical diagnosis. Neural Computing and Applications 23 (2013), 2387–2403.

[5] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. 2017. SMCQL: secure querying for federated databases. Proceedings of the VLDB Endowment 10, 6 (2017), 673–684.

[6] Daniel J Bernstein. 2006. Curve25519: new Diffie-Hellman speed records. In Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9 (PKC'06). Springer, 207–228.

[7] R. Bonta. 2022. California Consumer Privacy Act (CCPA). https://oag.ca.gov/privacy/ccpa Retrieved from State of California Department of Justice.

[8] Nicolas Bruno, YongChul Kwon, and Ming-Chuan Wu. 2014. Advanced join strategies for large-scale distributed computation. Proc. VLDB Endow. 7, 13 (Aug. 2014), 1484–1495.

[9] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In Proceedings of 42nd IEEE Symposium on Foundations of Computer Science (FOCS '01). IEEE, 136–145.

[10] Nishanth Chandran, Divya Gupta, and Akash Shah. 2022. Circuit-PSI with linear complexity via relaxed batch OPPRF. Proceedings on Privacy Enhancing Technologies (2022).

[11] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST) 2, 3 (2011), 27:1–27:27.

[12] Shuyu Chen, Mingxun Zhou, Haoyu Niu, Guopeng Lin, and Weili Han. 2026. Bifrost: A Much Simpler Secure Two-Party Data Join Protocol for Secure Data Analytics. arXiv:2602.01225

[13] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. 2020. Vafl: a method of vertical asynchronous federated learning. arXiv preprint arXiv:2007.06081 (2020).

[14] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. 2021. SecureBoost: A Lossless Federated Learning Framework . IEEE Intelligent Systems 36, 06 (Nov. 2021), 87–98.

[15] Michele Ciampi and Claudio Orlandi. 2018. Combining private set-intersection with secure two-party computation. In Proceedings of the International Conference on Security and Cryptography for Networks. Springer, 464–482.

[16] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A framework for efficient mixed-protocol secure two-party computation.. In Proceedings of 22nd Annual Network and Distributed System Security Symposium (NDSS).

[17] George Drosatos and Pavlos S Efraimidis. 2011. Privacy-preserving statistical analysis on ubiquitous health data. In Proceedings of the International conference on trust, privacy and security in digital business. Springer, 24–36.

[18] Saba Eskandarian and Matei Zaharia. 2019. ObliDB: oblivious query processing for secure databases. Proc. VLDB Endow. 13, 2 (Oct. 2019), 169–183.

[19] David Evans, Vladimir Kolesnikov, Mike Rosulek, et al. 2018. A pragmatic introduction to secure multi-party computation. Foundations and Trends® in Privacy and Security 2, 2-3 (2018), 70–246.

[20] Ying Gao, Yuxin Xie, Huanghao Deng, Zukun Zhu, and Yiyu Zhang. 2024. A Privacy-preserving Data Alignment Framework for Vertical Federated Learning. Journal of Electronics & Information Technology 46, 8 (2024), 3419–3427.

[21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2021. Oblivious key-value stores and amplification for private set intersection. In Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41. Springer, 395–425.

[22] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. 2004. Arcene. UCI Machine Learning Repository. https://archive.ics.uci.edu/dataset/167/arcene

[23] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. 2004. Guide to elliptic curve cryptography. Springer Science & Business Media.

[24] Meng Hao, Weiran Liu, Liqiang Peng, Hongwei Li, Cong Zhang, Hanxiao Chen, and Tianwei Zhang. 2024. Unbalanced Circuit-PSI from Oblivious Key-Value Retrieval. In Proceedings of 33rd USENIX Security Symposium (USENIX Security 24). 6435–6451.

[25] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. 2017. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. arXiv preprint arXiv:1711.10677 (2017).

[26] Daojing He, Runmeng Du, Shanshan Zhu, Min Zhang, Kaitai Liang, and Sammy Chan. 2022. Secure Logistic Regression for Vertical Federated Learning. IEEE Internet Computing 26, 2 (2022), 61–68.

[27] Yan Huang, David Evans, and Jonathan Katz. 2012. Private set intersection: Are garbled circuits better than custom protocols?. In Proceedings of 19th Annual Network and Distributed System Security Symposium (NDSS).

[28] Bernardo A Huberman, Matt Franklin, and Tad Hogg. 1999. Enhancing privacy and trust in electronic communities. In Proceedings of the 1st ACM conference on Electronic commerce (EC '99). 78–86.

[29] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20).

[30] Florian Kerschbaum, Erik-Oliver Blass, and Rasoul Akhavan Mahdavi. 2023. Faster Secure Comparisons with Offline Phase for Efficient Private Set Intersection. In Proceedings of 30th Annual Network and Distributed System Security Symposium (NDSS). The Internet Society, San Diego, California, USA.

[31] Bas Ketsman and Dan Suciu. 2017. A Worst-Case Optimal Multi-Round Algorithm for Parallel Computation of Conjunctive Queries. In Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '17). Association for Computing Machinery, 417–428.

[32] Neal Koblitz. 1987. Elliptic curve cryptosystems. Mathematics of computation 48, 177 (1987), 203–209.

[33] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. 2017. Practical multi-party private set intersection from symmetric-key techniques. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). 1257–1272.

[34] Simeon Krastnikov, Florian Kerschbaum, and Douglas Stebila. 2020. Efficient oblivious database joins. Proc. VLDB Endow. 13, 12 (July 2020), 2132–2145.

[35] Yann Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. CS 231N 7, 7 (2015), 3.

[36] LeCun, Yann and Cortes, Corinna and Burges, Christopher J.C. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/ Accessed: 2016-07-14.

[37] Guopeng Lin, Ruisheng Zhou, Shuyu Chen, Weili Han, Jin Tan, Wenjing Fang, Lei Wang, and Tao Wei. 2025. Kona: An Efficient Privacy-Preservation Framework for KNN Classification by Communication Optimization. In Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research), Vol. 267. PMLR, 37996–38012.

[38] Yehuda Lindell. 2020. Secure multiparty computation. Commun. ACM 64, 1 (2020), 86–96.

[39] Yang Liu, Bingsheng Zhang, Yuxiang Ma, Zhuo Ma, and Zecheng Wu. 2023. iPrivJoin: An ID-Private Data Join Framework for Privacy-Preserving Machine Learning. IEEE Transactions on Information Forensics and Security 18 (2023), 4300–4312.

[40] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15). IEEE Computer Society, 3730–3738.

[41] Jack PK Ma and Sherman SM Chow. 2022. Secure-computation-friendly private set intersection from oblivious compact graph evaluation. In Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '22). 1086–1097.

[42] Rasoul Akhavan Mahdavi, Nils Lukas, Faezeh Ebrahimianghazani, Thomas Humphries, Bailey Kacsmar, John Premkumar, Xinda Li, Simon Oya, Ehsan Amjadian, and Florian Kerschbaum. 2024. PEPSI: Practically Efficient Private Set Intersection in the Unbalanced Setting. In Proceedings of 33rd USENIX Security Symposium (USENIX Security 24). 6453–6470.

[43] Catherine Meadows. 1986. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In Proceedings of the 1986 IEEE Symposium on Security and Privacy. IEEE, 134–134.

[44] Victor S Miller. 1985. Use of elliptic curves in cryptography. In Conference on the theory and application of cryptographic techniques. Springer, 417–426.

[45] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In 2017 IEEE symposium on security and privacy (SP). IEEE, 19–38.

[46] Goldreich Oded. 2009. Foundations of cryptography: Volume 2, basic applications.

[47] Anifat Olawoyin, Carson Leung, and Alfredo Cuzzocrea. 2021. Privacy-Preserving Publishing and Visualization of Spatial-Temporal Information. In Proceedings of 2021 IEEE International Conference on Big Data (Big Data). IEEE, 5420–5429.

[48] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. 2019. Efficient circuit-based PSI with linear communication. In Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38. Springer, 122–153.

[49] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph M Hellerstein. 2021. Senate: a Maliciously-Secure MPC platform for collaborative analytics. In Proceedings of 30th USENIX Security Symposium

(USENIX Security 21). 2129–2146.
[50] Srinivasan Raghuraman and Peter Rindal. 2022. Blazing Fast PSI from Improved OKVS and Subfield VOLE. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22). 2505–2517.
[51] Peter Rindal and Phillipp Schoppmann. 2021. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part II. Springer, 901–930.
[52] Wolf Rödiger, Sam Idicula, Alfons Kemper, and Thomas Neumann. 2016. Flow-Join: Adaptive Skew Handling for Distributed Joins Over High-Speed Networks. In Proceedings of the 32nd IEEE International Conference on Data Engineering. IEEE Computer Society, 1194–1205.
[53] Rainer Schnell, Tobias Bachteler, and Jörg Reiher. 2011. A novel error-tolerant anonymous linking code. Available at SSRN 3549247 (2011).
[54] Adil Shamim. 2024. Education and Career Success. https://www.kaggle.com/datasets/adilshamim8/education-and-career-success/data
[55] Yongha Son and Jinhyuck Jeong. 2023. PSI with computation or Circuit-PSI for Unbalanced Sets from Homomorphic Encryption. In Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security (ASIA CCS '23). 342–356.
[56] Lushan Song, Qizhi Zhang, Yu Lin, Haoyu Niu, Daode Zhang, Zheng Qu, Weili Han, Jue Hong, Quanwei Cai, and Ye Wu. 2025. Suda: An Efficient and Secure Unbalanced Data Alignment Framework for Vertical Privacy-Preserving Machine Learning. In Proceedings of 34th USENIX Security Symposium (USENIX Security 25). 7663–7682.
[57] Xiangfu Song, Dong Yin, Jianli Bai, Changyu Dong, and Ee-Chien Chang. 2024. Secret-Shared Shuffle with Malicious Security. In Proceedings of 31st Annual Network and Distributed System Security Symposium (NDSS).
[58] Jiankai Sun, Xin Yang, Yuanshun Yao, Aonan Zhang, Weihao Gao, Junyuan Xie, and Chong Wang. 2021. Vertical federated learning without revealing intersection membership. arXiv preprint arXiv:2106.05508 (2021).
[59] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. 2021. CryptGPU: Fast privacy-preserving machine learning on the GPU. In Proceedings of 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 1021–1038.
[60] Yongxin Tong, Xuchen Pan, Yuxiang Zeng, Yexuan Shi, Chunbo Xue, Zimu Zhou, Xiaofei Zhang, Lei Chen, Yi Xu, Ke Xu, et al. 2022. Hu-fu: Efficient and secure spatial queries over data federation. Proceedings of the VLDB Endowment 15, 6 (2022), 1159.
[61] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. 2018. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Scientific data 5, 1 (2018), 1–9.
[62] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). A Practical Guide, 1st Ed., Cham: Springer International Publishing (2017).
[63] Kang Wei, Jun Li, Chuan Ma, Ming Ding, Sha Wei, Fan Wu, Guihai Chen, and Thilina Ranbaduge. 2022. Vertical federated learning: Challenges, methodologies and experiments. arXiv preprint arXiv:2202.04309 (2022).
[64] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy preserving vertical federated learning for tree-based models. Proceedings of the VLDB Endowment 13, 12 (2020), 2090–2103.
[65] Jianping Yan, Lifei Wei, Xiansong Qian, and Lei Zhang. 2025. IDPriU: A two-party ID-private data union protocol for privacy-preserving machine learning. Journal of Information Security and Applications 88 (2025), 103913.
[66] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In Proceedings of 27th annual symposium on foundations of computer science (Sfcs 1986) (SFCS '86). IEEE, 162–167.
[67] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In Proceedings of 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17) (NSDI'17). USENIX Association, Boston, MA, 283–298.